TUM School of Computation, Information and Technology
Technische Universität München

Solving Stochastic Games Reliably

Maximilian Philipp Weininger

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:
Prof. Dr. Helmut Seidl

Prüfende der Dissertation:
1. Prof. Dr. Jan Křetínský
2. Prof. Dr. Jean-François Raskin,
   Université Libre de Bruxelles

Die Dissertation wurde am 22.06.2022 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 31.10.2022 angenommen.

Abstract

By stochastic games, we understand zero-sum games played on a stochastic graph between two antagonistic players. Among the many uses of these games are verification of and strategy synthesis for reactive systems, multistage optimization problems, and robust decision-making under uncertainty. Applications include economics, IT security, medicine, robot motion planning, self-adaptive systems and autonomous driving.

Solving a stochastic game amounts to computing optimal strategies for the players as well as the resulting value of the game. We are interested in making solution methods **reliable** in the following sense.

- **Expressive formalisms:** When translating a real world problem to a mathematical model, we require formalisms that are expressive enough to capture all relevant details of the problem. In particular, we consider settings with limited information, making the resulting solutions more robust than when assuming precise knowledge; and we allow to model problems with multiple possibly conflicting objectives and compute optimal trade-offs.

- **Guaranteed solution algorithms:** All solution algorithms should give guarantees on their precision. Depending on the formalisms, we provide guarantees in the form of proving that the resulting value and strategies are indeed optimal, ε-optimal or probably approximately correct (PAC).

- **Explainable result:** Even when using guaranteed solution algorithms, the result is only correct under the assumption that the model is correct. To be reliable, we cannot blindly trust the modelling process, which is typically based on simulations or carried out by humans. Thus, we require the result to be explainable so that we can validate it and thereby the model.

Orthogonally, our solution methods have to be **scalable**. This means that the algorithms can not only solve small examples but rather scale to realistically large case studies.

In this thesis, we take a significant step towards solving stochastic games reliably. In particular, we provide reliable solution algorithms for simple stochastic games and both analyse and improve their scalability. We show how to extend the algorithms to give guarantees also when the formalisms include limited information or multiple objectives. We present a mature tool that supports explaining any memoryless pure strategy, hence allowing to validate results. Finally, we demonstrate the applicability of our techniques on a case study from IT security.

Zusammenfassung

Als stochastische Spiele verstehen wir Nullsummenspiele, die auf einem stochastischen Graph zwischen zwei Antagonisten gespielt werden. Zu den vielen Nutzungsmöglichkeiten dieser Spiele gehören Verifikation und Strategiesynthese für reaktive Systeme, mehrstufige Optimierungsprobleme, sowie robuste Entscheidungsfindung im Angesicht von Unsicherheit. Ihre Anwendungen erstrecken sich von Wirtschaftslehre über Sicherheit von informationstechnischen Systemen, Medizin, Bewegungsplanung von Robotern und Selbstregulierung von Systemen bis hin zu selbstfahrenden Autos.

Ein stochastisches Spiel zu lösen bedeutet, optimale Strategien und den resultierenden Wert des Spiels zu berechnen. Unsere Lösungsmethoden sollten **vertrauenswürdig** in folgendem Sinne sein.

- **Ausdrucksstarke Formalismen:** Beim Übersetzen eines Problems in ein mathematisches Modells benötigen wir Formalismen, die ausdrucksstark genug sind, um alle relevanten Details des Problems abzubilden, wobei wir insbesondere Gegebenheiten, die robustere Ergebnisse dadurch, dass wir nur begrenzte Informationen annehmen, erlauben, betrachten. Weiterhin untersuchen wir Gegebenheiten mit mehreren, möglicherweise gegensätzlichen Zielen und berechnen optimale Kompromisslösungen.

- **Lösungsalgorithmen mit Garantien:** Alle Lösungsalgorithmen sollten Garantien auf die Präzision ihrer Lösungen geben. Abhängig von den verwendeten Formalismen geben wir solche Garantien in Form eines Beweises der Optimalität, der ε-Optimalität oder der wahrscheinlichen, annähernden Korrektheit.

- **Erklärbare Ergebnisse:** Selbst bei der Nutzung von Lösungsalgorithmen mit Garantien baut die Korrektheit des Ergebnisses auf der Korrektheit des Modells auf. Allerdings können wir für die Vertrauenswürdigkeit eben dieses Ergebnisses dem Modellierungsprozess, der üblicherweise von Menschen durchgeführt wird, nicht blind vertrauen. Stattdessen müssen die Ergebnisse erklärbar sein, damit wir sie und damit auch das Modell validieren können.

Unabhängig davon sollten unsere Lösungsmethoden **skalierbar** sein. Das bedeutet, dass die Algorithmen nicht nur kleine Beispiele, sondern auch realisitische, große Fallstudien lösen können.

In dieser Doktorarbeit gehen wir einen erheblichen Schritt in Richtung des vertrauenswürdigen Lösens stochastischer Spiele. Wir beschreiben vertrauenswürdige Algorithmen zum Lösen einfacher, stochastischer Spiele, analysieren ihre Skalierbarkeit und verbessern diese sogar. Wir zeigen, wie man diese Algorithmen und ihre Fähigkeit, Garantien zu geben, auf Formalismen mit begrenzter Information und mehreren Zielen erweitert. Wir präsentieren ein Programm, das beim Erklären jedweder gedächtnislosen deterministischen Strategie unterstützt und damit die Validierung von Ergebnissen erlaubt. Zuletzt demonstrieren wir die Anwendbarkeit unserer Techniken an einer Fallstudie aus dem Feld der Sicherheit von informationstechnischen Systemen.

Acknowledgements

Many people helped me in various ways, making my time as a PhD-student both successful and fun. First and foremost, I want to thank all my co-authors: Tobi Meggendorfer, Clara Waldmann, Edon Kelmendi, Julia Eisentraut, Pranav Ashok, Kim G. Larsen, Adrian Le Coënt, Jakob Haahr Taankvist, Mathias Jackermeier, Pushpak Jagtap, Majid Zamani, Krishnendu Chatterjee, Tobias Winkler, Loris D'Antoni, Martin Helfrich, Emanuel Ramneantu, Joost-Pieter Katoen, Alexander Slivinskiy, Przemek Daca, Christoph Weinhuber, Mayank Yadav, Javier Esparza, Stefan Kiefer, Kush Grover, Shruti Misra and Jan Kretinsky. I enjoyed collaborating with you all and I am proud of the publications we produced together! In particular, I am grateful to the following people.

- Pranav, for growing up together with me. Always there to help me with any question, no matter how stupid. I feel like we really got out the best in each other, and I miss working with you.
- Both Tobis, who are so frighteningly smart and good at proofs. My heuristic for judging the validity of a statement is giving it to either Tobi and if they say "Yea, seems ok, here is a proof sketch", then the statement is true.
- My students Mathias, Emanuel, Sasha, and Christoph, for being excited about our research and investing way more time than they had to.
- Edon, who patiently stood in front of a whiteboard with me and taught me how to think about a proof. The idea of deflating that is used in so many of my publications originated from such a discussion.
- Martin, who, just like me, accidentally became a project lead of Automata Tutor. We embarked on this quest together, we helped Automata Tutor fulfil its destiny to support teachers and students, and we keep defending it from the evil hordes of technical problems.
- Krish, for giving me a whole new perspective on how to think about research.
- Javier, for being my mentor and having time for me whenever I needed it.
- Jan, for (i) believing in me more than I do and giving me so much confidence, (ii) always treating me as an equal, even when I was a Master's student who had forgotten how Parity acceptance works, (iii) teaching me the neat trick with the itemize inside a sentence to put more information into it than I thought was possible (and of course so many other useful things), and lastly (iv) helping me improve everything that I ask him to, be it a wild idea that we brainstorm about or the finalfinal$^{\text{TM}}$ version of the abstract that we have to submit in 20 minutes.

Further, I want to express my gratitude to people that influenced me on my path before even starting my PhD: my teacher Herr Janocha, who got me into the field of Informatics by telling me: "Of all the things I studied, this is the one where I really felt like I was getting smarter." Bernhard Küster, for believing in me and recruiting me into his team during his son's birthday party. Mathias Wilhelm, who taught me so much about working independently so we can work well together, and who made me see the big value in the little steps.

This dissertation was reviewed by my peers, namely Pranav Ashok, Muqsit Azeem and Christoph Weinhuber. Also, Konstantina Drainas helped me with all the annoying, time-consuming things like checking rules for commas or that all links in the bibliography are working. Thanks to all of you for taking the time to read this enormous document! In a similar vein: all papers in this thesis are peer-reviewed, and several reviewers read the papers in detail and gave very insightful feedback. I want to thank these anonymous reviewers for doing their job to such a high standard that it noticeably improved the papers.

I am also grateful to the people of our chair who made it fun when (if) I was there. I enjoyed our lunches and coffee breaks, and I'm proud to be a founding member of our slow-eating, late lunch group.

I want to thank my friends and family, for giving me the privilege of a wonderful childhood and support system. I can just follow my dreams because I know you are there to catch me whenever I fall.

Finally, I want to thank my wife, who is smart and beautiful. You always push me when I need motivation and calm me when I'm working too much — even though I usually have no idea which of the two I currently need. *Ευχαριστώ πολύ, μπισκότο μου.* Kartoffelreh.

Contents

List of Figures

List of Tables

Acronyms

APT	Advanced Persistent Threats	Page 101
BDD	Binary Decision Diagram	Page 96
BFS	Basic Feasible Solution	Page 67
BSCC	Bottom Strongly Connected Component	Page 21
bMC	bounded-parameter Markov Chain	Page 64
bMDP	bounded-parameter Markov Decision Process	Page 64
bSG	bounded-parameter Stochastic Game	Page 63
BVI	Bounded Value Iteration	Page 30
DIFT	Dynamic Information Flow Tracking	Page 101
DT	Decision Tree	Page 89
EC	End Component	Page 21
HOP	Higher Order Program	Page 38
IMC	Interval Markov Chain	Page 72
LP	Linear Programming	Page 39
MDP	Markov Decision Process	Page 17
MEC	Maximal End Component	Page 21
PAC	Probably Approximately Correct	Page 49
QP	Quadratic Programming	Page 37
SCC	Strongly Connected Component	Page 20
SEC	Simple End Component	Page 29
SG	Stochastic Game	Page 15
SI	Strategy Iteration	Page 34
SMC	Statistical Model Checking	Page 50
VI	Value Iteration	Page 24

1 Introduction

1.1 Motivation

"How should an agent behave to achieve its goal?"

Many questions in computer science can be phrased in this form. The agent can be very concrete, such as an autonomous car or robot, trying to reach a destination while avoiding obstacles [AD14]; or rather abstract, like an algorithm analysing "big data" to decide whether to grant a credit [HA16]. The agent's goals can be easy to formulate, for example beating a human grand master in chess [Has17b]; or they can be complicated multi-faceted problems where not even humans know exactly how to behave, such as an IT security algorithm with the goal of preventing data-theft [Col12] or a medical device deciding whether a patient is healthy based on some monitored parameters [SBS20]. Computer systems affect our everyday lives but also perform high-impact, critical decision-making.

All these systems share the central idea of how to answer the given question: they construct a mathematical model of the real world, formalising both the interaction of the agent with its environment as well as the goal of the agent. Then, in this mathematical world, a solution to the question is computed. This prescribes the behaviour of the agent in the real world.

Often, the focus lies on getting an acceptable solution quickly, instead of computing an optimal one. Moreover, many algorithms do not quantify how good the solution is, so it can in fact be arbitrarily bad. In such a case, we should only take the decisions of the system as a suggestion and must not rely on them. However, on the one hand, humans are accustomed to trusting the systems as they usually perform well; and on the other hand, many systems are not monitored, even if they perform critical tasks. This reliance on algorithms without guarantees on their solutions has led to disastrous consequences, costing millions of dollars or even human lives, with well-known examples being crashes of the Ariane-5 missile or aeroplanes of the airbus family [BK08, Chapter 1]. The danger posed by unreliable systems is aggravated by our society's increased dependency on black-box machine learning algorithms for high-stakes decisions [Rud19].

Thus, especially for decisions that carry high risk, we cannot rely on best-effort heuristics or blindly trust computer systems. We have to use methods whose results are **reliable**. This is the focus of the *formal verification* community in general and this dissertation in particular.

1.2 Challenges and solutions

The central challenge of this thesis is to make the methods we use more reliable. We differentiate three core components of **reliability**.

- **Expressive formalisms:** When formalising the real world using a mathematical model, we usually make assumptions and lose information. Still, we want to transfer the result of the mathematical model back to the real problem. Thus, we have to make sure that the formalisms we use in our model are expressive enough to capture all relevant details. This is called "*The Modeling Challenge*" in the handbook of model checking [CHVB18, Section 1.3.2].

  In particular, assuming that we know the precise behaviour of the environment that the agent interacts with often is unrealistic. Thus, we propose methods that work in a setting of *limited information* in Chapter 4. Additionally, the agent's goal often consists of multiple possibly conflicting objectives. In Chapter 5, we deal with such goals. For a broad overview of other more expressive formalisms and their relation to the thesis, we refer to Section 1.5.

- **Guaranteed solution algorithms:** The algorithms solving the mathematical model should give guarantees on the precision of their result. Only then can we rely on it being optimal or at least good enough. All of the algorithms we propose in this thesis (Chapters 3-7) provide provable guarantees.

- **Explainable result:** Even when using solution methods that are reliable in the sense of the previous two points, the result we get is guaranteed to be correct *only under the assumption that the model is correct.* This assumption is quite strong since the modelling process typically involves humans. They are likely to make mistakes, especially since it is difficult to use the formal, mathematical language that is necessary to construct a model.

  Thus, we do not want to blindly trust the model. Instead, if we can explain and *validate* the result, we can certify that it behaves as expected and hence the model was correct. Making the resulting behaviour explainable is the main focus of Chapter 6. Additionally, we show how a good choice of formalisms helps to understand the results for an IT security application in Chapter 7.

Orthogonally — or even contrary — to the challenge of reliability, we also have to address the **challenge of scalability**. The real world is complex and the mathematical models describing it are very large, a problem often called *state-space explosion* [BK08, Section 2.3]. Thus, the solution algorithms have to scale, i.e. be performant even for very large instances. This is also called "*The Algorithmic Challenge*" in the handbook of model checking [CHVB18, Section 1.3.1].

Methods from the field of *machine learning* focus only on this challenge, being able to scale to enormous system sizes. However, this comes at cost of being utterly unreliable: it is often not clear that the formalism really models the intended problem; the algorithms give no guarantees, neither how close they are to the optimum nor that the result is good in all cases; and the results often are black-boxes operating in ways beyond our understanding.

Our focus on reliability often impedes scalability, as formalisms become more complex and algorithms more conservative. Still, there are many ways in which the formal methods community has achieved scaling its methods to industry problems, many of which are summarized in [CHVB18, Section 1.3.1]. One recent direction not mentioned in that summary is to combine formal verification and machine learning, obtaining the advantages of both. The machine learning algorithms allow to quickly get a good guess of how to behave. Formal verification can use this guess as a basis for its computation, speeding up the process. If the guess was not good enough, the verification can also advise the machine learning algorithms on what to focus on before we repeat the process. This general idea has been applied in many settings, see e.g. [JKKK18] for a recent report.

In this thesis, we utilize such a combination of machine learning and formal methods in Section 3.1.3. Moreover, algorithms can be more scalable by weakening the guarantees. Instead of computing a provable optimum, we can provide a *statistical guarantee* of optimality, as we discuss in Section 4.1. Similarly, requiring an approximate guarantee allows for the usage of so-called anytime algorithms, which at any time during the computation can quantify the quality of their approximation and thus allow earlier termination; the algorithms in Section 3.1 and 5 are of this type.

Finally, we mention that the solutions prescribing the behaviour of an agent are often very large, as they scale with the problem size. However, they often have to be executed by embedded devices with limited memory. The contribution of Chapter 6 not only helps with explaining these solutions, but also compresses them and thus reduces the required memory.

1.3 Stochastic games

So far, we have argued that it is important to solve questions of the form "How should an agent behave to achieve its goal", and that our solution methods have to be both *scalable* and *reliable*.

In this thesis, we focus on questions that can be modelled as zero-sum **stochastic games**. These are games of two antagonistic players where one player winning implies that the other player loses. Slightly more formally, this very expressive mathematical modelling formalism includes

- controllable nondeterminism: our behaviour affects the state of the environment.
- adversarial nondeterminism: there are other agents that affect the state of the environment. By assuming they are adversarial and do everything in their power to hinder us, we can guarantee that the result of our analysis is safe even in the worst case. See e.g. [CHVB18, Chapter 27] for more motivation and applications.
- stochasticity: the outcome of the actions of agents is not fixed, but probabilistic, e.g. because a dice is rolled or a measurement is taken which is expected to fail with some probability. See e.g. [BK08, Chapter 10] or [CHVB18, Chapter 28] for more information.

Stochastic games are **practically useful**. As they are so expressive, they have been used in many fields, e.g. economics [Ami03], IT security [RES$^+$10], medicine [BD08], robot motion planning [LaV00], self-adaptive systems [CMG14] and autonomous driving [CKSW13]. We refer the interested reader to [SK16] for more case studies and to [Kwi16] for an overview of how to go from a practical problem to a stochastic game. Moreover, stochastic games generalize the well-known and often applied models of Markov chains [Nor98], deterministic games [CHVB18, Chapter 27] and Markov decision processes [Put94]. As a final practical motivation, we mention that they can serve as abstractions of large Markov decision processes [KKNP10] or robust variants of Markov decision processes in settings with limited information, see Section 4.2.

Stochastic games are also very interesting from a **complexity theoretical perspective**. For an overview of complexity classes and further references, see [Aar16, AB]. *Simple stochastic games*, which are stochastic games with the objective to reach a goal state, are in the complexity class UP ∩ co-UP [CF11, Theorem 4], a subset of NP ∩ co-NP. The latter class is of particular interest for complexity theorists as quite possibly there exists a polynomial algorithm for problems in this class [Joh07]. For a discussion on the efforts of finding such an algorithm for simple stochastic games, see Section 3.5.

Simple stochastic games are **fundamental** for another reason, namely that many variants of stochastic games with different objectives can be polynomially reduced to simple stochastic games [AM09, CF11]. Thus, in this thesis, we first focus on reliably solving simple stochastic games (Chapter 3) before making them more expressive (Chapters 4 and 5).

1.4 Which parts of this PhD-thesis do you want to read?

This dissertation contains many hours' worth of reading. Going through everything is a daunting task, especially if you, dear reader, have a particular motivation so that not all parts are of interest to you. Hence, I try to give you a good, modular structure and clear organizational paragraphs throughout the whole thesis. This should help you to decide which parts you want to read in order to achieve your goal, whatever that might be. I hope this thesis can be of assistance and I wish you the best of luck!

Outline: In the previous Sections 1.1- 1.3 we have motivated the contributions from different directions and referenced some sections of the main content. For a more rigid outline adhering to the table of contents, the next Section 1.5 provides the big picture and embeds the chapters of the thesis in their order of appearance.

Chapter structure: Every chapter starts with a short motivation, a clear problem statement and the chapter outline. Then every part inside the chapter is ordered as: state of the art — contribution — related work. The sections on the state of the art and contribution are at a rather high level, focussing on intuition and examples. For detailed formal descriptions and proofs, the sections also provide pointers to particular parts of the full papers in the appendix. The related work sections are quite extensive. They contain historical developments, inspiring alternative ideas as well as citations to papers that can serve as an introduction to broadly related research. Finally, every chapter ends with a summary, a discussion and an outlook.

Reader's motivation: This thesis is optimized for two different key motivations.

- If you are looking for an introduction to any of the core topics of the thesis, read the motivation, problem statement and state of the art and contribution sections. This should provide the central intuition and, if desired, enable you to find the relevant technical details in the full papers in the appendix.

Note that, apart from a dependency on the preliminaries, all chapters are self-contained, allowing you to immediately jump to the topic of interest.

- If you are looking for the relation of this research with other works (be it the bigger context or an idea for a research direction), you can focus on the related work, discussion and outlook sections.

  For the key ideas of a chapter, it suffices to read the motivation and problem statement at its beginning together with the summary at its end.

1.5 The big picture

In this section, we provide the relation of this thesis to many related research directions. We categorize contributions according to the underlying paradigm that they all have in common, as depicted in Figure 1.1. Given the complex real-world problem we want to solve as input, we first formalise it as a mathematical model. We differentiate between (i) the system, in our case usually a stochastic game, which describes the environment and how the agents can affect it and (ii) the specification, the encoding of the goal of the agent. The solution algorithms and their output differ for the various combinations of formalisms describing system and specification.

Figure 1.1 is a variation of the model checking methodology as given in [BK08, Figure 1.4] or [CHVB18, Figure 1]. This model checking approach has several classic advantages and disadvantages, as described in [BK08, Section 1.2.2] or [CHVB18, Section 1.1]. However, we emphasize that this underlying paradigm also occurs in fields other than formal verification: for example, reinforcement learning requires a way to execute the system (called Agent-Environment Interface in [SB18, Section 3.1]) and a well-designed reward signal encoding the specification [SB18, Section 17.4].

Our outline adheres to the figure as follows.

- Algorithms (Chapter 3): We first describe algorithms for the most fundamental formalism we use in this thesis, namely simple stochastic games. Based on this understanding, we describe the algorithmic extensions which are necessary for dealing with more expressive formalisms in later chapters.

- System (Chapter 4): We add the possibility to encode systems about which we only have limited information.

- Specification (Chapter 5): We add the possibility that our specification includes multiple possibly conflicting objectives.

- Output (Chapter 6): We describe methods for compressing and explaining the result of our algorithms while retaining its guarantees.

- Input (Chapter 7): We show the importance of choosing the right formalism for modelling the given input, using an example of an application to IT security.

In the rest of this section, we go through the parts of the paradigm in the same order as the overall outline, providing more context for our contribution, pointing to the related work sections of the chapters and more high-level related works.

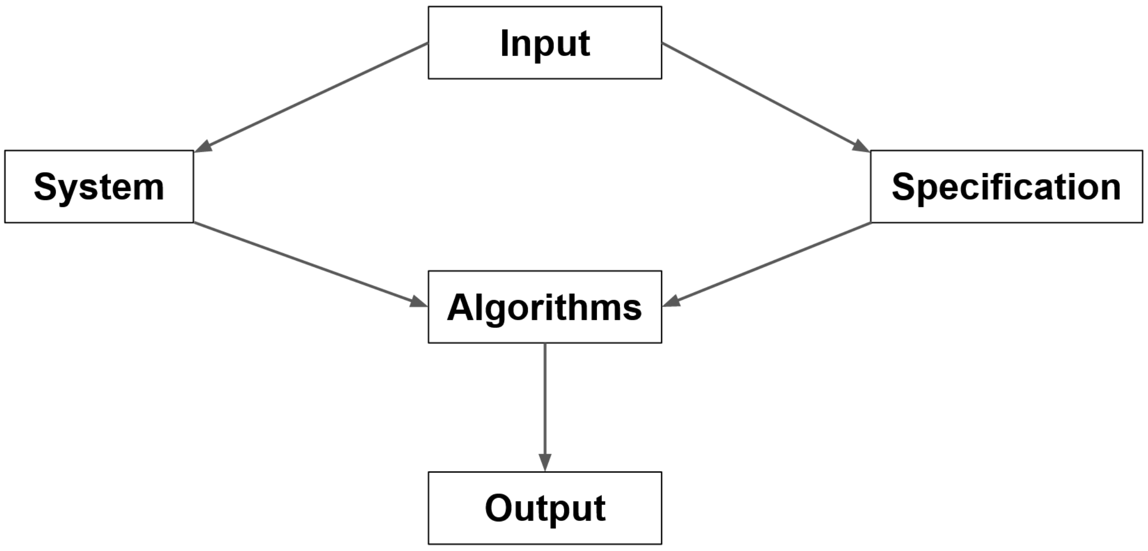


Figure 1.1: The underlying paradigm of the approaches considered in this thesis and many related works, which also inspired the outline of the thesis.

1.5.1 Algorithms

Guarantees All algorithms in this thesis give guarantees on their precision, addressing the second part of the challenge of reliability. For the basic algorithms, we discuss their guarantees in Section 3.5. They are either precise or approximate, quantifying the distance to the optimum. Chapter 5 also provides such an approximate guarantee. In Section 4.1, we give a probably approximately correct (PAC) guarantee; this is a statistical guarantee, claiming that with a high probability the result is correct. In Section 4.2 and Chapter 6, our contribution is independent of the exact solution algorithm, but it allows to retain whichever guarantees that algorithm had. In Chapter 7, our algorithm only gives a guarantee that it converges in the limit, which is insufficient to be reliable. However, we point out the necessary future steps to allow for quantifying the distance to the optimum.

Scalability We complement the discussion of the challenge of scalability in Section 1.2 with more related work: apart from the already mentioned overviews in [CHVB18, Section 1.3.1] and [JKKK18], we point to the description of abstraction in [BK08, Chapter 7 and 8], the original description of symbolic model checking [CMCH96], the seminal paper proposing guaranteed algorithms based on partial exploration [BCC$^+$14] and the recent overview in [BHK19, Chapter 5].

Note that the more expressive our formalisms are, the harder the problems typically become. This is visible in an increase in complexity: reachability in Markov decision processes is solvable in polynomial time [Put94] while solving simple stochastic games is in NP $\cap$ co-NP [Con92]. Potentially, the problems we study even become undecidable, for example when considering partially observable Markov decision processes [MHC03] or timed or hybrid systems [HR00]. Still, there might be good heuristics even for problems that are undecidable or that have a prohibitive theoretical worst-case complexity.

In Chapter 3, we give details on the developments of the main algorithms for solving stochastic games, namely value iteration, strategy iteration and quadratic programming, in Sections 3.1.4, 3.2.3 and 3.3.3, respectively. The discussion in Section 3.5 provides a theoretical and practical comparison.

Tools and practical implementations There are many practical challenges to overcome when transferring the reliable algorithms from formal verification to industrial practice. Such challenges and ways to deal with them (as well as ways not to do it) are discussed in [CHVB18, Chapter 23], and the software architecture of modern model checkers is analysed in [KLvdPT19]. Moreover, we point out that there are many tools that allow for implementing and testing algorithms, such as SCOTS [RZ16] and Uppaal [LPY97] for hybrid systems or the probabilistic model checkers PRISM [KNP11, KNPS20], Storm [DJKV17], the MODEST toolset [HH14] and others, see [CHVB18, Section 28.7.1]. Moreover, there are collections of benchmarks [KNP12, HKP$^+$19] and competitions to encourage practical improvements, for example QComp [HHH$^+$19a, BHK$^+$20] or SYNTCOMP [JBB$^+$17, JPA$^+$22].

1.5.2 System

We refer to [SV15] for a short summary of the historical development of stochastic game theory and to [NS03, FV97] for extensive books on stochastic games.

We mention four directions in which a system formalism can be made more expressive: the first two, limited information and infinite systems, can be applied to games as well as other formalisms, while the latter two, number of players and concurrency, are relevant only in games.

Limited information In Chapter 4, we consider two variations of limited information. This is practically relevant because we cannot assume that we always have information about the full topology of the system, let alone precise knowledge of all transition probabilities [JL91, Wal96, KU02, GLD00]. Moreover, limited information can serve as an abstraction of continuous systems [KKLW12, LAB15] or allow to make the resulting decision making more robust [WTM12].

On the one hand, we study a setting where we can only simulate the system and observe its behaviour; on the other hand, we assume that instead of precise knowledge about the probabilities occurring in the system, we know intervals on these probabilities. We discuss variants of these settings in Sections 4.1.5 and 4.2.4. The latter section also contains an overview of other settings with limited information.

Infinite systems In this thesis, we only look at systems with a finite, discrete state-space. The following formalisms have infinite state-space, but some way of compactly representing the system in order to still reason about its behaviour.

In one direction, there are countably infinite games with recursion [EY15]. Another way to make games countably infinite is to add lossy channels [AHdA$^+$08].

In another direction, we want to model systems which complement discrete state-variables with continuous ones for tracking time or sensor values. This is of immense practical interest and has been the topic of many papers and books, see e.g. [Alu15, LS17] for books calling these systems cyber-physical and [Ras05] as well as [CHVB18, Chapter 29 and 30] for books calling them hybrid. As mentioned above, problems in this setting quickly become undecidable [HR00]. If the whole evolution of the system follows a differential equation, this special case is called differential game [Fri94].

Number of players Recall that stochastic games contain two players, i.e. two kinds of nondeterminism, as well as stochasticity. Thus, they are also called $2\frac{1}{2}$-player games, where the half player corresponds to the stochasticity. Naturally, they generalize games with fewer players, namely Markov chains with $\frac{1}{2}$, Markov decision processes with $1\frac{1}{2}$, and deterministic games with 1 or 2 players. For games with more than $2\frac{1}{2}$ players, we distinguish two cases: if the objective is zero-sum, i.e. one set of players winning corresponds to the other set of players losing, then we can equivalently consider a game with $2\frac{1}{2}$ players [CFK⁺13a]. Otherwise, we need solution concepts like *equilibria* that capture the rational interaction of players. We refer to the next Section 1.5.3 for more information on this.

Concurrent games In this thesis, we investigate games that are *turn-based*, i.e. at every state exactly one player decides the next action. In concurrent games, e.g. [dAHK07], at every state both players choose an action and the system evolves according to both their choices. We refer to [San20] for a PhD-thesis summarizing recent progress on concurrent games. Note that Matrix-games are equivalent to concurrent games with one state [FV97, Appendix G]. A variant of concurrent games are *bidding games*, see e.g. [AH20], where the next move is determined based on an "auction" between the players that is held at every state.

1.5.3 Specification

On an intuitive level, we can view single-objective zero-sum specifications as follows: firstly, we need a way to assign a number to every possible play of the game, i.e. every sequence of states and actions that can occur. This can be a binary classification, saying that some plays are winning and others losing, or one with a larger domain, for example assigning a cumulative reward or cost to every play. Secondly, we need a way to aggregate the numbers of multiple plays, most commonly by using expectation. Thirdly, we have to specify how we resolve the nondeterminism in the game, i.e. how we quantify the strategies of the players. Finally, we have to pose the question we are interested in, which can be computing optimal strategies and values or verifying that the value is larger than a certain threshold. We now go through these steps and recall some ways in which they can be modified; afterwards, we comment on multi-objective and non-zero-sum specifications. For an overview with a different structure, we refer to [BHK19, Chapter 4].

Objectives In our intuitive description, an objective is the way to assign a number to a play. The distinction between classifying with binary or larger domain is usually denominated as *qualitative* or *quantitative* objectives, and we refer to [CHVB18, Section 27.2.2] for an overview of several well-known objectives belonging to both classes. We highlight that there are several logics for encoding properties [CHVB18, Chapters 2 and 28] and that the automata-based approach for representing objectives, see also [CHVB18, Chapter 4], was a historical breakthrough. Note that objectives can also be classified according to whether their value depends only on a finite number of steps (finite-horizon) or whether potentially the whole play is relevant (infinite-horizon). A recently introduced special case are window-objectives, e.g. [CDRR15], where the payoff is considered over a local finite window sliding along the play.

We comment on several historical names for special classes of games, namely Shapley, Gilette and Everette games. We base our classification on the description in [HKL$^+$11]. All these games are concurrent stochastic games, but they differ in their objective. Shapley games have a discounted reward objective, Gilette games a mean-payoff objective and Everette games (also called recursive games, but not to be confused with recursive games as in [EY15]) a terminal reward objective.

We emphasize again that the objective of reachability which is the focus of this thesis is very fundamental. This is not only because many other objectives can be polynomially reduced to reachability [AM09, CF11], but also because solution algorithms for many other objectives depend on solving reachability. For example, solving ω-regular objectives [CHVB18, Section 28.6] or mean-payoff objectives [ACD$^+$17] intuitively amounts to analysing the cycles of the system and then performing reachability analysis on these cycles.

Expectation or risk-aware measures Expectation is the most common choice for aggregating information about many possible plays in a probabilistic system. Intuitively, we multiply the value of every play with the probability of the play and take the average. However, expectation might in fact not be the most natural or useful choice, since it is not risk-aware. Instead, we might want to consider measures such as the conditional value-at-risk [KM18] or the ideas of cumulative prospect theory [LM13].

Quantification of strategies Asking for the existence of a strategy makes sense if we are controlling the agent in question and want to find a good way to behave; asking that something holds for all strategies gives information about the worst case, defending against whatever an opponent can do. In games, the order of quantification can be important, i.e. who fixes their strategy first. As an intuitive example, assume we are playing rock-paper-scissors and you have to tell your opponent what you will play before they decide; this order puts you at a distinct disadvantage. Still, in many cases, it actually is irrelevant who fixes the strategy first. In such cases, we say that game is *determined.* Determinacy has been proven abstractly for large classes of stochastic games, see e.g. [Mar90, MS98].

Optimum, qualitative decision or quantitative decision We can be interested in computing the optimal value or optimal strategies. If our problem is to decide whether the optimum has a certain property, we can differentiate between qualitative and quantitative questions.[1] Qualitative analysis distinguishes only whether the specification is satisfied with a probability of 0, some positive probability or with a probability of 1; for the latter

[1]Note that these words are overloaded to denote both the domain of the objective as well as the overall question.

case, we also differentiate sure, almost-sure and limit-sure winning [dAHK07]. In contrast, in quantitative analysis, we want to compute the exact probability of winning. Note that recently there have been works focussing not on computing the optimum, but rather "satisficing", which means ensuring that the probability of winning is higher than a threshold [BCV21]. A related concept is good-enough synthesis [AK20], where the specification only has to be satisfied if such satisfaction is possible at all.

Multiple objectives The topic of Chapter 5 are specifications that consist of multiple objectives. We discuss in detail different ways of combining such objectives in Section 5.3.

Non-zero-sum games In non-zero-sum games, the players are not antagonistic. Instead, both of them have their own objective, and their objectives are only partially (or even not at all) conflicting. In this case, the solution concept is not a single optimal value, but we look for *equilibria*. These are combinations of strategies where no player has an incentive to deviate. There exist many different concepts of equilibria, depending on the number of players as well as the assumptions about their rationality and motivation.

We present a selection of relevant works. The concept of equilibria was introduced by Nash [NJ50], but many extensions have been proposed, e.g. relevant equilibria [BBGT21] or Stackelberg games [BRT21]. The concepts of correlated equilibria [KNPS22] and core in [GKW19] assume that the players can and want to cooperate. Synthesis in the presence of other rational agents with their own objectives is called rational synthesis. It was introduced in [FKL10] and strengthened in [KPV16]. Small modifications of the objectives of the other players in order to achieve stability or improve social welfare were studied in [AAK15]. Finally, we refer to [BCH$^+$16] for a survey of solution concepts in the context of reactive synthesis.

1.5.4 Output

Model checking or synthesis Originally, model checking answers the decision problem of: does the system satisfy the specification? Thus, the output is a yes/no-answer, accompanied by either a proof of correctness or a counter-example witnessing a violation of the specification [CHVB18, Section 1.1]. We highlight that counter-examples can be used for explaining bugs of the system or refining an abstraction. For more information, see e.g. [AL09, DJW$^+$14, ÁBD$^+$14, BCC$^+$15].

If the models contain nondeterminism, i.e. an agent with choices, we have to specify whether this agent is beneficial or adversarial. In the former case, we check whether there exists a behaviour of this agent, a *strategy*, such that the specification is satisfied; in the latter case, we check that it is satisfied for all behaviours. The witness for such a question commonly is a strategy that ensures/violates satisfaction of the specification. Thus, the question of *synthesis* of such strategies is closely related to model checking. For more information on this relation, see [CHVB18, Chapter 27], and for a history of the synthesis problem, see [Cha07b, Chapter 1].

For games, there are the notions of quantitatively or strategically solving them, which means computing the optimal values of the considered specification or the optimal strategies. In the cases we study in this thesis, it possible to obtain one solution from the other, see also [AM09].

Explainability Given the synthesised strategy which should ensure that the system satisfies the specification, a human engineer can validate this strategy. This allows uncovering mistakes made in the modelling process and thereby increase the reliability of the result. The main goal of Chapter 6 is to post-process strategies to make them explainable and we report some success stories where we detected bugs and assisted the modelling process.

There is a recent trend to explain the results of algorithms that do not give guarantees. We provide an overview in Section 6.3. In this case, the explanation does not serve as validation of an already quite reliable result, but rather as a way to get at least some degree of reliability.

1.5.5 Input

It is very important to find a fitting formal model for the given complex real-world problem. This is addressed in [CHVB18, Chapter 3 and Section 28.7.3] as well as [BK08, Section 1.2.1] and [ABD+15]. These approaches require human input, which is laborious and potentially error prone. Instead, we can also learn the model from simulations, obtaining a probabilistic guarantee of its correctness, see e.g. Section 4.1 or [MCJ+12, HdHA15, WSYP18, WSQJ21].

When modelling, the choice of formalisms immensely affects the outcome of the verification, as we show using an application from IT security in Chapter 7. On the one hand, if the model is not expressive enough, the result is meaningless; on the other hand, if the model is too expressive, the necessary algorithms or solution concepts might be unnecessarily complex. Moreover, the result might be more understandable and explainable given the right modelling of the specification. In particular, encoding multiple objectives into a single utility function (as is necessary for e.g. reinforcement learning [SB18, Section 17.4]) obscures the result. This yields statements like “the combined utility is below 147.3”, whereas encoding the objectives separately allows for explaining “At a cost of 83€, our chances of catching the hacker are 40%”.

Remark 1. *At the end of this high-level overview, I want to point out that the related work sections in this thesis benefited greatly from the following websites.*

- *The dblp computer science bibliography [Ley02], available at* `https://dblp.org/`*, especially for the functionality of giving good bibtex-information for all papers in their database, but also for getting an overview of the work of certain (teams of) authors.*
- *Google Scholar, available at* `https://scholar.google.com/`*, for getting bibtex-information on the few papers that dblp didn't have, and for stumbling upon some highly cited papers that I would never have come across without their heuristics.*
- *Connected papers, available at* `https://www.connectedpapers.com/` *for finding relations, streams of research and new papers. Also, it is a beautiful feeling to know all papers that (according to their heuristics) are most connected to an important citation.*

1.6 Publication summary

This section provides a list of all papers I co-authored. It first enumerates all papers included in this thesis, secondly all other co-authored papers and lastly concludes with a summary.

Papers included in the thesis

This list groups papers by the chapter which discusses them and shows in which appendix the full papers are reprinted.

Chapter 3 - Guaranteed algorithms for solving simple stochastic games

A Value Iteration for Simple Stochastic Games: Stopping Criterion and Learning Algorithm (Information and Computation 2022, [EKKW22])

B Comparison of Algorithms for Simple Stochastic Games (Information and Computation 2022, [KRSW22])

Chapter 4 - Expressive system formalisms: Limited information

C PAC Statistical Model Checking for Markov Decision Processes and Stochastic Games (CAV 2019 [AKW19a])

D Statistical Model Checking: Black or White? (ISoLA 2020, [ADKW20])

E Satisfiability Bounds for ω-Regular Properties in Bounded-Parameter Markov Decision Processes (CDC 2019, [WMK19])

Chapter 5 - Expressive specification formalisms: Multiple objectives

F Approximating Values of Generalized-Reachability Stochastic Games (LICS 2020, [ACK+20])

Chapter 6 - Explainable output

G dtControl 2.0: Explainable Strategy Representation via Decision Tree Learning Steered by Experts (TACAS 2021 [AJK+21a])

Chapter 7 - Application: Defending against advanced persistent threats

H Guaranteed Trade-Offs in Dynamic Information Flow Tracking Games (CDC 2021 [WGMK21])

Other co-authored papers

This list of further papers that were published during my PhD-time again is grouped according to their relation to the chapters of the thesis.

Chapter 3 - Guaranteed algorithms for solving simple stochastic games

- Conference version of Paper A, using the same title (CAV18, [KKKW18])
- Conference version of Paper B, using the same title (GandALF20, [KRSW20])

Chapter 4 - Expressive system formalisms: Limited information

- Enforcing omega-Regular Properties in Markov Chains by Restarting (CONCUR 2021, [EKKW21])

Chapter 5 - Expressive specification formalisms: Multiple objectives

- Stochastic Games with Lexicographic Reachability-Safety Objectives (CAV 2020 [CKWW20])
- Stochastic Games with Disjunctions of Multiple Objectives (GandALF 2021, [WW21])

Chapter 6 - Explainable output

- SOS: Safe, Optimal and Small Strategies for Hybrid Markov Decision Processes (QEST 2019, [AKL$^+$19])
- dtControl: decision tree learning algorithms for controller representation (HSCC 2020, [AJJ$^+$20a])
- dtControl: decision tree learning algorithms for controller representation (HSCC 2020, [AJJ$^+$20b], tool demonstration)

Other

- Automata Tutor v3 (CAV 2020, [DHK$^+$20])
- Index appearance record with preorders (Acta Informatica 2021 [KMWW21])

Remark 2. *We make a short excursion to highlight that there is a recent development to improve the teaching of formal methods using online tools. Automata Tutor [DHK$^+$20] is a part of this development. Further work in that direction was presented at the workshop on formal methods education online (FOMEO), which I co-organized.*

Its website[2] *provides links to and descriptions of many useful tools, including DiMo [HLS21], Hintikka's World [Sch18, CGNS19], Iltis [GLP$^+$18, GLH$^+$19], Larch*[3]*, LogAx [LHJN21], LogEx [LHJ19], NaDeA [VFS18], pseuCo [FH19], Sequent Calculus Trainer [EHL17] and The Incredible Proof Machine [Bre16].*

Summary

In total, I have published 18 papers during my PhD-studies, namely 11 regular conference papers (3×CAV, 2×CDC, CONCUR, 2×GandALF, Isola, LICS and QEST), 3 conference tool papers (CAV, HSCC and TACAS), one conference demo paper (HSCC) and 3 journal papers (2×Information and Computation, Acta Informatica). Before my PhD-studies, I co-authored one conference paper (TACAS) and one journal paper (Nature methods).

As of June 18th 2022, the number of citations for the papers published during my PhD-studies is 126 according to Google Scholar;[4] including the papers published before the start of my PhD-studies, it is 296.

[2]https://www7.in.tum.de/~kretinsk/fomeo.html

[3]https://thelarch.notion.site/thelarch/Larch-73c9d52748e64d4b83e2812e458b7c44

[4]https://scholar.google.com/citations?user=88fkEooAAAAJ&hl=de&oi=ao

2 Preliminaries

In this chapter, we give the preliminary definitions that are relevant for all the included works.

Intuition of the important concepts. Before the formal definitions, we give an intuitive summary of the concepts to make the preliminaries more readable. *Stochastic games* are games played on a graph. Every node of the graph (called *state*) belongs to one of two players, which we call Maximizer and Minimizer. These names are chosen because in all cases we consider, one player is trying to maximize an objective and the other tries to minimize it.

Playing the game starts in a given initial state. Then repeatedly the player to whom the state belongs chooses an edge (called *action*), and the play moves along that edge to the next state. However, as the game is *stochastic*, an action can have multiple successor states, and the successor is then sampled according to a probability distribution associated with the action. In this way, a play moves through the game graph, depending on the *strategies* of the players, i.e. on their choices of which action to pick. Plays are infinitely long, so eventually, the play ends up in some cycle of the graph.

Additionally to the system of the stochastic game, we consider an objective, which encodes the specification that we want to check. For example, in the *reachability* objective player Maximizer wants to maximize the chances of reaching a given set of goal sets, while player Minimizer has the opposite objective of avoiding the goal set. We are always interested in two things: (i) the optimal strategies for both players as well as (ii) the resulting probability to satisfy the objective under these optimal strategies.

2.1 Stochastic games

A probability distribution on a finite set X is a mapping $\delta : X \to [0,1]$ such that $\sum_{x \in X} \delta(x) = 1$, i.e. it assigns a probability between 0 and 1 to every element of X such that the overall sum of probabilities is equal to 1. The set of all probability distribution on X is denoted by $\mathcal{D}(X)$.

We now define discrete-time turn-based stochastic games as in e.g. [Con92].

Definition 1 (Stochastic Game (SG)). *A (turn-based)* stochastic game *(SG) is a tuple* $G := (\mathsf{S}, \mathsf{S}_\square, \mathsf{S}_\bigcirc, \mathsf{A}, \mathsf{Av}, \delta)$*, where*

- S *is a finite, non-empty set of states, partitioned into the the states of Maximizer* $\mathsf{S}_\square$ *and Minimizer* $\mathsf{S}_\bigcirc$*. Partitioned means that* $\mathsf{S}_\square \subseteq \mathsf{S}$, $\mathsf{S}_\bigcirc \subseteq \mathsf{S}$, $\mathsf{S}_\square \cup \mathsf{S}_\bigcirc = \mathsf{S}$ *and* $\mathsf{S}_\square \cap \mathsf{S}_\bigcirc = \emptyset$.

- A *is a finite, non-empty set of actions.*

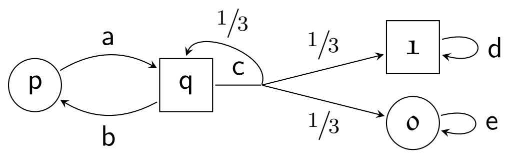


Figure 2.1: An example of an SG with $\mathsf{S} = \{\mathsf{p}, \mathsf{q}, \mathsf{1}, \mathsf{0}\}$, $\mathsf{S}_\square = \{\mathsf{q}, \mathsf{1}\}$, $\mathsf{S}_\bigcirc = \{\mathsf{p}, \mathsf{0}\}$ and the set of actions $\mathsf{A} = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}\}$; $\mathsf{Av}(\mathsf{p}) = \{\mathsf{a}\}$ with $\delta(\mathsf{p}, \mathsf{a})(\mathsf{q}) = 1$; $\mathsf{Av}(\mathsf{q}) = \{\mathsf{b}, \mathsf{c}\}$ with $\delta(\mathsf{q}, \mathsf{b})(\mathsf{p}) = 1$ and $\delta(\mathsf{q}, \mathsf{c})(\mathsf{q}) = \delta(\mathsf{q}, \mathsf{c})(\mathsf{1}) = \delta(\mathsf{q}, \mathsf{c})(\mathsf{0}) = \frac{1}{3}$. For actions with only one successor, we do not depict the transition probability 1. Taken from [EKKW22, Figure 1].

- $\mathsf{Av} : \mathsf{S} \to 2^{\mathsf{A}}$ *is a total function assigning a non-empty set of available actions to every state.*
- $\delta : \mathsf{S} \times \mathsf{A} \to \mathcal{D}(\mathsf{S})$ *is the transition function. Given a state and an available action, it returns a probability distribution over successor states.*

In Figure 2.1, we give an example of a stochastic game from [EKKW22], which we use as running example for several chapters of the thesis. To complement our definition, we mention useful **standard restrictions and notations**.

- In contrast to the notation in e.g. [Con92], we do not consider stochastic states, but a stochastic transition function as in e.g. [EKKW22]. Both formalisations are equivalent. We prefer the stochastic transition function because it is closer to the standard notation for Markov decision processes, see e.g. [Put94].
- Given a state set $T \subseteq \mathsf{S}$, we write $T_\square := T \cap \mathsf{S}_\square$ and $T_\bigcirc := T \cap \mathsf{S}_\bigcirc$ to denote all Maximizer respectively Minimizer states in T.
- We assume that SGs are *non-blocking*, i.e. for all $s \in \mathsf{S}$ it holds that $\mathsf{Av}(s) \neq \emptyset$. This is reasonable, since we study infinite-horizon behaviour, and if there was no available action in a state, the play could not move on. It is not restrictive because if we want to have a state s where the play is stuck, we can model this behaviour by having a single available action that surely loops back into s.
- The transition function is a partial function that is only defined for tuples (s, a) of a state $s \in \mathsf{S}$ and an available action $a \in \mathsf{Av}(s)$.
- Instead of $\delta(s, a)(s')$ we write $\delta(s, a, s')$, slightly abusing notation when accessing a probability distribution.
- We write $\mathsf{Post}(s, a) := \{s' \mid \delta(s, a, s') > 0\}$ to denote all successors of a state-action pair.

An SG contains two players, i.e. two kinds of nondeterminism, as well as stochasticity. Thus, SGs are also called $2\frac{1}{2}$-player games, where the half player corresponds to the stochasticity. Consequently, **SGs generalize games** with $\frac{1}{2}$, 1, $1\frac{1}{2}$ and 2 players. For deterministic games with 1 or 2 players [CHVB18, Chapter 27], we have that for all $s, s' \in \mathsf{S}, a \in \mathsf{Av}(s) : \delta(s, a, s') \in \{0, 1\}$. In Markov chains ($\frac{1}{2}$-player games, see e.g. [Nor98]) we have that $|\mathsf{Av}(s)| = 1$ for all states s of both players,[1] and in Markov decision processes (MDP, 1½-player games, see e.g.[How60, Put94]) $|\mathsf{Av}(s)| = 1$ holds for all states of at least one player[2]. For an excellent overview of these models and how to solve them when considering different objectives, we refer the interested reader to [CH08]. In this thesis, we often talk about Markov chains or Markov decision processes, as these are well-researched models and it is useful to compare to them and transfer ideas.

2.2 Semantics: Paths and strategies

The semantics of an SG is given in the standard way by means of paths and strategies. In the intuitive description, we said that a play moves through the graph depending on the choices of the players. Such a play is called *path* and the choices of the players are defined by their *strategies*. Given strategies of both players, we can compute the probability of a path and thus argue about the events we are interested in. For example, the probability of reaching a goal state t is given by the cumulative probability of all paths that contain the state t.

Towards a more **formal definition**, we first define the following notation: for a set X we use X^* and X^ω to denote all finite respectively infinite sequences consisting of elements of X. Then, an *infinite path* ρ is an infinite sequence $\rho = s_0 a_0 s_1 a_1 \dots \in (\mathsf{S} \times \mathsf{A})^\omega$, such that for every $i \in \mathbb{N}$, $a_i \in \mathsf{Av}(s_i)$ and $s_{i+1} \in \mathsf{Post}(s_i, a_i)$. We denote the set of all infinite paths by Paths. *Finite paths* are defined analogously as elements of $(\mathsf{S} \times \mathsf{A})^* \times \mathsf{S}$.

A *memoryless pure strategy* is a function $\sigma : \mathsf{S} \to \mathsf{A}$ mapping every state to an available action.[3] These kinds of strategies are sufficient for the reachability objective [Con92] and thus of most importance to this work. A *memoryless randomized strategy* is a function $\sigma : \mathsf{S} \to \mathcal{D}(\mathsf{A})$, which can randomize over multiple actions instead of mapping every state to a single one. For completeness, we also mention *history-dependent strategies*, also called strategies with memory. They not only depend on the current state but can access the whole history of the finite path that was played so far. They are relevant in Section 4.2. However, we can omit the formal definition because we avoid a full technical treatment in the thesis in order to focus on the general implications of our results. Finally, we mention that strategies can also be called *controller*, *policy* or *scheduler*.

Throughout the thesis, we always use $\sigma : \mathsf{S}_\square \to \mathsf{A}$ to denote Maximizer's and $\tau : \mathsf{S}_\bigcirc \to \mathsf{A}$ to denote Minimizer's strategies. A pair of memoryless strategies (σ, τ) together with an SG G induces a Markov chain $G^{\sigma,\tau}$; and on every Markov chain, there exists a unique probability measure $\mathbb{P}$ on measurable sets of paths [BK08, Chapter 10].

We now explain this standard argument in a bit more detail. Intuitively, given an SG and a pair of strategies, every state always plays the single action that is recommended by the strategy. In the case of randomized strategies, it plays a fixed distribution over the actions. More formally, we now show how to construct a Markov chain as in [BK08, Definition 10.1] from an SG G and a pair of memoryless strategies (σ, τ). The construction

[1] When a state has exactly one action, it is irrelevant which player owns the state and we can omit the information about the partitioning of the state-space.

[2] Equivalently, we can require that all states belong to the same player, i.e. either $\mathsf{S}_\square = \emptyset$ or $\mathsf{S}_\bigcirc = \emptyset$.

[3] This means we require that $\sigma(s) \in \mathsf{Av}(s)$ for all $s \in \mathsf{S}$

works on randomized strategies and hence also applies to the special case of pure ones. The Markov chain has the same state-space as G. Note that [BK08, Definition 10.1] even allows a countably infinite set, but we only need a finite set. The new transition function $\delta' : \mathsf{S} \to \mathcal{D}(\mathsf{S})$ does not depend on actions any more. For a Maximizer state $s \in \mathsf{S}_{\Box}$, it is given as $\delta'(s, s') = \sum_{a \in \mathsf{Av}(s)} \delta(s, \sigma(a), s')$, and dually with σ replaced by τ for states of Minimizer. The other components of [BK08, Definition 10.1], the initial distribution and the labelling function, are related to objectives. The labelling function in fact is not relevant for specifying the probability measure. The initial distribution defines where the considered paths start. We use an additional parameter $s \in \mathsf{S}$, i.e. we start in the given s with a probability of 1. In the end, we get $\mathbb{P}^{\sigma,\tau}_{s,G}$, the probability measure for the SG G on infinite paths starting in s under the strategies (σ, τ). For a description of how to obtain a probability measure on this Markov chain, we refer to the extensive description in [BK08, Section 10.1], in particular the Excursus on Probability Spaces and the Definitions 10.9 and 10.10.

When considering history-dependent strategies, the construction differs. The state-space of the Markov chain is not S anymore, but instead the set of all infinite paths Paths. Note that then the Markov chain is countably infinite. As these strategies are not relevant for the content of this thesis, we refrain from giving the details and refer the interested reader to [BK08, Section 10.6], in particular Definitions 10.92 and the following descriptions.

2.3 Objectives

Now that we have defined the model of stochastic game and the probability space over the paths occurring under strategies of both players, we can talk about the objectives of the players. Objectives formalise the goals of both players by assigning a value to each path.

We are interested both in *quantitatively* and *strategically* solving SGs, as differentiated in [AM09]. In other words, we want to compute the *value of the game*, which is the optimal expected value that the players can ensure; as well as the *optimal strategies* that the players use to obtain this value. Sometimes, we allow for small deviations of ε, i.e. we do require the optimal value, but it suffices to have something that differs by at most ε. In that case, we speak of ε-approximation of the value and ε-optimal strategies.

Note that we can derive the value from optimal strategies and vice versa: when we have (ε-)optimal strategies, we can obtain the value by applying the strategies and solving the induced Markov chain — which boils down to simple equation solving. Given the values of a reachability objective for all states, we can compute optimal strategies by picking locally optimal actions and some additional graph analysis, see [AM09, Theorem 2]. Also, see our work [EKKW22, Appendix A] for an extensive description and proof in the case of obtaining ε-optimal strategies from an ε-approximation of the value.

2.3.1 Reachability

Reachability is the most important objective for this thesis and the only one we formally define. As mentioned before, it is very fundamental because many other objectives depend on reachability or can be reduced to reachability, see Section 1.3.

A reachability objective for an SG G is specified by an initial state $s_0 \in \mathsf{S}$ and a set of goal states $\mathsf{F} \subseteq \mathsf{S}$ (also known as targets or **f**inal states). Maximizer tries to maximize the probability to reach a goal state from the initial state, while Minimizer has the opposite objective. Formally, we denote the (measurable) set of all infinite paths that eventually reach a goal state as $\Diamond\mathsf{F} := \{\rho \in \mathsf{Paths} \mid \rho = s_0 a_0 s_1 a_1 \dots \in (\mathsf{S} \times \mathsf{A})^\omega \wedge \exists i \in \mathbb{N}.s_i \in \mathsf{F}\}$.

Then, for every state $s \in \mathsf{S}$, we define the *value in s* as

$$\mathsf{V}_G(s) := \sup_\sigma \inf_\tau \mathbb{P}^{\sigma,\tau}_{s,G}(\Diamond\mathsf{F}) = \inf_\tau \sup_\sigma \mathbb{P}^{\sigma,\tau}_{s,G}(\Diamond\mathsf{F}).$$

The value of a state s is the accumulated probability of all paths starting in s that reach a goal state under the optimal strategies of both players. Another way of looking at it is that the value of a path is 1 if the path contains a goal state and 0 otherwise. Then the value of a state s is the expectation of a random variable over paths, where this random variable is distributed according to the probability distribution $\mathbb{P}^{\sigma,\tau}_{s,G}$. The *value of the game* is $\mathsf{V}_G(s_0)$, where s_0 is the given initial state. We often omit the G in the subscript when it is clear from the context.

Strategies are called *optimal* if they are in the supremum or infimum; equivalently, they obtain at least the value against all strategies of the opponent. We know that there always exist optimal memoryless pure strategies for both players in SGs with a reachability objective [Con92, Lemma 4 and 5]. For $0 \leq \varepsilon \leq 1$, we call a strategy σ of Maximizer ε-*optimal* if for all states $s \in \mathsf{S}$ we have $\inf_\tau \mathbb{P}^{\sigma,\tau}_{s,G}(\Diamond\mathsf{F}) \in [\mathsf{V}(s) \pm \varepsilon]$, and dually for Minimizer with $\inf_\tau$ replaced by $\sup_\sigma$.

We mention several **interesting properties** of SGs with a reachability objective.

- They are *determined*, i.e. it does not matter which player fixes a strategy first [Con92, Lemma 6]. This is why we can switch the order of supremum and infimum in the definition of the value in a state.

- They are *zero-sum*, i.e. one player cannot improve their outcome without worsening the outcome for the other player.

- They are in UP $\cap$ co-UP [CF11, Theorem 4], see also the discussion in Section 1.3.

- Instead of considering a single initial state, we can use initial distributions. This is equivalent to adding a new initial state that transitions according to the initial distribution. Dually, one can consider a single goal state instead of a set, by adding a sure transition from every state in F to the new goal state. We choose a single initial, but multiple target states, as we deem it most natural when modelling a real problem: we know where we start but have multiple possible goal states.

- We can assume that all goal states are absorbing, i.e. they only have one action that surely self-loops. This is because the value of a path only depends on whether the state was reached once and it is irrelevant how the play continues afterwards.

2.3.2 Other objectives

We now mention several other objectives which are relevant in certain parts of the thesis. Since the discussion of the results will be at a rather high level, we do not require a formal definition.

- *Safety* is the dual of reachability. The goal of Maximizer is to maximize the chance of remaining in a safe set of states. This is equivalent to minimizing the chance of reaching the complement of the safe states, the states to be avoided. Hence maximizing safety is the same as minimizing reachability. Chapter 6 deals (among other things) with strategies arising from safety objectives.

- *Expected total reward*, cf. [CFK$^+$13a, Section 3.2 and 4.3], uses a reward structure to assign a rational number to every state. Then the objective of Maximizer is to maximize the expected sum of all rewards. There are several variants of this objective, e.g. indicating how to deal with cycles that lead to infinite rewards. We use total reward objectives in Chapter 7.

- *ω-regular objectives*, cf. [CH12], assign labels from a finite set of atomic propositions to every state. Thus, every path corresponds to an infinite word over the set of atomic propositions. An ω-regular objective then specifies an ω-regular language L — usually in the form of an automaton or a formula in linear temporal logic (LTL). The value of a path is 1 if the associated infinite word is in L and 0 otherwise. This very rich class of objectives contains objectives classes such as reachability, safety, and liveness, as well as objectives given by ω-regular automata, e.g. Büchi, Rabin, Streett, Müller, parity. The contribution in Section 4.2 applies to ω-regular objectives.

2.4 Strongly connected components and end components

We define two graph theoretic concepts that are relevant when analysing and solving SGs: strongly connected components and end components. Intuitively, they capture the notion of state sets in which a play can cycle. In strongly connected components, every state can reach every other state with positive probability; in end components, states can reach each other almost-surely, i.e. with a probability of 1.

Definition 2 (Strongly connected component (SCC))**.** *A non-empty set $T \subseteq \mathsf{S}$ of states is* strongly connected *if for every pair $s, s' \in \mathsf{S}$ there is a path (of non-zero length) from s to s'. Such a set T is a* strongly connected component (SCC) *if it is maximal w.r.t. set inclusion, i.e. there exists no strongly connected T' with $T \subsetneq T'$.*

SCCs of an SG can be computed in linear time using Tarjan's algorithm [Tar72]. Moreover, the SCCs form a directed acyclic graph, so we have a topological ordering of these components. This is useful because then we know that no SCC can reach any state in an SCC that comes before it in the topological ordering. This insight is useful for several algorithm, see [DMWG11, KM17] and Section 3.4.2.

However, SCCs only consider the underlying graph, since they depend on the existence of some path. Hence, they are agnostic of probabilities and the actions of the players. The concept of end component (EC, [DA98]) takes these into account. Intuitively, an SCC is an EC if, under some pair of strategies, a play can *end* there, i.e. surely remain in the EC infinitely long.

Definition 3 (End component (EC))**.** *A non-empty set $T \subseteq \mathsf{S}$ of states is an* end component (EC) *if there is a non-empty set $B \subseteq \bigcup_{s \in T} \mathsf{Av}(s)$ of actions such that (i) for each $s \in T, a \in B \cap \mathsf{Av}(s)$ we have $\mathsf{Post}(s,a) \subseteq T$ and (ii) for each $s, s' \in T$ there is a finite path $\rho = s a_0 s_1 a_1 \dots a_n s' \in (T \times B)^* \times T$, i.e. the path goes from s to s' while staying inside T and only using actions in B.*

A *maximal end component (MEC)* is an EC T that is inclusion maximal, i.e. there is no EC T' such that $T \subsetneq T'$. A MEC decomposition of the SG, i.e. a directed acyclic graph of MECs, can be computed in polynomial time, see [CY95] for basic or [CH14] for improved algorithms. Note that these works refer to MDPs. However, MEC computation for SGs assumes that both players cooperate and thus amounts to computing MECs in the MDP where all states belong to the same player. This already hints at a problem of MECs in SGs that we address in Chapter 3: the concept does not take into account that the players are adversarial and one player might actually rather leave an EC, preventing the path from "ending" there.

An SCC or EC T is called *bottom*, if for all states $s \in T$, $a \in \mathsf{Av}(s)$ we have $\mathsf{Post}(s,a) \subseteq T$, i.e. no transition leaves the SCC/EC. Bottom SCCs (BSCCs) are relevant in Section 4.1. Under any pair of strategies (σ, τ), the path in an SG G will end up in a MEC [DA98, Theorem 3.2] and hence in a BSCC of the induced Markov chain $G^{\sigma,\tau}$.

Example 1 (ECs and SCCs)**.** *Consider the SG in Figure 2.1. There are three ECs: $\{\mathsf{p},\mathsf{q}\}$, $\{\mathtt{1}\}$ and $\{\mathtt{0}\}$. All of these are also MECs and SCCs and the latter two are bottom. Note that $\{\mathsf{p},\mathsf{q}\}$ is an EC because there exists a strategy staying in it, namely picking b in q. The fact that there are strategies exiting the state set only makes this EC non-bottom.*

If there was some probability to go from $\mathtt{0}$ back to p, then $\{\mathsf{p},\mathsf{q},\mathtt{0}\}$ would be an SCC, since every state can reach every other state. It would not be an EC because there is a chance to go to $\mathtt{1}$. Hence, no pair of strategies remains inside the state set surely and allows to reach every state in it. △

If an SG contains no ECs except for designated absorbing states (e.g. a goal and a sink state that both self-loop with a probability of 1), it is called *stopping.* The intuition for this term is that under all strategies, the absorbing states are reached with a probability of 1 and the game "stops" (i.e. for infinitely many steps remains in the same designated goal or sink state). This restricted class of SGs has received quite some attention for two reasons: firstly, the absence of ECs makes algorithms and proofs easier. Secondly, there exists a polynomial transformation from an arbitrary SG to a stopping one with almost the same value [Con93]. However, this transformation is impractical, as we detail in Section 3.3.2.

3 Guaranteed algorithms for solving simple stochastic games

Motivation In this chapter, we deal with the problem of simple stochastic games, i.e. turn-based stochastic games (SGs) with a reachability objective. This serves as the basis of this thesis, as the other chapters build on the algorithms described in this chapter. Moreover, as discussed in Section 1.3, simple SGs are fundamental and practically useful. We address the challenges described in Section 1.2 as follows: all algorithms described in this chapter are *reliable* in the sense that they give guarantees on their result and we investigate and improve their *scalability*.

Problem statement Given an SG G, an initial state s_0 and a set of goal states F we want to compute (an ε-approximation of) the value of the game $\mathsf{V}_G(s_0)$. Most of our algorithms compute the value function V_G for all states, not just the initial state. Moreover, we are interested in (ε-)optimal strategies achieving these values. Recall that, given (ε-)optimal strategies we can infer (an ε-approximation of) the value function and vice versa, see Section 2.3.

Note that it is possible by simple graph algorithms (e.g. a backwards search from the goal states) to find the states with no possible path to the target. We define the set of sink states, i.e. of all states that cannot reach any state in F, as $\mathsf{Z} := \{z \in \mathsf{S} \mid \neg\exists \rho \in \Diamond\mathsf{F}. \rho_0 = z\}$, where ρ_0 denotes the starting state of a path. All states in Z trivially have value **z**ero, and hence every strategy is optimal. Similarly, all states in F have value one. Hence, we only need to compute the values and strategies for states in the set of "unknown" states $\mathsf{S}_? := \mathsf{S} \setminus (\mathsf{F} \cup \mathsf{Z})$.

Chapter outline We describe the three known classes of algorithms that are used to solve simple SGs, namely value iteration, strategy iteration and quadratic programming. For every class, we have a section that first describes the basic form, then our improvements and lastly summarizes related developments. The section on value iteration is based on [EKKW22] (Appendix A) and the sections on strategy iteration and quadratic programming on [KRSW22] (Appendix B). Afterwards, we outline the experimental results of both papers, i.e. we practically compare the algorithms. Finally, we summarize the chapter, theoretically compare the algorithms and discuss the implications of our experimental results.

3.1 Value iteration (VI)

3.1.1 State of the art

Value iteration (VI) is a well-known algorithm in many settings, in particular for probabilistic model checking. It leverages the fact that the value V is the *least fixpoint*[1] of the so-called *Bellman equations*, see e.g. [Con93].

$$\mathsf{V}(s) := \begin{cases} 1 & \text{if } s \in \mathsf{F} \\ \max_{a\in\mathsf{Av}(s)} \mathsf{V}(s,a) & \text{if } s \in \mathsf{S}_\Box \\ \min_{a\in\mathsf{Av}(s)} \mathsf{V}(s,a) & \text{if } s \in \mathsf{S}_\bigcirc \end{cases} \tag{3.1}$$

where

$$\mathsf{V}(s,a) := \sum_{s'\in S} \delta(s,a,s') \cdot \mathsf{V}(s') \tag{3.2}$$

VI computes a sequence of under-approximations of the value $\mathsf{L}_i : \mathsf{S} \to [0,1]$. The initial lower bound L_0 is set to be certainly smaller than the value, i.e. $\mathsf{L}_0(s) = 0$ for all $s \in \mathsf{S} \setminus \mathsf{F}$, and only for the goal states $f \in \mathsf{F}$ we initialize $\mathsf{L}_0(f) = 1$. Then we repeatedly apply Bellman updates according to Equations (3.3) and (3.4)

$$\mathsf{L}_n(s) := \begin{cases} \max_{a\in\mathsf{Av}(s)} \mathsf{L}_n(s,a) & \text{if } s \in \mathsf{S}_\Box \\ \min_{a\in\mathsf{Av}(s)} \mathsf{L}_n(s,a) & \text{if } s \in \mathsf{S}_\bigcirc \end{cases} \tag{3.3}$$

$$\mathsf{L}_n(s,a) := \sum_{s'\in\mathsf{S}} \delta(s,a,s') \cdot \mathsf{L}_{n-1}(s') \tag{3.4}$$

Example 2 (Basics of VI)**.** *For the convenience of the reader, we show our running example of an SG from Figure 2.1 again in Figure 3.1a. We illustrate VI from below in the first two columns of the table in Figure 3.1b. The other columns are only relevant for later examples.*

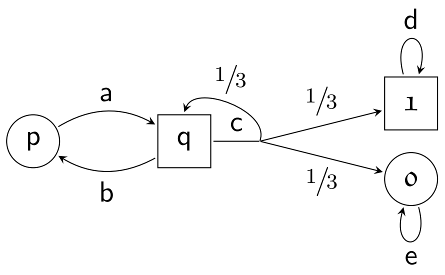
For the states in $\mathsf{F} = \{\mathtt{1}\}$ and $\mathsf{Z} = \{\mathtt{0}\}$, the value trivially is 1 respectively 0, and hence we do not include them in the table. The first row shows the initial value for L_0. State p only has a single action that surely goes to q. Thus, we always have $\mathsf{L}_i(\mathsf{p}) = \mathsf{L}_{i-1}(\mathsf{q})$. In a sense, the information from q is always propagated to p with a delay of one iteration.

We now explain the Bellman update for state q.

$$\mathsf{L}_1(\mathsf{q}) = \max_{a\in\mathsf{Av}(s)} \mathsf{L}_0(s,a) = \sum_{s'\in\mathsf{S}} \delta(\mathsf{q},\mathsf{c},s') \cdot \mathsf{L}_0(s') = 1/3 + 0 + 0 = 1/3$$

Since q is a Maximizer state, it picks the action maximizing the probability to reach the goal; this is action c because using b only promises an estimate of 0 by going to p surely. In contrast, for action c, the resulting value is $\delta(\mathsf{q},\mathsf{c},\mathtt{1}) \cdot \mathsf{L}_0(\mathtt{1}) = 1/3$. Note that the terms for both q and $\mathtt{0}$ do not contribute to the estimate because $\mathsf{L}_0(\mathsf{q}) = \mathsf{L}_0(\mathtt{0}) = 0$.

[1]Note that all sink states in Z have a value of 0. While there are infinitely many fixpoints of the Bellman equations for these states, the least one always assigns 0 to all $z \in \mathsf{Z}$.



(a)

i	$\mathsf{L}_i(\mathsf{p})$	$\mathsf{L}_i(\mathsf{q})$	$\mathsf{U}_i(\mathsf{p})$	$\mathsf{U}_i(\mathsf{q})$	$\mathsf{U}_i^*(\mathsf{q},\mathsf{c})$
0	0	0	1	1	-
1	0	$1/3$	1	1	$2/3$
2	$1/3$	$4/9$	1	1	$5/9$
3	$4/9$	$13/27$	1	1	$14/27$

(b)

Figure 3.1: (a): Our running example of an SG. (b): An illustration of the first few iterations of VI on the running example.

However, in the next iteration, we utilize the knowledge that $\mathsf{L}_1(\mathsf{q}) = 1/3$ *and hence the self loop contributes* $\delta(\mathsf{q},\mathsf{c},\mathsf{q}) \cdot \mathsf{L}_1(\mathsf{q}) = 1/9$*. Thus the resulting estimate is* $\mathsf{L}_2(\mathsf{q}) = 4/9$*. Note that using action* c *still promises a higher value than using action* b*. Repeatedly applying Bellman updates, we will get estimates closer and closer to the correct value of* $\mathsf{V}(\mathsf{p}) = \mathsf{V}(\mathsf{q}) = 1/2$*.*

Note that in the i-th iteration, VI computes the probability to reach the goal in i steps, see e.g. [BK08, Remark 10.104]. But for any finite number of iterations i, there is a path which always uses the self-loop in state q*. This path has positive probability (namely* $(1/3)^{i-1}$*, assuming we start in state* p*). Thus, VI cannot converge to the value in finite time because after i steps there still is a positive chance to remain in* $\{\mathsf{p},\mathsf{q}\}$*. In the limit, the probability of this path that always loops is 0, and thus VI converges.* △

We **formalise the intuition** gained from the example as follows: the sequence of approximations converges to the value in the limit, i.e. $\lim_{i\to\infty} \mathsf{L}_i = \mathsf{V}$ [CH08]. However, convergence may happen *only* in the limit and not in finite time. Example 2 shows this for a very small SG, and the same occurs for Markov decision processes (MDPs, i.e. SGs with only one player) or even Markov chains (where every state only has a single action).[2] The only stopping criterion of VI for SGs known previous to our work was to run VI for a specific number of steps, which is exponential in the size of the SG; afterwards, one could round the computed approximation to the value [CH08]. This approach is impractical because it always needs exponential time. Hence, practically the algorithm was stopped when it was observed to make little progress in an iteration, but this could lead to arbitrarily wrong results [HM18]. This is why we introduce a more practical stopping criterion.

3.1.2 Contribution: Bounded value iteration with a guaranteed stopping criterion

To estimate how close the current under-approximation L_i and the value V are, we introduce an additional sequence of over-approximations $\mathsf{U}_i : \mathsf{S} \to [0,1]$. Then we know that the value of every state s is between $\mathsf{L}_i(s)$ and $\mathsf{U}_i(s)$, and thus we obtain an ε-approximation of $\mathsf{V}(s)$ as soon as $\mathsf{U}_i(s) - \mathsf{L}_i(s) < \varepsilon$.

[2]Note that the SG in Figure 3.1a is equivalent to an MDP where all states belong to Maximizer because all Minimizer states have only a single action. Moreover, removing action b turns the example into a Markov chain where the problem of non-convergence in finite time persists.

The problem of non-convergence from above

Naively, we initialize our over-approximation to $\mathsf{U}_0(s) = 1$ for all states and then apply Bellman updates as for the under-approximation, using Equations (3.3) and (3.4) (replacing L with U). We can also[3] set $\mathsf{U}_0(z) = 0$ for all easily pre-computable sink states $z \in \mathsf{Z}$, dually to setting the lower bound of goal states to 1.

For this algorithm to be sound, we require correctness and convergence. *Correctness* means that $\mathsf{L}_i(s) \leq \mathsf{V}(s) \leq \mathsf{U}_i(s)$ for all iterations i and states s. This is easy to prove, e.g. by an induction or by observing that Bellman updates cannot reduce an over-approximation below below the least fixpoint of the Bellman equations, and that hence $\mathsf{U}_i(s) \geq \mathsf{V}(s)$ for all iterations and states. *Convergence* means that $\lim_{i\to\infty} \mathsf{U}_i = \mathsf{V}$. This is quite difficult to obtain because the VI algorithm can get stuck at greater fixpoints. We now illustrate this with an example.

Example 3 (Non-convergence of VI from above)**.** *Consider again the SG and table in Figure 3.1. The right side of the table shows the behaviour of the over-approximation* U*. Even though we used the smarter initialization and set* $\mathsf{U}_0(\mathsf{o}) = 0$*, the upper bound of* q *and* p *does not decrease, but is stuck at 1. This is because* $\mathsf{U}_0(\mathsf{q}, \mathsf{b}) = 1$ *and* $\mathsf{U}_0(\mathsf{q}, \mathsf{c}) = 2/3$*. The preferred action in state* q *is* b *since it promises a higher value. Hence the over-approximation does not change, i.e.* $\mathsf{U}_1(\mathsf{q}) = \mathsf{U}_0(\mathsf{q})$*. Bellman updates do not progress towards the value (the least fixpoint of the Bellman equations) because* U_0 *is a greater fixpoint of the Bellman equations.*

In other words: looking at the current upper bound, Maximizer is under the impression that going back to p *and staying in the end component (EC)* $\{\mathsf{p}, \mathsf{q}\}$ *yields a higher value. However, cycling in the EC forever by always picking* b *actually reduces the probability to reach the target to 0. So a convergent VI algorithm has to perform an additional step to inform states in an EC that they should not depend on each other, but on an action that* exits *the EC. In the example, we want that* $\mathsf{U}_1(\mathsf{q}) = \mathsf{U}_0(\mathsf{q}, \mathsf{c}) = 2/3$*.*

If we continued to update in this fashion, as exemplified in the column U_i^* *of the table in Figure 3.1b, then we would get a sequence converging to the correct value of* $1/2$*. We also see that we quickly get a precise estimate of the value. With only the lower bound, after three iterations we can only conclude that the value of* q *is greater than* $13/27$*. Using both the lower and upper bound, we know that it is in the interval* $[13/27, 14/27]$*. For* $\varepsilon = 0.1$*, we could terminate the algorithm now, as the difference between lower and upper bound is smaller than* ε*.* △

The **problem of greater fixpoints** hindering the convergence of the over-approximation occurs only in ECs. Moreover, it only happens in certain kinds of ECs. To define these, we introduce the notion of an exiting state-action pair. Let $T \subseteq \mathsf{S}$ be a set of states. Then we say (s, a) exits T (or (s, a) is an exit of T) if $s \in T, a \in \mathsf{Av}(s)$ and $\mathsf{Post}(s, a) \not\subseteq T$. Depending on whether $s \in T_\square$ or $s \in T_\bigcirc$, we call (s, a) a Maximizer- respectively Minimizer-exit. Additionally, we define the best exits of the players as follows (from [EKKW22, Definition 3]). Note that the definition defines the best exit according to a function f. This is because

[3]We highlight that this pre-computation is not necessary for Algorithm 1 to converge because the operation of deflating that we will add includes this pre-computation.

we are interested both in the true best exits according to V as well as those exits that seem best according to the current estimate U.

Definition 4 (Best exit bexit)**.** *Given a set of states $T \subseteq \mathsf{S}$ and a function $f\colon \mathsf{S} \to [0,1]$, the best exit according to f from T of Maximizer and Minimizer, respectively, is defined as*

$$\mathsf{bexit}_f^{\Box}(T) := \max_{\substack{s \in T_{\Box} \\ (s,a) \text{ exits } T}} f(s,a)$$

$$\mathsf{bexit}_f^{\bigcirc}(T) := \min_{\substack{s \in T_{\bigcirc} \\ (s,a) \text{ exits } T}} f(s,a)$$

with the convention that $\max_{\emptyset} = 0$ *and* $\min_{\emptyset} = 1$*, and where $f(s,a)$ is the one step unfolding of f as in Equation* (3.2)*.*

Using these definitions, we can **characterize problematic ECs**. In the following, we let $T \subseteq \mathsf{S} \setminus \mathsf{F}$ be an EC. If Minimizer's best exit has a higher value than Maximizer's best exit (i.e. $\mathsf{bexit}_{\mathsf{V}}^{\bigcirc}(T) > \mathsf{bexit}_{\mathsf{V}}^{\Box}(T)$), then Minimizer's optimal strategy is to remain in the EC T and allow Maximizer to use the best Maximizer-exit. However, if instead of V we consider an over-approximation $\mathsf{U} > \mathsf{V}$, then staying in the EC looks more profitable for Maximizer, as it promises a higher value than that of the best available exit $\mathsf{bexit}_{\mathsf{V}}^{\Box}(T)$. Thus VI from above does not converge in such an EC because Maximizer always chooses staying actions, inducing a cyclic dependency at a too high value. In other words (see [EKKW22, Lemma 1]): for every m satisfying $\mathsf{bexit}_{\mathsf{V}}^{\bigcirc}(T) \geq m \geq \mathsf{bexit}_{\mathsf{V}}^{\Box}(T)$), there exists a fixpoint of the Bellman equations (Equation (3.1) and (3.2)) which is greater or equal than m for all states in T and for at least one state in T is equal to m.

Example 4 (Multiple fixpoints)**.** *In the SG in Figure 3.1a, Minimizer has no exit from the EC $T = \{\mathsf{p}, \mathsf{q}\}$, which by convention is equivalent to having an exit with value* 1*. Intuitively, upon reaching the EC, Minimizer cannot decrease the value but instead has to allow Maximizer to use the best Maximizer-exit.*

We have $\mathsf{bexit}_{\mathsf{V}}^{\Box}(T) = \mathsf{V}(\mathsf{q},\mathsf{c}) = 0.5$. Hence, for every $0.5 \leq m \leq 1$, we can find a fixpoint of the Bellman equations. In this simple example, it suffices to set $\mathsf{U}(\mathsf{p}) = \mathsf{U}(\mathsf{q}) = m$. This is a fixpoint because Maximizer prefers staying and getting m to using the exit. In particular, using U_0 and setting the estimate for both states to 1 *also is a fixpoint. This is why VI from above is stuck at a greater fixpoint and cannot converge to the value* V. $\triangle$

We have identified the key problem: whenever Minimizer's best exit has a higher value than Maximizer's best exit (or Minimizer has no exit at all), the Bellman equations have multiple fixpoints and thus VI from above does not converge. We call such ECs *bloated* [EKKW22, Definition 4], indicating that over-approximations stay blown-up at too high estimates. Bloated ECs indeed capture the whole problem. We have proven in [EKKW22, Theorem 1] that in SGs without bloated ECs there is only a single fixpoint of the Bellman equations, and thus VI from above converges. For the proof, we refer to the paper attached in Appendix A. Here, we only exemplify an unproblematic EC to give the intuition that indeed only bloated ECs are a problem.

Example 5 (Unproblematic ECs)**.** *Consider again the SG in Figure 3.1a, but modified so that the Minimizer state* p *has an exiting action leading to the sink state.*

Independent of the estimate for q*, Minimizer's best strategy is to play this new action instead of* a*, yielding a value of* 0 *in* p*. Thus, the Maximizer state* q *realizes that going to* p *will yield a value of* 0 *and always prefers to use action* c*. The only fixpoint of the Bellman equations then is* $\mathsf{V}(\mathsf{p}) = 0$ *and* $\mathsf{V}(\mathsf{q}) = 0.5$*, and VI converges to it from above and below.*

Note that an analogous argument can be made if the newly added action takes any value between 0 *and* 0.5*.* $\triangle$

Finding and deflating problematic end components

Now that we have identified the problem, we need a way to **algorithmically solve** it. Intuitively, we want to release the pressure of the too-high approximation by making sure Maximizer's states depend on exiting actions. Thus, we introduce the procedure of *deflating*: it reduces the estimates of all states in an EC T to the over-approximation of the best exit $\mathsf{bexit}^{\Box}_{\mathsf{U}}(T)$. Note that we replaced the function f in the index with U. This is because during VI we do not know the value of the exits (since we are just computing the values), but we only know an over-approximation.

Example 6 (Deflating)**.** *Assume that in iteration* i*, we are deflating the EC in Figure 3.1a. This means that we set the estimate for both states to* $\mathsf{bexit}^{\Box}_{\mathsf{U}_i}(T) = \mathsf{U}_i(\mathsf{q}, \mathsf{c})$*. This way, we obtain the sequence of upper bounds as indicated in the column* U^*_i*, and hence VI from above converges.* $\triangle$

This procedure is sound [EKKW22, Lemma 5] for any state set because no state can achieve a greater probability of reaching the target than the state in the set with the highest probability to do so, i.e. the best exit.

Yet, this is not sufficient to ensure convergence. Deflating always decreases to the estimate of the best exit from the whole EC. However, Minimizer can possibly restrict the movement inside the EC and prevent Maximizer from reaching the best exit.

Example 7 (Decomposing ECs to find simple ECs)**.** *Consider the EC in Figure 3.2. The letters* α *and* β *on the exiting actions denote the values of the action. Formally, they represent a transition that reaches a goal state with a probability of* α *(respectively* β*) and a sink state with the remaining probability. We have omitted this for readability.*

Depending on the relation between α *and* β*, Minimizer's optimal strategy changes. If* $\alpha < \beta$*, then the Minimizer state* r *will prefer action* a*. Maximizer state* s *has to use the exit* e *to gain* α*, as otherwise being stuck in the EC* $\{\mathsf{r}, \mathsf{s}\}$ *would result in never reaching the goal. State t has to use action* f *to achieve the highest possible value of* β*.*

However, $\{\mathsf{r}, \mathsf{s}, \mathsf{t}\}$ *is a bloated EC.*[4] *Hence, both Maximizer states will not realize that they have to leave, but rather choose to use the staying actions in order to keep the too-high upper bound. Deflating the whole EC will not resolve the problem because then we set the upper bound of all states to (our current over-approximation of)* β*. This is still insufficient for states* r *and* s*, which should be deflated to* $\alpha < \beta$*. We have to deflate only the EC* $\{\mathsf{r}, \mathsf{s}\}$*.*

[4]Unless $\alpha = \beta = 1$. In this corner case, the EC is not bloated, since Minimizer's best exit is equal to the value.

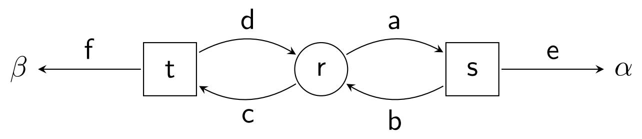


Figure 3.2: Example of an EC where not all states have the same value. α and β denote the value of the exiting actions. Depending on their relation, different subsets of $\{\mathsf{r}, \mathsf{s}, \mathsf{t}\}$ form a simple EC. Based on [EKKW22, Figure 3].

Dually, if $\alpha > \beta$, Minimizer will prefer action c *and we have* $\mathsf{V}(\mathsf{r}) = \mathsf{V}(\mathsf{t}) = \beta < \alpha = \mathsf{V}(\mathsf{s})$*. Hence, in this case we have to deflate the EC* $\{\mathsf{r}, \mathsf{t}\}$*.* △

We emphasize that this simple example shows the fundamental difficulties that we have when dealing with SGs: *decomposing into maximal end components (MECs) is not sufficient because different states in a MEC can have different values.* Moreover, the more-fine grained decomposition that we need for the computation depends on the values of the states, which are what we want to compute. This dependency prevents us from precomputing a decomposition of the SG as previous work does for MDPs [BCC+14, HM18].

As it is insufficient to deflate the whole EC, we have to find the correct subset to deflate. This subset depends on the strategy of Minimizer because Minimizer restricts which exit Maximizer can reach. In particular, Minimizer can prevent Maximizer from reaching the best exit of the whole EC.

Intuitively, we have to find the attractors of Maximizer-exits, i.e. those states that can reach the exit even if Minimizer plays optimally. All states in such an attractor have the same value, namely that of the best exit they can reach. We capture this notion with the following definition (from [EKKW22, Definition 5]):

Definition 5 (Simple end component (SEC))**.** *An EC T is called* simple end component (SEC) *if for all $s \in T$ we have* $\mathsf{V}(s) = \mathsf{bexit}^{\Box}_{\mathsf{V}}(T)$*.*

Equivalently, an EC is simple if it is an EC in the SG with all suboptimal Minimizer actions removed [EKKW22, Lemma 4]. Intuitively, this second definition captures the idea that under an optimal strategy, the best thing Minimizer can do is to allow Maximizer to choose the best exit from the SEC; Minimizer has already restricted the movement of Maximizer as much as possible. Moreover, this gives us a way to compute SECs [EKKW22, Algorithm 2]: we guess Minimizer's suboptimal actions according to the current under-approximation L. Since L converges to the true value V in the limit, eventually we guess correctly and find the true SECs. Deflating these infinitely often achieves convergence [EKKW22, Theorem 2].

The algorithm

We show the pseudocode of convergent VI from above and below, called bounded VI (BVI), in Algorithm 1. The key theorem of [EKKW22] is the following:

Theorem 1. *Algorithm 1 produces monotonic sequences* L *under- and* U *over-approximating* V *and terminates for every $\epsilon > 0$.*

Algorithm 1 Bounded value iteration algorithm from [EKKW22]

Require: SG G, precision ε

Ensure: functions $\mathsf{L}, \mathsf{U} \colon \mathsf{S} \to [0,1]$ such that for all states $s \in \mathsf{S}$, $\mathsf{U}(s) - \mathsf{L}(s) < \varepsilon$ and $\mathsf{L}(s) \leq \mathsf{V}_G(s) \leq \mathsf{U}(s)$

```
1:  for $s \in \mathsf{S}$ do                                        ▷ Initialization
2:      $\mathsf{L}(s) = 0$                                           ▷ Lower bound
3:      $\mathsf{U}(s) = 1$                                           ▷ Upper bound
4:  end for
5:  for $s \in \mathsf{F}$ do $\mathsf{L}(s) = 1$               ▷ Value of targets is determined a priori
6:  repeat
7:      $\mathsf{L}, \mathsf{U}$ get updated according to Eqs. (3.3) and (3.4)    ▷ Bellman updates
8:      for every SEC candidate $T$ do
9:          for $s \in T$ do
10:             $\mathsf{U}(s) \leftarrow \mathsf{bexit}^{\Box}_{\mathsf{U}}(T)$    ▷ Deflate, i.e. decrease U to best exit
11:         end for
12:     end for
13: until $\mathsf{U}(s) - \mathsf{L}(s) < \varepsilon$ for all $s \in \mathsf{S}$    ▷ Guaranteed error bound
14: return $(\mathsf{L}, \mathsf{U})$
```

For the formal proof, we refer to the proof of Theorem 2 in [EKKW22], which relies on several quite technical lemmata. Here, we summarize the **key ideas of the algorithm** and give intuitive reasons for its correctness.

In Lines 2, 5 and 7, the algorithm performs classical VI from below, i.e. it initializes the lower bound and then performs Bellman updates. However, as Example 2 showed, this converges only in the limit. To get a practical stopping criterion, we complement VI from below by introducing a second sequence of over-approximations. It is initialized in Line 3 and updated by Bellman updates in Line 7. If this sequence converged to the value, it would allow us to terminate when we reach ε-precision, see Line 13.

Using only Bellman updates, the over-approximation need not converge, see Examples 3 and 4. The problem are bloated ECs, or more specifically the SECs (Definition 5) inside the bloated ECs. We want to find these SECs and deflate them, removing the cyclic dependencies by making the estimates depend on exiting actions. However, finding the SECs is difficult, as they depend on the value. Thus, we cannot pre-compute them, but instead, we guess SEC-candidates based on the increasingly precise under-approximation (Line 8, using [EKKW22, Algorithm 2]). Then we *deflate* the SEC-candidates, see Example 6 and [EKKW22, Algorithm 3 and 4].

This is correct, as deflating is sound even for wrong SEC-candidates (see [EKKW22, Lemma 5]). It converges since we eventually guess correctly and then deflate the true SECs. Note that it still might be necessary to deflate infinitely often and the over-approximation converges only in the limit, as in Example 6.

Types of end components

Before concluding this section, we list important types of ECs, pointing out several special cases. Let T be an EC.

- Unproblematic ECs (see Example 5): these ECs are not bloated, i.e. $\mathsf{bexit}^{\bigcirc}_{\mathsf{V}}(T) \leq \mathsf{bexit}^{\square}_{\mathsf{V}}(T)$). If all ECs are unproblematic, Bellman updates suffice to get a convergent sequence of over-approximations [EKKW22, Theorem 1].
- Bloated ECs (see [EKKW22, Definition 4]): in these ECs we have $\mathsf{bexit}^{\bigcirc}_{\mathsf{V}}(T) > \mathsf{bexit}^{\square}_{\mathsf{V}}(T)$). They prevent the convergence of VI from above, and we deal with them via deflating. Note that a bloated EC can contain parts that are unproblematic.
- Simple ECs (see Definition 5): these ECs are the core of the problem. Every bloated EC contains a SEC, and every[5] SEC is bloated [EKKW22, Lemma 3]. We have provided two equivalent definitions: all states in SECs have the same value, and SECs are ECs under Minimizer's optimal strategies.
- ECs with no Maximizer-exit: these ECs have value 0, as Maximizer cannot force the play to exit the EC, and thus it can never reach the target (unless Minimizer plays suboptimally). Since we use the convention that $\max_{\emptyset} = 0$, deflating such an EC correctly sets the over-approximation to 0. Note that this is why we do not have to take special care of sink states Z when initializing the upper bound (as already highlighted in Footnote 3): either a sink state is part of an EC with no Maximizer exit and its over-approximation is deflated correctly, or it can only reach such ECs with no Maximizer exit, and Bellman updates will decrease the over-approximation to 0 within the first $|\mathsf{S}|$ iterations.
- ECs with no Minimizer-exit: unlike the previous case, these ECs can be highly non-trivial. Both the EC in Figure 3.1a and in Figure 3.2 have no Minimizer exit, but still need to be deflated. While in Figure 3.1a the whole EC is a SEC, Figure 3.2 shows that this need not be the case in general.
- ECs where all states belong to a single-player: such an EC is always simple. If all states belong to Minimizer, we have no Maximizer-exit and the value trivially is 0. If all states belong to Maximizer, we have no Minimizer-exit and need to deflate.

  Note that all ECs in MDPs belong to this category and thus are simple. This is why minimizing reachability in MDPs works by Bellman updates, while for maximizing reachability one needs more complex concepts [BCC^{+}14, HM18]. The idea of these papers is to *collapse* the ECs, replacing them with a single new state that can choose between all exits of the EC. This corresponds to our intuition that in a SEC, Maximizer can choose to use any exit and will prefer the best one. Note that this solution of collapsing is in principle applicable to SGs: collapsing SECs also yields a correct algorithm, see [EKKW22, Remark 1]. However, to compute SECs we need to know the value, which is what we want to compute in the first place. Hence, our algorithm is not based on collapsing because if we guess a SEC wrongly and then collapse it, we would get a wrong result. Thus, we apply the more conservative deflating instead.

[5]Except for the corner case where $\mathsf{bexit}^{\bigcirc}_{\mathsf{V}}(T) = \mathsf{bexit}^{\square}_{\mathsf{V}}(T)$. Such a SEC is not bloated.

3.1.3 Contribution: Simulation-based value iteration

As described in Section 1.2, we have to deal with the *challenge of scalability*, specifically the state-space explosion problem [BK08, Section 2.3]. This means that the state-space of the model grows exponentially in the number of tracked variables and often is so large that it is difficult or even infeasible to fit into memory. In these cases, it is not possible to use an algorithm working on the whole state-space at once such as Algorithm 1.

One approach to deal with this builds on the insight that often not all parts of the state-space are **relevant** or **likely**. A state is not *relevant* if it is only reached under a suboptimal strategy. Then, computing its exact value is not necessary when we are only interested in the value of the initial state, see [EKKW22, Example 9]. A state is not *likely* if it is only reached with low probability, e.g. if several errors have to occur together at once. When we are interested in ε-precise results, we can ignore states that are reached with low probability because they do not significantly affect the value of the initial state. This idea is investigated in more detail in [KM20], where the authors identify the *core* of a model, i.e. the states that are unlikely to be left. This core can be several orders of magnitude smaller than the whole model.

A way to approximate the relevant and likely part of the state-space is to **run simulations** of the model. We do this by starting in the initial state and then in every state s_i picking an action $a \in \mathsf{Av}(s_i)$ and sampling a successor s_{i+1} according to the probability distribution $\delta(s_i, a)$. For more details on the simulations, see [EKKW22, Algorithm 5 and Example 6]. Using these simulations, we construct a partial model of the system and only update this instead of the whole state-space. We focus on the relevant part of the state-space by using heuristics that pick promising actions, e.g. actions with a high upper bound. We explore the likely part of the state-space by sampling according to the transition function.

Even though the partial model is generated based on heuristics and randomness, the resulting approach gives correctness guarantees and terminates almost-surely [EKKW22, Theorem 3]. The proof is quite similar to that for the simulation-based variant of VI for MDP [BCC$^+$14]. The key difference is that ECs are not collapsed anymore, but instead deflated. This however does not significantly change the argument. For the pseudocode of the simulation-based algorithm as well as the full proof, we refer to [EKKW22, Section 5].

As a final remark, we highlight that this algorithm combines techniques from machine learning and formal verification, benefiting from the advantages of both: the heuristics for selecting the relevant part of the state-space are from machine learning, thus they are good at quickly identifying the core of the model. However, on their own, they would not provide guarantees on the result. By complementing the heuristics with a guaranteed computation of under- and over-approximation, we ensure that the result is correct. Moreover, the computed guaranteed bounds can be used to inform the heuristics and improve their guesses.

3.1.4 Related work

For a thorough description of VI in many settings and its various properties, we refer the interested reader to the textbook [Put94] or the survey paper [CH08]. VI for solving SGs (and MDPs) with reachability objectives, as well as various other objectives, takes at most exponentially many iterations [CH08]. There are models where VI actually needs this exponential number of iterations [BKN$^+$19]. It is the default algorithm in the well-known probabilistic model checkers PRISM [KNP11] and Storm [DJKV17], as well as the only one available in PRISM-games [KNPS20].

For a long time, the **stopping criterion** used in practice checked whether in all states s the difference between $\mathsf{L}_i(s)$ and $\mathsf{L}_{i-1}(s)$ was small. This naive approach could however return arbitrarily wrong results [HM18]. In 2014, two independent teams of researchers developed the idea of collapsing ECs in MDPs in order to eliminate the cyclic dependencies [BCC$^+$14, HM18]. The PhD thesis [Ujm15] discusses an extension to SGs where the interleaving of players is severely limited, i.e. in every EC, all states must belong to exactly one player. For more description of collapsing and why it is not sufficient for arbitrary SGs, see the last paragraph of Section 3.1.2 and [EKKW22, Section 3]. Previous to our work, there were two stopping criteria for SGs, both of which are however impractical. The algorithm from [CH08], which we describe at the end of Section 3.1.1, always requires an exponential number of steps. Alternatively, one can transform the SG into a stopping one as described in [Con92, Section 3]. Stopping SGs do not contain ECs, hence also no bloated ECs and thus VI converges from above and below. However, as we discuss in Section 3.3.2, this transformation is inherently impractical because it introduces probabilities that are smaller than machine precision.

There exists various **heuristics** to speed up the convergence of VI. For MDP, recently optimistic VI [HK20] and sound VI [QK18] have been developed and could be extended to SGs. Further, for SGs, a combination of VI with retrograde analysis yields the deterministic algorithm with one of the best currently known upper bounds on runtime [IM12]; it is exponential only in the number of *actions with random outcome*,[6] i.e. actions $a \in \mathsf{Av}(s)$ such that $\delta(s, a, s') \notin \{0, 1\}$ for some $s, s' \in \mathsf{S}$. However, this algorithm requires the SG to only use probabilities 0, 1 or 0.5. We can transform every SG to satisfy this constraint at the cost of a polynomial blow-up of the state-space, see the discussion in Section 3.3.1.

Another idea for heuristics is to use *asynchronous* VI. Its key insight is that we do not have to update all states of the SG in every iteration, but can focus on ones that are important. There are various ways to do this. Two of these are Gauß-Seidel VI and topological VI [DMWG11], which we describe in Section 3.4.2. The simulation-based algorithm we propose in Section 3.1.3 also is an instance of asynchronous VI: we update states in the order that is suggested by our simulations. Our algorithm is based on the one for general MDPs [BCC$^+$14], which in turn extends the one for stopping MDPs [MLG05]. We mention that there are further machine learning techniques which have been used for solving SGs [BT00, LL08, BBS08, WT16, TT16, AY17]. These, however, usually provide no or weak guarantees. Note that some of the cited papers consider more expressive specification or slightly different systems.

The bounded VI variant given in Algorithm 1 has been improved in [PTHH20] and generalized to settings with limited information [AKW19a] (see Section 4.1), multiple objectives [ACK$^+$20] (see Chapter 5) and to concurrent SGs [EKR19].

[6] In their formulation of SGs, they use three kinds of states: Maximizer, Minimizer and Random, and then count the number of random vertices. Since we use a probabilistic transition function instead of the probabilistic states, we have to adapt the definition.

3.2 Strategy iteration (SI)

3.2.1 State of the art

Algorithm 2 Strategy iteration algorithm as given in [KRSW22]

Require: SG G

Ensure: Optimal Maximizer strategy σ and function $\mathsf{L} : \mathsf{S} \to [0,1]$ with $\mathsf{L} = \mathsf{V}_G$

```
1: $\sigma' \leftarrow$ arbitrary Maximizer *attractor strategy*        ▷ Proper initial strategy
2: **repeat**
3:     $\sigma \leftarrow \sigma'$
4:     **for** $s \in \mathsf{S}$ **do**
5:         $\mathsf{L}(s) \leftarrow \inf_\tau \mathbb{P}_s^{\sigma,\tau}(\Diamond \mathsf{F})$        ▷ Solve MDP for opponent
6:     **end for**
7:     **for** $s \in \mathsf{S}_\square$ **do**
8:         **if** $\sigma(s) \in \arg\max_{a \in \mathsf{Av}(s)} \mathsf{L}(s,a)$ **then**
9:             $\sigma'(s) \leftarrow \sigma(s)$        ▷ Only change $\sigma$ on strict improvement
10:        **else**
11:            $\sigma'(s) \leftarrow$ any element of $\arg\max_{a \in \mathsf{Av}(s)} \mathsf{L}(s,a)$
12:        **end if**
13:    **end for**
14: **until** $\sigma = \sigma'$
15: **return** $(\sigma, \mathsf{L})$
```

The approach of *strategy iteration* (SI), see e.g. [FV97], computes a sequence of improving strategies, not a sequence of approximations of the value vector as VI does. The pseudocode for SI is given in Algorithm 2. Starting from an arbitrary memoryless pure strategy of one player (Line 1, in our pseudocode it is Maximizer), we repeatedly compute the global best response of the opponent (Line 5) and then locally improve the Maximizer strategy (Line 11). The resulting sequence of Maximizer strategies is monotonically improving and converges to an optimal strategy [CdAH13, Theorem 3]. The value function and an optimal Minimizer strategy are computed in the last iteration as the best response to the optimal Maximizer strategy (Line 5). Both can be returned, but we only return the Maximizer strategy and the value function, since the optimal Minimizer strategy is only implicitly computed and hence not visible in the pseudocode. The algorithm terminates after at most exponentially many iterations, as there are exponentially many memoryless pure strategies.

We mention several interesting technicalities.

- In non-stopping SGs (ones containing ECs in $\mathsf{S}_?$), the initial Maximizer strategy has to be *proper*, i.e. ensure that either a goal or a sink state is reached almost-surely; it must not stay in some EC, as otherwise, the algorithm might not converge. In Algorithm 2, we use the construction of the *attractor strategy* [CdAH13, Section 5.3] to ensure that our initial guess is a proper strategy (Line 1). It first analyses the game graph to find the sink states Z. Then, it performs a backwards breadth-first search, starting from the union of the sets of goal and sink states. A state that is discovered in the i-th iteration of the search has to choose some action that reaches a successor state discovered in the $(i-1)$-th iteration with positive probability. By construction, such an action and successor state exists. The resulting strategy is proper because it has a positive chance of progressing towards the set of goal or sink states in every step and thus reaches either a goal or sink state almost-surely.

- One might question why one strategy is the global best response while the other is only locally updated. This is highly relevant, as otherwise the sequence of strategies can contain cycles and we do not necessarily converge to the optimal one [Con93, Section 2.2].

- For termination, it is necessary that we only update the strategy if switching strictly improves the value, as otherwise, we might cycle infinitely between equivalent strategies. This is realized by Lines 8 and 9 which ensure that we leave the strategy unchanged if it is still optimal.

3.2.2 Contribution: Hyper-parameters and warm start

Solving the **opponent-MDP** in Line 5 is typically done using linear programming, see e.g. [FV97, Section 6.3]. However, this is not necessary [KRSW22, Section 4.2]. One can equivalently solve the MDP using SI (computing a sequence of improving opponent strategies) or even bounded VI. For the latter to be practical, we want to utilize the practical stopping criterion described in the previous section. Since we only need to know the locally optimal actions, we can potentially avoid computing the precise values, but instead, compute until the lower bound of one action is higher than the upper bound of all other actions. So the solver for the opponent-MDP is a hyper-parameter of SI, and depending on the structure of the MDP, using other solution algorithms can be advantageous. Note that using value iteration to solve the opponent MDP is similar in spirit to the notion of modified policy iteration in [Put94].

Further, the fact that we fix the Maximizer strategy first is arbitrary. It might be advantageous to rather compute a sequence of improving Minimizer strategies, e.g. if that player has fewer strategies, i.e. fewer states and actions per state.

As a final idea, the initial strategy affects how fast we are, since if we are closer to the optimum, we need fewer iterations. We suggest using available prior knowledge to improve the initial guess; this is called **warm start** [KRSW22, Section 4.3].

- If we have domain knowledge about the model, for example, the intuition that in case of a transmission error in a protocol we want to resend a package, we can include this in our initial strategy.

- We can utilize fast pre-computations, e.g. of the set of sink states Z.

- We can improve the MDP-solving in Line 5 by giving it the knowledge of the outer algorithm: we anyway save the estimate L of the previous iteration and, since the strategies of Maximizer get monotonically better, L is certainly a lower bound for the values in the MDP.

- Before starting the SI, we can run unguaranteed VI for a fixed number of iterations. This yields a rough estimate of the value function which we use to inform our choice of the initial strategy. This is similar in spirit to the idea of optimistic VI [HK20]: utilize the ability of VI to usually deliver tight lower bounds quickly and then verify them.

3.2.3 Related work

Historically, SI was first proposed in [HK66] for the class of irreducible concurrent SGs. It was adapted for turn-based SGs with average reward objectives in [VTRF83]; this algorithm is also described in the textbook [FV97]. Since reachability objectives can be encoded as average reward objectives, this was the first variant of the algorithm able to deal with the problem we consider. The formulation explicitly for the setting of simple SGs was given in [Con93], together with a randomized version.

Regarding **complexity**, we know that the algorithm can take at most exponentially many iterations because there are at most exponentially many memoryless pure strategies and the algorithm is monotonically improving. For the lower bound, we know that the associated decision problem is PSPACE-complete for both SI with one update per iteration (called single-switch strategy iteration [FS15]) as well as with all possible updates per iteration (called all-switches strategy iteration [FS18]). Both proofs build on the seminal work by Friedman [Fri11], which also shows that exponentially many iterations can be required in the worst case. Further, even for MDPs this exponential number of iterations can be necessary [Fea10]. In contrast, if we consider SGs with a constant discount factor, then SI is in fact strongly polynomial [HMZ13].

We now sketch the **development of various variants of SI over the years**. We use $\tilde{\mathcal{O}}$ to denote asymptotic runtime omitting polynomial terms as was done in [DG11]. Note that our contributions for SI are practical heuristics that do not affect the asymptotic runtime.

After the randomized variant from [Con93], a first randomized subexponential algorithm was introduced in [Lud95]. For an SG with n states, its expected number of iterations is $\tilde{\mathcal{O}}(2^{\sqrt{n}})$. In [Som05], a variant of SI is given which is based on a geometric interpretation of node valuations; the paper only provides an exponential upper bound on the number of iterations.

The influential paper [GH08] gave an algorithm that has good runtime on SGs with little randomness. Denoting the number of actions with random outcome as r (or equivalently the number of random states, see Footnote 6), the runtime is in $\tilde{\mathcal{O}}(r!)$. Based on this, [DG11] gave another subexponential algorithm running in $\tilde{\mathcal{O}}(\sqrt{r!})$ and [ACS19] combined the ideas from [Lud95] and [GH08] to get expected running time $2^{\mathcal{O}(r)}$. Note that the algorithm based on value iteration from [IM12] also builds on the ideas of [GH08]. Its runtime is $\tilde{\mathcal{O}}(2^r)$, but unlike [ACS19], it requires all probabilities to be either 0, 1 or 0.5, while [ACS19] works with arbitrary random states.

In a different direction, [CdAH13] gave a deterministic SI algorithm for concurrent SGs. A careful analysis of the algorithm from [CdAH13] in [IM12] showed that it runs in $\tilde{\mathcal{O}}(4^r)$. The standard SI approach was analysed in detail in [TVK11] and shown to converge in $\mathcal{O}(2^n/n)$, improving the trivial exponential bound. The authors also provide a randomized variant expected to take time in $\mathcal{O}(2^{0.78n})$. The SI variant from [dM21] is the first deterministic one to have runtime in $\tilde{\mathcal{O}}(2^{cn})$ with $c < 1$. Recently, [AdMS21] provided a generic SI method and hence a uniform framework to reason about all the SI variants that exist. They give general arguments that reprove many of the aforementioned ad-hoc results.

3.3 Quadratic programming (QP)

3.3.1 State of the art

The idea of *quadratic programming* (QP) is to encode the structure of the SG in the constraints of an optimization problem. The only global optimum of the objective function is attained if and only if the variable for every state is set to the value of that state [Con93, Section 3.1]. The proof, as well as the construction of the QP, rely on the game being in a certain normal form, which consists of four conditions.

- no-one-act: If $|\mathsf{Av}(s)| = 1$, then s is a target or a sink.
- half-probs: For all $s, s' \in \mathsf{S}, a \in \mathsf{Av}(s) : \delta(s, a, s') \in \{0, 0.5, 1\}$.
- two-act: For all $s \in \mathsf{S} : |\mathsf{Av}(s)| \leq 2$.
- stopping: There are no ECs in G (except for the sink and goal states).

The advantage of two-act and no-one-act is that the objective function of the QP requires every (non-sink and non-target) state to have exactly 2 actions. The constraint half-probs occurs because [Con93] used a game formulation with average nodes, which slightly simplified the proofs. Finally, stopping was necessary because end components introduce multiple fixpoints to the Bellman equations (see Example 4), and thus the solution of the QP would not be unique anymore. For every SG, we can construct a polynomially larger one in normal form, such that the original value can be obtained by solving the SG in normal form. The transformation to normal form was originally given in [Con92] and we give an extensive description in [KRSW22, Section 3.3.2].

The QP in Figure 3.3 computes the value of an SG in normal form; it is the same as in [Con93], but adapted to fit our notation. The constraints encode the structure of the

$$\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{\substack{s\in\mathsf{S}\\ \mathsf{Av}(s)=\{a,b\}}} (\mathsf{V}(s) - \mathsf{V}(s,a))(\mathsf{V}(s) - \mathsf{V}(s,b)) & \\
\text{subject to} & \mathsf{V}(s) \geq \mathsf{V}(s,a) & \forall s \in \mathsf{S}_\Box : |\mathsf{Av}(s)| = 2, \forall a \in \mathsf{Av}(s) \\
& \mathsf{V}(s) \leq \mathsf{V}(s,a) & \forall s \in \mathsf{S}_\bigcirc : |\mathsf{Av}(s)| = 2, \forall a \in \mathsf{Av}(s) \\
& \mathsf{V}(s) = 1 & \forall s \in \mathsf{F} \\
& \mathsf{V}(s) = 0 & \forall s \in \mathsf{Z}
\end{array}$$

$$\text{where } \mathsf{V}(s,a) = \left\{ \begin{array}{ll} \mathsf{V}(s') & \text{for } \mathsf{Post}(s,a) = \{s'\} \\ {}^1\!/\!{}_2\mathsf{V}(s') + {}^1\!/\!{}_2\mathsf{V}(s'') & \text{for } \mathsf{Post}(s,a) = \{s', s''\} \end{array} \right\}$$

Figure 3.3: The QP to compute the value of a simple SG in normal form from [Con93], adapted to our notation. Taken from [KRSW22, Section 3.3.1].

SG: every Maximizer state has a value higher than its best action, dually every Minimizer state uses the action with the lowest value, and the value of goal and sink states is fixed. Note that the definition of $\mathsf{V}(s, a)$, in essence, is the Bellman equation (3.2); it is restricted to the probabilities 0, 1 or 0.5 due to the constraint half-probs. To minimize the objective function, the summand for every state has to be minimized. Since no summand can be negative, the optimum of the objective function is 0. It is obtained when for every state s, its value $\mathsf{V}(s)$ is equal to the value of at least one of its actions $\mathsf{V}(s, a)$ or $\mathsf{V}(s, b)$. Thus, the program computes the true value function V; see [Con93, Section 3.1] for the full proof. The program is quadratic because by two-act all states (except goals and sinks) have two actions and hence the summand for every state is a quadratic term.

3.3.2 Contribution: Avoiding the transformation to normal form

The transformation into normal form blows up the size of the SG and thus the time and memory consumption of the solution algorithms. Thus, for each constraint, we developed an alternative, so that in the end we have an optimization problem that works on the original game. Here, we give a high-level summary of the ideas; see [KRSW22, Section 4.1] for the full description and technical proofs.

The first two constraints, no-one-act and half-probs, can easily be avoided. For no-one-act, the transformation into normal form adds a dummy action that is never used. Instead, one can directly encode in the QP that every state with one action has the value of its successor. For half-probs, one can allow the QP to use arbitrary probabilities by using the Bellman equation (3.2) instead of restricting it to only use probabilities 0, 1 or 0.5.

The constraint two-act is indeed necessary to obtain a QP since the objective function relies on the fact that every state has at most two successors. However, one can use a higher-order program (HOP) to avoid this transformation. To this end, we replace[7] the quadratic term for every state s with the product $\Pi_{a \in \mathsf{Av}(s)}(\mathsf{V}(s) - \mathsf{V}(s, a))$. Hence, the degree of the terms depends on the number of actions per state. In the experimental Section 3.4, we evaluate both a QP where we have transformed the SG in order to satisfy two-act, as well as a HOP on the original SG.

Finally, the constraint stopping is difficult to avoid, but practically the most important one. To understand the problem, we first summarize the idea to transform an SG into a polynomially larger stopping SG from [Con92]. In essence, we modify every transition, so that with a small probability of ε the transition leads to a sink state. Hence, the SG becomes stopping, since a sink state is reached almost-surely. If ε is chosen sufficiently small, then the difference between the value of the modified and original SG is so small that the original value can be obtained by rounding; this works because the value of the SG is a rational number whose denominator can be bounded by the size of the SG.

The probability ε has to be very small; a conservative upper bound is $(\frac{1}{4})^{\mathsf{S}}$ since the difference between original and transformed SG for every state has to be smaller than this number [Con92, Lemma 8]. According to the IEEE 754.2019 standard,[8] the commonly used double machine precision is 10^{-16}, so already for 27 states, the necessary ε becomes

[7]This description omits a technical detail.

[8]`https://standards.ieee.org/content/ieee-standards/en/standard/754-2019.html`

smaller than machine precision. Thus, this transformation to a stopping SG is inherently impractical.

Our approach introduces additional constraints for every maximal EC. Similar to the idea of deflating in Section 3.1.2, these constraints ensure that all states in the EC depend on an exiting action. For the description of the algorithm constructing the constraints, see [KRSW22, Section 4.1.4 and Algorithm 3]. Note that our algorithm introduces exponentially many constraints, so the resulting QP or HOP is exponential in the size of the MECs and hence potentially exponential in the size of the SG. Still, as we show in the experimental evaluation, this exponential approach is a lot more practical than the original idea of introducing the ε-transitions.

We summarize the contribution of this section in the following theorem:

Theorem 2. *We can transform an arbitrary SG G into a polynomially larger SG G' such that G' satisfies* two-act *and* $\mathsf{V}_{G'} = \mathsf{V}_G$.

For an SG G' satisfying two-act*, we can use a QP to compute* V'_G.

For an arbitrary SG G, we can use a HOP to compute V_G *without modifying G.*

Proof. The first statement is taken from [Con93, Section 4]. We repeat it to emphasize that even though the QP is only applicable to SGs satisfying two-act, it can still be used in a polynomial solution algorithm by combining it with this transformation.

The second statement is equivalent to saying that we can drop the constraints half-probs, no-one-act and stopping. We have discussed the intuition for this above; for the proofs, see [KRSW22, Lemmata 2, 3 and 4].

The third statement is equivalent to saying that we can also drop the two-act constraint if we allow for the program to have a degree higher than two. For the proof, see [KRSW22, Lemma 1]. □

3.3.3 Related work

To the best of our knowledge, using QP as a solution method for simple SG has not been investigated since the initial suggestion in [Con93]. We believe that this was due to the fact that it was accepted as common knowledge that linear programming (LP) and QP are not efficient. For example, [GH08] states that using QP "may have prohibitive algorithmic cost and makes the implementation tedious." [Page 3] and LP "requires high-precision arithmetic" [Page 4]. In the case of MDP, we can use LP instead of QP, which takes polynomial time and is asymptotically optimal and thus theoretically superior to VI or SI. "In practice, however, most probabilistic verification tools use (approximate) iterative numerical methods, such as value iteration [Put94], since they scale to much larger systems and are amenable to symbolic (BDD-based) implementations. Value iteration can also be used for time-bounded (finite-horizon) properties, which is impractical with LP" [FKP12, Pages 1 and 2]. Similarly, [KM17] reports that "compared to linear programming (LP), SI scales much better. [...] [LP] procedures are not very useful in practice" [Page 2]; and [HM18] states that LP procedures "seem unable to scale to large systems, though some results have been obtained recently by mixing it with numerical methods [Gir14]" [Page 2]. An experimental comparison in [ACD^{+}17] showed that LP indeed did not scale well to models with many maximal ECs.

Nonetheless, we believe QP and LP should be investigated. One motivation is that convex QPs are solvable in polynomial time [KTK80] and hence might allow to find a polynomial algorithm for solving simple SGs. The convexity is important, since solving non-convex QPs is NP-hard [Sah74].

Additionally, even if the problem was NP-hard, many solvers have been developed for these kinds of optimization problems and are still actively advanced, see [AAK17] for a recent comparison. Moreover, there are lots of ways in which one can improve the performance of an optimizer. In particular, [SLYD20, Section 2] discusses several works that improve the search heuristics, the hyper-parameters of the algorithms or methods to identify substructures. Finally, the single paper investigating LP for MDPs problems we are aware of [Gir14] already improved the performance of LP significantly. However, we are not aware of any improvement for solving simple SGs with QP in the past 30 years, and hence it is not surprising that it currently is not as performant as VI and SI, which have been investigated a lot, see Sections 3.1.4 and 3.2.3.

As a side note, we point out that simple SGs are an LP-type problem, and using this formulation yielded the first strongly subexponential algorithm for the problem [Hal07].

3.4 Experimental analysis

In this section, we summarize the experimental results of [EKKW22] and [KRSW22]. The goal of [EKKW22] was (i) to show that when complementing VI with deflating to obtain guarantees, the overhead often is negligible and (ii) to compare the method of deflating with the established approach of collapsing ECs in MDPs. The goal of [KRSW22] was to compare the algorithms and find the most practical.

We first describe the algorithms and case studies we used for the analysis, then comment on the additional optimizations we considered and finally give an overview of the experimental results of both papers. Note that, as these results are quite extensive, we only focus on the key messages and refer to the papers in the appendix for the detailed analysis.

3.4.1 Algorithms and case studies

We have implemented all our **algorithms** as an extension of PRISM-games [KNPS20].[9] This includes bounded and simulation-based VI, SI with different solvers for the opponent MDP and the option of a warm start, as well as QP and HOP without the transformation to normal form. PRISM-games already contained the unguaranteed VI for simple stochastic games. We have also added several other things to compare to, for example variants of QP based on only lifting some of the constraints. For more details, see [KRSW22, Sections 5.1.1 and 5.1.2]. The code is available in a GitHub repository[10] and at the time of writing, we are still actively adding more sophisticated algorithms and technical improvements.

As **case studies**, we mainly used MDPs and SGs with reachability objectives from the PRISM benchmark suite [KNP12] and the PRISM-games case study website.[11] Moreover,

[9]Note that our implementation is based on the most recent version available at the time of implementing, namely PRISM-Games 2 [KPW16].

[10]`https://github.com/ga67vib/Algorithms-For-Stochastic-Games`

[11]`http://www.prismmodelchecker.org/games/casestudies.php`

for the comparison in [KRSW22], we also included models we developed in our previous work [CKWW20] and several handcrafted models of interesting theoretical corner cases. The prism model files of all considered SGs are collected in our GitHub repository (see Footnote 10) and a more detailed description is given in [EKKW22, Section 6.1] and [KRSW22, Section 5.1.3].

3.4.2 Optimizations

We structured the code such that the following optimizations are easily applicable to all variants of certain algorithms, as well as new implementations thereof.

- Topological approach: Working on the whole graph of the model at once can be suboptimal because certain states, e.g. those closer to the goal, have already converged and need not be updated anymore. Thus, instead of considering the whole model, we can instead compute the topological ordering of strongly connected components (see Definition 2 and its discussion) and then proceed in a backwards fashion, SCC by SCC. This idea was suggested for VI in MDPs [DMWG11] and for SI in MDPs [KM17]. The idea can be extended to SGs in a straightforward manner, see [KRSW22, Section 4.4].

  We have implemented a topological version of all our algorithms working on the whole state-space, namely bounded VI as well as all variants of SI, QP and HOP. The optimization is not applicable to the simulation-based algorithm.

- Gauß-Seidel VI: This variant of asynchronous VI immediately uses the results of the current iteration when possible. Formally, in an update according to Equation (3.4), we use $\mathsf{L}_n(s')$ instead of $\mathsf{L}_{n-1}(s')$ if a new estimate $\mathsf{L}_n(s')$ already is available. This can speed up convergence because results are propagated more quickly, see [EKKW22, Example 8]. This optimization is available for all VI variants in our code.

- Deflating less often: In bounded VI, the operation of deflating, or more specifically of finding SEC-candidates, is quite costly. Hence, one can only perform it every n iterations instead of every time. Of course, for high n, this can also slow down the computation since we might only be wasting time until the next deflation happens, but it can also speed up computation. The analysis in [EKKW22, Section 6.2] indicates that using $n = 10$ is a reasonable sweet spot.

- Caching: Most of our algorithms have to use maximal ECs repeatedly. However, it suffices to compute them only once and cache them for all further iterations. Interestingly, this simple engineering improvement on a very practical level can have as much or more impact than a theoretical improvement.

- We suggest two further ideas related to dealing with ECs in VI, see [EKKW22, Section 6.2]. However, they both did not speed up the computation. The first idea is to collapse SECs when our estimates suffice to prove that the EC is indeed simple. The second idea is to prefer actions exiting an EC when running simulations for the simulation-based variant. Both ideas seem useful on a theoretical level. We conjecture that, by using a better implementation of collapsing or more suitable data structures for tracking exiting actions, these optimizations could still have merit, even though in our experiments they did not.

3.4.3 Results

The results of [EKKW22] showed that the overhead of using deflating to obtain guarantees often is negligible and that deflating is not much worse than collapsing in MDPs. Note that every EC only can be collapsed once, while it potentially has to be deflated very often. Thus, especially if the convergence rate of VI is low on a certain model, collapsing is preferable. This is also visible since the approaches using deflating are always slower than those based on collapsing, see [EKKW22, Section 6.3].

The results of [KRSW22] are very extensive, including comparisons between the algorithm classes as well as within them, analysing the impact of the different optimizations and variants. Here, we give the two main conclusions.

- **There is no clear winner.** In our overview (see [KRSW22, Section 5.5.1]), we see that all variants of both VI and SI can solve most of the case studies in time, namely between 26 and 31 of the 34 considered ones. Their verification times are comparable for all but the hardest models. In contrast, all but the most optimized variants of QP are significantly worse, timing out after 15 minutes for around half the case studies. The original approach of [Con93] can only solve 3 case studies. The best variants of QP and HOP perform similar to VI and SI, solving 24 respectively 25 solved problems and taking a similar time to do so.

  Concerning the memory usage, all algorithms except for the unoptimized QP are reasonable, see [KRSW22, Section 5.5.2].

  Looking in a bit more detail, we see that the algorithms struggle with different case studies: for example, on the model "hm_30" every VI variant times out after 15 minutes, while SI and QP can succeed in less than a second. On the other hand, for "BigMec_e4", VI needs around four minutes, while SI and QP time out.

- **The structure of the considered model is very important.** Both of the models mentioned in the previous point are rather small, one with 61 and one with 20,003 states. Still, they are very hard for the algorithms because hm_30 contains many probabilistic cycles and BigMec_e4 contains a huge maximal EC. On the other hand, almost all of the large models we consider are solved within less than 80 seconds, often even less than 20, even though their state-spaces have sizes of 100,000 or even 480,000. Thus, the structural properties — number of probabilistic loops, minimum occurring probabilities, number and size of maximal ECs — are more important than the size of the model.

3.5 Summary, discussion and outlook

Summary There are three classes of algorithms for solving simple stochastic games: value iteration, strategy iteration and quadratic programming. We have explained all of them and presented several modifications directed at improving the verification time. Moreover, for VI we have introduced the first practical stopping criterion.

Our experimental analysis revealed that the structure of the considered case study is very important for the performance of the algorithms, and thus there is no algorithm that is best in all cases.

Discussion Before discussing the implications of our experimental evaluation, we recall relevant theoretical properties of all algorithm classes in order to set the practical results into a more general context.

Our theoretical comparison focusses on two properties: *precision* and *complexity.*

- **Precision:** VI converges only in the limit. The precise algorithm based on rounding from [CH08] is best-case exponential and hence impractical. Thus, a practical implementation of VI is approximate by design, only giving ε-guarantees.

  In contrast, SI and QP theoretically are precise algorithms. Still, practical implementations are usually not. For SI, the precise guarantee requires precisely solving the MDP after fixing a strategy. This can be achieved by either using linear programming or SI for MDPs, provided that one precisely solves the equation system of the Markov chain after fixing both strategies. Both of these approaches are not supported by PRISM, but instead, either the MDP or the Markov chain are solved using VI, thus rendering the overall algorithm imprecise, too. We have added both approaches to the code in our GitHub repository (see Footnote 10) after the publication of [KRSW22].

  The precision of QP and HOP depends on the solver that we use. To the best of our knowledge, the ones that performed best in our experiments — gurobi[12] and MINOS[13] — do not offer a precise computation mode.

  Finally, note that all computations using floating-point precision introduce rounding errors. A way to address this was suggested in [Har22] but has not been implemented for our algorithms so far.

- **Complexity:** While all algorithms take at most exponential time, for VI and SI this exponential number of iterations can also be necessary. In [BKN+19], the authors proved that the decision problem that VI solves in every iteration is EXPTIME-complete. Similarly, for SI the associated decision problem is PSPACE-complete [FS18] and an exponential number of iterations is necessary [Fri11].

  In contrast, QP still has the potential to yield a polynomial algorithm. This could be achieved by finding a polynomially sized, convex QP, since solving convex QPs can be done in polynomial time [KTK80]. Currently, our QP is exponential and it is unclear for both our and the previous polynomially sized QP from [Con93] whether they are convex. If they are not, then solving them is known to be NP-hard [Sah74].

Our versions of VI, both bounded and simulation-based, have one advantage over SI and QP: they are **anytime algorithms**, which means at any time during the computation we

[12]`https://www.gurobi.com/`

[13]`http://sbsi-sol-optimize.com/asp/sol_product_minos.htm`

can stop and get a result with an estimate of its precision. In contrast, SI and QP have to finish the computation to provide precision guarantees.

Now that we have provided the theoretical context, we discuss the **practical implications** of our results. We were surprised to find that the performance of QP often is comparable to that of the other algorithms. It was accepted as common knowledge that linear programming and QP are less efficient, as discussed in Section 3.3.3. We view our results as an indication that it might be practically worthwhile to investigate linear and quadratic programming. In particular, we saw that switching the solver from CPLEX to Gurobi had a huge impact on the running time. This shows that the verification time of simple SGs can be reduced by any progress in the area of solvers for QPs, be it fundamental or via clever engineering. Moreover, while VI and SI were researched and improved over the past decades, as discussed in Sections 3.1.4 and 3.2.3, we are not aware of any similar works in the area of QP. Thus, there might be a lot of room for theoretical improvements in the encoding of the SG as QP. Our work has already pushed QP's performance from being practically useless to being comparable on many models.

Similarly, we were surprised that SI performed on par with VI because it was reported as similar or sometimes considerably slower than VI in [FKNP11]. Also, VI is the only algorithm implemented in PRISM-Games [KNPS20].

Outlook We see potential for **practical improvements** of the algorithms in the following directions.

- Implement and evaluate the additional variants of VI and SI that are mentioned in Sections 3.1.4 and 3.2.3. Additionally, combine the verification algorithms with heuristics based on machine learning, as we did with the simulation-based variant of VI in Section 3.1.3.
- Reduce QP or HOP verification time by using better solvers, tuning the hyper-parameters of the solvers or improving the encoding of the problem into a QP/HOP.
- Find clear correlations or even causalities between structural properties of a case study and the performance of the algorithms. From this, develop a portfolio solver that first analyses the structure and then picks the most suitable algorithm.

For properly evaluating the algorithms, we also believe that it is important to improve the setup for practical experiments, i.e. the available case studies and implementations. Currently, we only know few **case studies** (less than 15). Although they are scalable, it is possible that we tune algorithms for these specific cases rather than the general problem. Having more case studies or a good random generation is required. This could take a form similar to or be included in the quantitative verification benchmark set [HKP^{+}19], which is a web application that provides scalable model files and metadata for many probabilistic models. It includes Markov chains and MDPs, but not SGs. All of the **implementations** of our algorithms are an extension of PRISM-games. However, as competitions like QComp [BHK^{+}20] show, (i) having different implementations for the same algorithm can make a huge difference and in particular (ii) PRISM — the basis model checker for our implementation — is often outperformed by other model checkers.

Note that there is a vicious cycle: If there are few case studies, it is hardly worth investing in a well-engineered, performant implementation. But without these implementations, there is little incentive to create more case studies.

Now that we have discussed directions for practically oriented future work, we turn our attention to **theoretical improvements**. A natural next step is to extend the given algorithms to other settings. This includes settings with limited information or multiple objectives, which are discussed in the next two chapters. In this outlook, we focus on **more expressive specification formalisms**, see Section 1.5.3. We expect the algorithms for other objectives to work similarly to the ones presented for reachability in this paper because for MDPs the other objectives are also similar to reachability. For example, bounded VI in MDPs with total expected reward objective [BKL+17] differs mainly in the fact that one first has to compute an over-approximation since it cannot start at 1 as the highest possible probability. SI for MDPs with mean-payoff [KM17] uses the same underlying idea as for reachability, but keeps track of two metrics (gain and bias) in order to decide how to improve a strategy. A key problem when extending bounded VI to total expected reward in SGs is that there are different semantics of the objective [CFK+13a] and that in some cases the value is not the least, but the greatest fixpoint of the Bellman equations. Thus, VI from below can be stuck. To address this, probably a dual of deflating can be derived, which then increases the lower approximants so that they converge to the value. In the direction of parity objectives, the relationship between simple ECs and tangles [vD18] should be investigated. Tangles are a concept used for solving non-stochastic games with parity objectives; a tangle is a strongly connected subgraph of the game where one player can surely win. This is similar to simple ECs because there, Minimizer cannot prevent Maximizer from choosing the best exit, so in a sense, Minimizer cannot prevent Maximizer from winning the simple EC.

Finally, the problem of deriving a **polynomial algorithm for simple SGs** is of great interest, as discussed in Section 1.3. Since QPs as a solution for simple SGs have not been thoroughly investigated yet, this might be a promising direction. Using the normal form of [Con93], we can construct a polynomially sized QP. A more practical polynomial QP might be possible by finding a solution for MECs that does not enumerate an exponential number of strategies. The key difficulty for showing that the algorithm takes polynomial time is understanding whether the QP is convex or altering it so that it becomes convex. Overcoming this difficulty would give a positive answer to the long-standing open question of whether simple stochastic games can be solved in polynomial time.

4 Expressive system formalisms: Limited information

Motivation In Chapter 3, we have seen how to solve simple stochastic games. However, the algorithms require precise knowledge of all transition probabilities. This assumption might make our results unreliable because in practice we often only have limited information about the system, as discussed in Section 1.5.2.

In the following, we present two ways of modelling limited information by making the formalisms for the system more expressive. The first approach assumes that we can simulate our system and then applies statistical model checking. Thus, we get statistical guarantees on the behaviour of the model and its value. We mention that this approach also addresses the challenge of scalability (see Section 1.2): if a system is too large or complex to be analysed completely, statistical model checking can be used to get an estimate of the value of the game [YS02].

The second approach does not use simulations. Instead, it assumes that for every transition, we know an interval that the probability is contained in. Then the analysis can provide information on the best and worst possible instantiation of the probabilities. Thus, the analysis is robust with respect to the given imprecisions of the transition probabilities.

Problem statement The problem is quite similar to the one in the previous chapter, i.e. given a stochastic game (SG) G, an initial state s_0 and a set of goal states F, we want to compute an ε-approximation of the value and the corresponding ε-optimal strategies.

However, this time we cannot rely on knowing the exact transition function δ. We suggest two approaches.

- In Section 4.1, we assume that we can run and observe simulations of the system. Additionally, in the so-called *black-box* setting, we require a lower bound on all transition probabilities $p_{\min} \leq \min_{\substack{s \in \mathsf{S}, a \in \mathsf{Av}(s) \\ t \in \mathsf{Post}(s,a)}} \delta(s,a,t)$. Alternatively, in the *grey-box* setting, we know the topology of the SG.

  We want to compute an estimate of $\mathsf{V}_G(s_0)$ with a probably approximately correct (PAC) guarantee, a guarantee that with a high probability the estimate is ε-close. Note that in this setting, we only care about the estimate for the value of the SG in the initial state, not the whole value function.

- In Section 4.2, we assume that for every transition we have a probability interval, i.e. we have two transition functions $\underline{\delta}$ and $\overline{\delta}$ such that for all $s, s' \in \mathsf{S}, a \in \mathsf{Av}(s)$ we have $\underline{\delta}(s,a,s') \leq \delta(s,a,s') \leq \overline{\delta}(s,a,s')$. We want to compute a best- and worst-case estimate of the value function.

Chapter outline Section 4.1 summarizes the papers [AKW19a] and [ADKW20] (Appendix C and D), describing methods based on statistical model checking for dealing with unknown transition probabilities. Section 4.2 describes how to deal with bounded-parameter SGs, i.e. SGs where we have intervals on the transition probabilities, as detailed in [WMK19] (Appendix E). Inside the sections, we always summarize the state of the art before our work, then describe our contribution and lastly set it into the bigger context of the related work. Finally, in Section 4.3 we summarize and give an outlook.

4.1 Statistical model checking

4.1.1 State of the art

In this section, we consider algorithms that have limited information about the SG. We first define formally what we mean by that in this work. Afterwards, we give an intuitive description and then shortly discuss how our definition relates to the state of the art.

Definition 6 (Black-box and grey-box)**.** *An algorithm can simulate an SG G if*

- *it knows the initial state s_0,*
- *for a given state, an oracle returns its player and available actions,*
- *it can begin a simulation in s_0,*
- *if a simulation has reached a state s, the algorithm can sample a successor $t \in \mathsf{Post}(s,a)$ according to $\delta(s,a)$ for a chosen action $a \in \mathsf{Av}(s)$. The action is either chosen by the algorithm or by an opponent that satisfies some basic fairness assumption (discussed below). The simulation can continue in state t.*

An algorithm works in the black-box *setting if it can simulate the input SG G and additionally knows $p_{min} \leq \min_{\substack{s \in \mathsf{S}, a \in \mathsf{Av}(s) \\ t \in \mathsf{Post}(s,a)}} \delta(s,a,t)$, an under-approximation of the minimum transition probability.*

An algorithm works in the grey-box *setting, if it can simulate the input SG G and additionally knows the number $|\mathsf{Post}(s,a)|$ of successors for a given state s and action a.*

Intuitively, we can consider three settings: white-, grey- and black-box. In white-box (the setting of the previous Chapter 3), we can access the whole tuple of the given SG G, in particular also the transition function δ. In grey- and black-box, we cannot access the tuple nor the exact transition probabilities, but instead, have to rely on simulations of the system. While in grey-box we still know information about the local topology around the state we simulate (its successors under all actions), in black-box we know nothing about the system except for a global lower bound on the transition probabilities.

Discussion of Definition 6 Our **definition of simulation** conforms to black-box systems in the sense of [SVA04] with sampling from the initial state. It is slightly stricter than [YS02] or [RP09], where simulations can be run from any desired state. In these works, the knowledge of $p_{\min}$ was not required; this was only introduced in [BCC$^+$14, DHKP17] in order to deal with infinite-horizon properties. However, it is a relatively light assumption in many realistic scenarios. For instance, "bounds on the rates for reaction kinetics in chemical reaction systems are typically known; for models in the PRISM language [KNP11], the bounds can be easily inferred without constructing the respective state-space" [DHKP17, Pages 2-3].

The idea of **knowing the topology** of the system was used in [HJB$^+$10] and [YCZ10, Section 3.1]. These works actually assumed to know the topology of the whole SG, i.e. the whole tuple except for the precise transition probabilities. We refrain from that because our motivation is to use as little information as possible about the given SG. Still, the settings are similar, as given enough simulations in our grey-box setting, we can almost-surely infer the whole topology. In [BGJS20], the algorithm knows the exact set of successors $\mathsf{Post}(s, a)$, which is slightly stronger than our assumption of only knowing the number of successors $|\mathsf{Post}(s, a)|$. Still, as before their algorithm could be adapted to our setting, since the exact set of successors can almost-surely be inferred in our grey-box setting.

We believe that the grey-box setting has slightly **stronger assumptions** than the black-box setting, as grey-box has topological information about every state, while black-box only has a global lower bound. However, one can also argue that the grey-box setting is more realistic because of "systems for which probabilities are governed by uncertain forces (e.g. error rates): in this case, it is not easy to have a lower bound on the minimal transition probability, but we can assume that the set of transitions is known" [BGJS20, Page 20]. We provide algorithms for both the grey- and black-box settings, allowing users of our work to choose which setting is more useful in their opinion. According to our practical experience, grey-box should be preferred, if possible.

Finally, we discuss how the **actions are chosen** during the simulations. To give guarantees, intuitively we need to avoid a player hiding their best actions. Formally, in order to be able to infer the value under optimal strategies, we require that with positive probability some optimal action is played in every state. We suggest three approaches.

- If during the simulations of the system we can control both players, we can ensure that we pick optimal actions for both (as is done for one player in [BCC$^+$14]).
- Alternatively, we can assume that a player plays fairly, i.e. every action has a positive chance to be picked in the future (as in [AKW19a]). This could be realized by sampling actions uniformly, but also by skewing towards more promising actions.
- Finally, we can assume that both players eventually converge towards their optimal strategy and only play optimal actions (as in [LN81]).

Probably approximately correct guarantee (PAC guarantee) In this section, we are interested in algorithms that provide a *probably approximately correct guarantee (PAC guarantee)* [Val84]. An algorithm gives a PAC guarantee if for any $\varepsilon, \varDelta \in (0, 1)$, it returns an ε-approximation of the true value with high probability $1 - \varDelta$. In other words: *probably* (unless we are unlucky, which only has a chance of $\varDelta$) the result of the algorithm is *approximately* correct (it is off by an error of at most ε).

Statistical model checking (SMC) Our algorithm is an instance of *statistical model checking (SMC)*, see e.g. [YS02]. We give an overview of the developments of SMC in Section 4.1.5 and an algorithmic description in [ADKW20]. The latter work also discusses the similarities and differences between SMC and the value iteration algorithms from the previous Chapter 3, i.e. classical, bounded or simulation-based value iteration. In the following, we recall the basic idea of SMC as far as it is relevant for our algorithm.

In essence, SMC for Markov chains works by running simulations through the model, which are stopped when either a goal state is reached or we detect that we are stuck in a sink. We only remember whether a simulation reached the goal or not. Then, we can estimate the value as the average of successful simulations over all simulations. To obtain a probabilistic guarantee, we do not report the average, but rather a confidence interval around it; the width of the interval depends on the required confidence $1 - \varDelta$ and the number of simulations. Hence, by increasing the number of simulations we can ensure that the result is approximately correct, i.e. ε-close to the true value.

There are two key complications when applying SMC to simple stochastic games.

- We need a way to resolve the **nondeterminism**. Instead of only storing whether a simulation from the initial state reached the goal or not, the idea of [SLW$^+$06, BCC$^+$14] is to keep an accumulator for every state-action pair, which stores statistics about the performance of simulations using this state-action pair. Then, during simulations, the nondeterminism is resolved based on the estimates in these accumulators. Intuitively, after enough simulations, we infer the optimal strategies. Hence, the model reduces to the Markov chain under optimal strategies and we can estimate the value as the average number of simulations that reach the goal. However, the number of simulations we need in order to infer anything about the actions is beyond infeasible. Its order of magnitude is at least 10^8 seconds even for the smallest case studies in the experimental evaluation of [AKW19a, Section 4], and for medium case studies, it is larger than $10^{100,000}$ seconds. For comparison: the age of the universe is estimated to be around 10^{11} seconds. We address this in Section 4.1.2 by making the algorithm more efficient in several ways.

- SMC crucially relies on the ability to detect that a **simulation is stuck** in a cycle and cannot reach the target anymore. This is difficult even for Markov chains, as we discuss in Section 4.1.5. Only recently a fast solution in the black-box setting was proposed [DHKP17], which relies on the knowledge of the minimum transition probability $p_{\min}$. The idea is to terminate a simulation when it is very unlikely that a transition leaving the current cycle has been overlooked. Our algorithm extends this approach, see Section 4.1.2.2 for more details.

  Note that, when dealing with Markov decision processes (MDPs) or SGs, the problem of terminating a simulation becomes even harder, since we have to deal with end components (ECs). In these, a simulation can be stuck, even though under optimal strategies we can still reach the goal. This is because staying might falsely look preferable to Maximizer, as we described in Example 2. Hence, most previous approaches consider finite-horizon properties (see Section 4.1.5), thus effectively eliminating ECs from consideration because after finite time the game almost-surely reaches a sink state. However, our objective is infinite-horizon reachability.

Thus, we have to deal with ECs. The basis of our algorithm is [BCC+14], which *collapses* ECs. This technique replaces an EC with a single state and hence eliminates the cyclic dependencies. For more information about collapsing, see the last paragraph of Section 3.1.2 or [EKKW22, Section 3]. However, collapsing is not sufficient to deal with SGs, as we discuss in the previous Chapter 3. Hence, we replace it with *deflating*, see Section 4.1.2.3.

4.1.2 Contribution: PAC statistical model checking of SGs

We extend the algorithm from [BCC+14, Section 4.2] in two directions: firstly, we improve the runtime of the algorithm from usually larger than the age of the universe to being able to deal with reasonable systems. We achieve this by switching to a *model-based view* and having a *faster detection of being stuck* in an EC. Secondly, we replace collapsing with deflating and thereby provide an *extension to SGs* for the algorithm that previously only was applicable to MDPs. Overall, we provide an *algorithm with a PAC guarantee* for SGs with limited information.

To the best of our knowledge, this is the first algorithm providing a PAC guarantee for arbitrary SG in the black-box setting. Note that we explain how to modify the algorithm to be applicable in the grey-box setting in Section 4.1.3.

4.1.2.1 Model-based view

The algorithm of [BCC+14, Section 4.2] is model-free. This means it is agnostic of the graph underlying the considered model and only stores an accumulator for the estimate of every state-action pair. While this is quite memory-efficient, it also induces the requirement of running an infeasibly large number of simulations before being able to perform an update with a probabilistic guarantee.

In contrast, our algorithm is model-based, i.e. we construct a partial model of the explored state-space and estimate the transition probabilities. Storing this additional information allows us to extract information from the simulations more easily. To construct the model, we remember all states, actions and transitions we have encountered during simulations. Moreover, since we do not know the transition probabilities, we have to infer them from our sampled data. For this, we count the occurrences of state-action pairs and transitions. When in some state s we play action $a \in \mathsf{Av}(s)$ and reach state t, we increase our occurrence counters $\#(s, a)$ for the state-action pair and $\#(s, a, t)$ for the transition. With this, we can estimate the transition probability as the average number of times t was sampled as successor when playing (s, a), i.e. ${}^{\#(s,a,t)}/_{\#(s,a)}$. To obtain a probabilistic guarantee, we do not just take this average, but additionally, subtract a confidence width c which depends on the required correctness probability $\varDelta$ of the PAC guarantee. For the details of how to obtain c using the Hoeffding bound [Hoe63], see [AKW19a, Section 3.2]. Note that c can be greater than 1, so we have to limit our estimate to not be less than 0. Overall, we **estimate the transition function** δ by using

$$\tilde{\delta}(s, a, t) := \max(0, {}^{\#(s,a,t)}/_{\#(s,a)} - c).$$

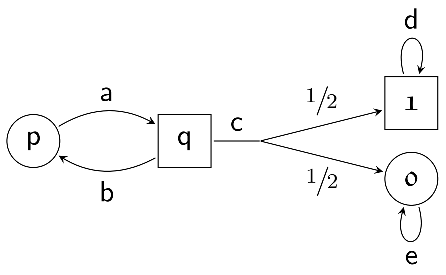


Figure 4.1: Our running example of an SG for Section 4.1.

Example 8 (Estimated transition function $\tilde{\delta}$). *We show our running example for Section 4.1 in Figure 4.1. It is a modification of the SG in Figure 2.1 because we removed the self-loop on* (q, c).

Assume we have seen 5 samples of action c*, where 1 of them went to* $\mathsf{1}$ *and 4 of them to* $\mathsf{0}$*. Note that, in a sense, we were unlucky, as going to* $\mathsf{1}$ *seems quite unlikely based on our observations, while actually, the probability is equal to that of going to* $\mathsf{0}$*. The confidence width for* $\Delta = 0.1$ *and 5 samples is* 0.48 *(see [AKW19a, Example 1] for the calculation). Hence, given these observations, we estimate* $\tilde{\delta}(\mathsf{q}, \mathsf{c}, \mathsf{1}) = {}^{\#(\mathsf{q},\mathsf{c},\mathsf{1})}/_{\#(\mathsf{q},\mathsf{c})} - 0.48 = {}^{1}/_{5} - 0.48 = 0$ *and* $\tilde{\delta}(\mathsf{q}, \mathsf{c}, \mathsf{0}) = {}^{4}/_{5} - 0.48 = 0.32$*. Note that both estimates are in fact lower than the true transition probabilities.*

When increasing the number of samples for every transition, by the law of large numbers our estimates $\tilde{\delta}$ *will converge to* δ*. Subtracting the confidence width ensures that with high probability* $1 - \Delta$ *we always get a conservative estimate, lower than the true transition probability.* $\triangle$

Using $\tilde{\delta}$, we can give the **modified update equations** for the under- and over-approximations L and U. They are as in the white-box setting, see Equation (3.3), with the difference that we do not use $\mathsf{L}(s, a)$ or $\mathsf{U}(s, a)$ because they depend on the true transition probabilities δ. Instead, we use $\tilde{\mathsf{L}}$ and $\tilde{\mathsf{U}}$, which are defined as follows:

$$\tilde{\mathsf{L}}(s, a) := \sum_{t:\#(s,a,t)>0} \tilde{\delta}(s, a, t) \cdot \mathsf{L}(t) \tag{4.1}$$

$$\tilde{\mathsf{U}}(s, a) := \left(\sum_{t:\#(s,a,t)>0} \tilde{\delta}(s, a, t) \cdot \mathsf{U}(t) \right) + \left(1 - \sum_{t:\#(s,a,t)>0} \tilde{\delta}(s, a, t) \right) \tag{4.2}$$

Compared to Equation (3.4), we have replaced δ with $\tilde{\delta}$. However, the sum of all our estimates for the transition probabilities does not add up to 1, as we always subtract the confidence width to get a safe under-estimation of the probabilities. For the remaining probability, i.e. $\left(1 - \sum_{t:\#(s,a,t)>0} \tilde{\delta}(s, a, t)\right)$, we assume that it goes to a sink when calculating $\tilde{\mathsf{L}}$ (effectively making the term 0) and that it goes to a goal when calculating $\tilde{\mathsf{U}}$ (multiplying the term by 1). Thus, after the updates our under- and over-approximations L and U are still correct [AKW19a, Lemma 1].

Example 9 (Modified update equations). *Consider again our running example from Figure 4.1 with the 5 samples of action* c *as described in Example 8. Assume that we already* *found out that* o *is a sink with value 0; how we infer this is explained in the next Section 4.1.2.2.*

Then, already with these 5 samples, we know $\widetilde{\mathsf{U}}(\mathsf{q},\mathsf{c}) = (0 \cdot 1 + 0.32 \cdot 0) + (0.68) = 0.68$. *Intuitively, we are certain (unless we are unlucky, which is unlikely at probability* $\varDelta = 0.1$*) that 0.32 probability goes to the sink. Still, concerning the lower bound, we are not yet sure enough about what reaches the goal to give any estimate* $\widetilde{\mathsf{L}}(\mathsf{q},\mathsf{c}) > 0$. *Given 500 samples of action* c*, the confidence width becomes smaller than 0.05. In that case, since we have this confidence width for both the upper and the lower bound, we can decrease the total precision for this action to* 0.1*, i.e. return an interval in the order of* $[0.45, 0.55]$.

In contrast, with the same error tolerance, $\varDelta = 0.1$ *the approach of [BCC+14] has to sample this transition more than* 10^9 *times before it can attempt a single update.* △

In summary, by switching to a model-based approach, we can estimate the transition probabilities reasonably well already after very few simulations and get a reasonable level of confidence with a realistic[1] number of samples. We can use these estimates to update our under- and over-approximation. This drastic decrease in runtime when compared to the model-free approach comes at the price of increasing the memory consumption. However, while saving triples of state, action and successor instead of only state-action pairs asymptotically increases the memory from $\mathcal{O}(\mathsf{S} \times \mathsf{A})$ to $\mathcal{O}(\mathsf{S}^2 \times \mathsf{A})$, practically states usually have only a small, bounded number of successors. Thus, the increase in memory is not significant, while the improvement in runtime is.

4.1.2.2 Faster detection of being stuck

We cannot let our simulations run infinitely long. There are two natural points for stopping the simulation: (i) when we reach a goal state, since then the value certainly is 1; or (ii) when we reach a non-goal bottom strongly connected component (BSCC), i.e. a cycle in the game graph that has no chance of reaching the goal. There, the value certainly is 0.

Detecting that we have reached a goal is easy, but being sufficiently certain about being stuck in a BSCC is hard. In the black-box setting, we have to sample long enough to exclude the possibility that some action has a non-zero probability of exiting the BSCC.

Example 10 (Difficulties of the stuck-detection)**.** *Consider the first simulation that reaches the state* o *in our running example from Figure 4.1. We see that there is only one action* e*, but we do not know how many successors it has. Imagine that with a very small probability,* e *can transition to the goal state. In that case, concluding that we cannot reach the goal and setting the value to 0 would be completely wrong because even the small probability would make us reach the goal almost-surely.*

No matter how often we sample e*, we can never get* $\widetilde{\delta}(\mathsf{o},\mathsf{e},\mathsf{o}) = 1$*, since the confidence width will always be positive. Thus, we need a different detection mechanism.* △

To detect that we are stuck in a BSCC, we utilize our knowledge of the minimum transition probability $p_{\min}$. If we have sampled some state-action pair (s, a) for n times, the probability to overlook a transition is $(1 - p_{\min})^n$. Hence, we require that $(1 - p_{\min})^n < \varDelta$

[1]Depending on $\varDelta$ and the size of the SG, the number might still be infeasible, but it is certainly more realistic than that of previous approaches.

because then we are sure enough that we did not overlook any transition. Solving this inequation for n yields $n > \frac{\ln(\Delta)}{\ln(1-p_{\min})}$.

Example 11 (Stuck-detection using $p_{\min}$)**.** *Consider our running example from Figure 4.1, an error tolerance $\Delta = 0.1$ and assume we know that $p_{min} > 1/3$. Note that p_{min} is an under-estimate for the actual minimum occurring transition probability of $1/2$.*

Then, if we sample action e *for $n > \frac{\ln(0.1)}{\ln(1-1/3)} = 5.6$ times, we can conclude that the state* o *indeed is a BSCC and set its value to 0. If there was a transition to the goal, we would have seen it with a probability greater than* 0.1*, so the chance of an error is lower than our error tolerance.* △

So far, we have only considered the very simple case of a one-state BSCC. However, BSCCs can be arbitrarily large. Moreover, depending on our guidance heuristics for selecting actions, it is possible that we are not stuck in a BSCC, but in an EC. This would be the case if the EC was not bottom, but the player with the exit believes that staying in the EC is better than exiting, which can easily occur if the Maximizer strategy depends on the over-approximation U (similar to what we described in Example 2). To deal with these issues, we check whether according to our current model, we could be in an EC. If yes, then we additionally check that we have seen all staying actions of the EC often enough to not overlook any transition. Only then do we conclude that we are stuck in this EC. For a more detailed discussion, see [AKW19a, Section 3.3].

The idea of considering the partial model to find ECs is an extension of [DHKP17], where they find candidate BSCCs in the partial model induced by the current simulation. Further, note that we discuss variants of our detection mechanism in Section 4.1.3.

4.1.2.3 Extension to stochastic games

The extension to SGs is quite straightforward: we replace the collapsing from [BCC$^+$14] with deflating, as we did in Chapter 3. Additionally, since we are in the setting with limited information, we have to ensure that whatever we deflate is an EC with the required confidence, using the detection mechanism we just discussed. If we did not do this, we could overlook an exit and hence deflate our over-approximation to be smaller than the value, as in Example 10. Finally, since we do not know the exact transition probabilities, our computation of the over-approximation of the best exit depends on $\tilde{\mathsf{U}}$, and our guess of the optimal Minimizer strategy to compute simple ECs depends on $\tilde{\mathsf{L}}$. For a more detailed discussion, see [AKW19a, Section 3.4].

4.1.2.4 Obtaining the PAC Guarantee

We have now described all the ingredients of our algorithm. However, combining them and obtaining a PAC guarantee still is surprisingly difficult. In particular, we need to take further steps to ensure that our error probability is less than Δ. Firstly, note that for every transition probability, we allow an error probability of Δ. Thus, the bound on the overall error probability is Δ times the number of transitions, which is too large. We resolve this problem by distributing Δ over all transitions, e.g. by uniformly dividing it among them.

Still, with every simulation we estimate the transition probabilities anew, every time possibly making a mistake. These **error probabilities aggregate**.[2] We resolve this as follows: the algorithm is divided into a simulation-phase, where we gather knowledge and can use any estimates and guidance heuristics we want. After a certain number of simulations, we start the second phase, which is called guaranteed bounded value iteration phase (guaranteed BVI-phase). There, we construct a partial model, estimating all the transition probabilities once. Then we perform BVI for a certain number of steps. If the estimates for the transition probabilities were good enough, we converge and obtain an ε-precise result. Otherwise, we can use the bounds obtained in the BVI-phase to guide the next simulation-phase.

By splitting the algorithm into these two phases, we can use the simulations to gather information, but do not rely on the bounds it calculates to be correct. The guaranteed BVI-phase uses all the data gathered during the simulations for one informed guess of the partial model and hence can give a probabilistic guarantee. For the pseudocode of the algorithm and a more detailed discussion, see [AKW19a, Section 3.5]. Overall, we obtain the following theorem ([AKW19a, Theorem 2]).

Theorem 3. *With a sufficient number of simulations per simulation-phase, Algorithm 7 of [AKW19a] is PAC for any input parameters G, F, ε and Δ. That means it terminates almost-surely and returns an interval for $\mathsf{V}_G(s_0)$ of width smaller than ε that is correct with a probability of at least $1-\delta$.*

However, the number of simulations that we require for this guarantee can still be infeasibly high. Thus, we also provide another theorem ([AKW19a, Theorem 1]) which is the key to the practical applicability of our algorithm.

Theorem 4. *For any number of simulations per simulation-phase, Algorithm 7 of [AKW19a] is an anytime algorithm with the following property: When it is stopped, it returns an interval for $\mathsf{V}_G(s_0)$ that is correct with a probability of at least $1-\delta$.*

The theorem allows us to run the algorithm with any number of simulations that we have time to run and then know that the returned result is probably correct. While this result can be useless (e.g. the interval [0,1]), it can also be good enough to get an approximation of the value in a case where otherwise we would not be able to know anything.

4.1.3 Contribution: Detecting stuck simulations in different settings

In Section 4.1.2.2, we considered the black-box setting, i.e. we used $p_{\min}$ to detect that we are stuck in an EC by estimating the probability to overlook a successor. If we have some knowledge of the topology, i.e. we are in the grey-box or white-box setting, there are more efficient ways of detecting ECs. Given a set of states $T \subseteq \mathsf{S}$ we can verify that it is an EC by checking that for all states $s \in T$ and actions $a \in \mathsf{Av}(s)$ played by the current strategies, we have $|\mathsf{Post}(s,a)| = |\widehat{\mathsf{Post}}(s,a)|$, where $\widehat{\mathsf{Post}}(s,a)$ is the number of successors we observed while simulating the state-action pair (s,a). Note that for this we only need the number of successors $|\mathsf{Post}(s,a)|$, not the exact set of successors $\mathsf{Post}(s,a)$ nor the

[2] For an idea of how not to deal with aggregating errors, see [Mun19].

transition probabilities. Assuming we know $\mathsf{Post}(s,a)$ (as in other variants of the grey-box setting), our detection can be faster, since we do not have to simulate until seeing all successors of every action. Knowing the precise transition probabilities (white-box setting) does not make the stuck-detection easier.

Further, if we know that $|\mathsf{Post}(s,a)| = |\hat{\mathsf{Post}}(s,a)|$ for some state-action pair (s,a), we can be less conservative in the updating Equations (4.1) and (4.2): instead of assuming all the remaining probability goes to a sink or goal, we know that it has to reach one of the successors. Hence we can use their lower and upper bounds.

Our detection mechanism is not only applicable in our PAC algorithm, but we can also use it in classical SMC. In particular, if we consider a Markov chain in grey-or white-box setting, we do not have to assume knowledge of the second eigenvalue [LP08, YCZ10] or transform the whole state-space [HJB$^+$10, YCZ10] (see Section 4.1.5 for more information on these related works). Instead, we can rely on our simple containment check.

We refer to [ADKW20, Sections 4 and 5] for an in-depth discussion of different detection mechanisms, possible practical improvements and the impact of model structure on variants of SMC in different settings.

4.1.4 Experimental results

We first report on the results of the algorithm for statistical model checking of SGs from [AKW19a] (Section 4.1.2). Afterwards, we recall the key messages from the experimental comparison in [ADKW20].

4.1.4.1 Statistical model checking of stochastic games

We ran Algorithm 7 of [AKW19a] on MDPs from the quantitative verification benchmark set [HKP$^+$19] and on SGs from the PRISM-games case study website.[3] We did not choose the number of simulations necessary to obtain the PAC guarantee. This is because this number probably is very high and since we only proved its existence, not a concrete bound. Instead, we used the property of Theorem 4, namely that for any number of simulations, we get a probabilistic guarantee. We ran the algorithm both in the black-box and the grey-box setting in order to estimate the impact of the additional information.

Figure 4.2 shows the evolution of the lower and upper bounds in both the grey- and the black-box settings for 4 different models. This is a representative selection in the following sense: we see that the bounds converge from above and below, independent of whether the value is close to 1 or 0.5; that in the grey-box setting the bounds always converge more quickly than in the black-box setting; and that sometimes black-box does not converge too precisely (bottom left) or at all (bottom right). Graphs for the other models are in [AKW19b, Appendix C] and a table of the results in [AKW19a, Section 4].

In the black-box setting, within 30 minutes we obtained a precision of $\varepsilon < 0.1$ on 6 of 16 benchmarks. This is in stark contrast to the state of the art, which cannot finish a single update in our lifetime. In the grey-box setting, the results were even better: within 6 minutes, we achieved a precision of $\varepsilon < 0.1$ for 14 of 16 benchmarks; within the 30 minutes, we achieved a precision of $\varepsilon < 0.01$ for 15 of 16 benchmarks.

[3] `http://www.prismmodelchecker.org/games/casestudies.php`

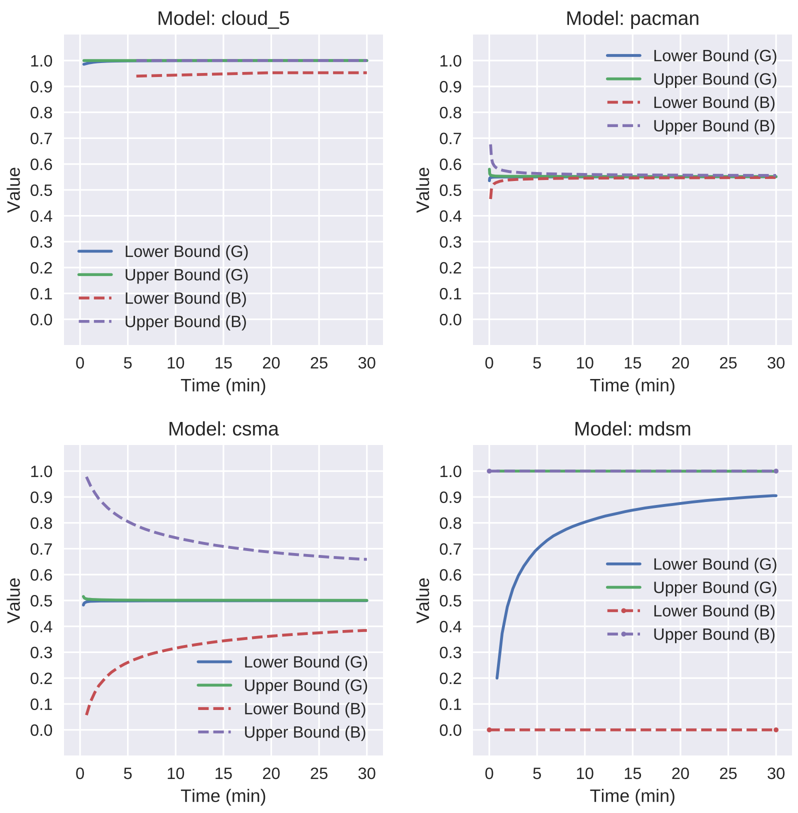


Figure 4.2: Performance of our algorithm on various MDP and SG benchmarks in grey-and black-box settings. Solid lines denote the bounds in the grey-box setting while dashed lines denote the bounds in the black-box setting. The plotted bounds are obtained after each BVI-phase, hence they do not start at the interval $[0, 1]$ nor the time 0. Taken from [AKW19a, Figure 2].

In summary, we see that our algorithm is significantly better than the state of the art in the black-box setting, making it possible to sometimes get estimates of the value. In the grey-box setting, it actually is practically useful to obtain a good estimate of the value within a reasonable time.

4.1.4.2 Further comparison of black-box and grey-box statistical model checking

The goal of [ADKW20] was to compare SMC in the black-and grey-box setting and to compare it with an algorithm that gives non-probabilistic guarantees, namely bounded value iteration (BVI, see Section 3.1.2). Note that the paper only considered Markov chains from the PRISM benchmark suite [KNP12], not SGs.

The main results are displayed in Table 4.1. The first columns describe the model, giving its name, size, the minimum occurring transition probability $p_{\min}$ (which affects the stuck-detection in the black-box setting), and the number of BSCCs as well as their maximum size. The last three columns show the runtime of SMC in the black-and grey-box

Table 4.1: Runtime (in seconds) comparison of black- and grey-SMC for various benchmarks. Runtimes of bounded value iteration (BVI) are also presented as a baseline. Based on [ADKW20, Table 1].

Model/	Size	$p_{\min}$	BSCC	SMC		BVI
Property			(No., Max. Size)	black-	grey-	
`bluetooth(10)` `time_qual`	$> 569K$	7.81×10^{-3}	$> 5.8K, 1$	9	**7**	TO
`brp_nodl(10K,10K)` `p1_qual`	$> 40M$	1×10^{-2}	$> 4.5K, 1$	86	**84**	TO
`crowds_nodl(8,20)` `positive_qual`	68M	5×10^{-2}	$> 3K, 1$	10	**8**	TO
`egl(20,20)` `unfairA_qual`	1719T	5×10^{-1}	$1, 1$	43	**25**	TO
`gridworld(400,0.999)` `prop_qual`	384M	1×10^{-3}	$796, 160K$	15	**8**	TO
`herman-17` `4tokens`	10G	4.7×10^{-7}	$1, 34$	TO	**73**	98
`leader6_11` `elected_qual`	$> 280K$	5.6×10^{-7}	$1, 1$	**106**	152	OOM
`nand(50,3)` `reliable_qual`	11M	2×10^{-2}	$51, 1$	11	**10**	455
`tandem(2K)` `reach_qual`	$> 1.7M$	2.4×10^{-5}	$1, > 501K$	**7**	**7**	62

setting, called black-SMC and grey-SMC, as well as the runtime of BVI. In summary, the experiments reveal the following.

- For most benchmarks, black-SMC and grey-SMC perform similarly. This is because most of the models only contain trivial BSCCs of size 1, in which both black-and grey-SMC can quickly conclude that they are stuck.

- The advantage of grey-SMC is visible on benchmarks with non-trivial BSCCs, which is only herman-17 in Table 4.1. Here, black-SMC quickly fails while grey-SMC is extremely competitive. The table exemplifies this for N= 17, and [ADKW20, Section 6.2] provides a more detailed analysis for multiple N.

- Another advantage of grey-SMC is that it is independent of $p_{\min}$, while black-SMC is extremely sensitive to it. This is analysed in more detail in [ADKW20, Section 6.1].

- Classical techniques such as VI and BVI fail when either the model is too large or the transition probabilities are too small. However, they are still to be used for strongly connected systems, where the whole state-space needs to be analysed for every run in both SMC approaches, but only once for (B)VI.

4.1.5 Related work

We give an overview of the developments of SMC in the past two decades. We focus on setting our contribution apart from similar works but also mention relevant related directions of research. For further extensive surveys on the topic, see [LDB10, Kre16, AP18, LLT$^+$19]. Note that there is a special track dedicated to SMC at the conference Isola, see [LL14] for the first and [LL20] for the most recent edition (at the time of writing).

For a comparison of statistical and numerical solution methods (like the ones in Chapter 3), we refer the interested reader to either [You06], which includes an empirical comparison and presents a framework for expressing correctness guarantees; or to our paper [ADKW20], which gives a detailed exposition of the algorithmic similarities and differences of value iteration (classical, bounded or simulation-based) and statistical model checking, as well as a discussion of their applicability.

Historically,[4] statistical model checking started with the use of simulation and statistics based methods for solving **Markov chains with time- or step-bounded properties**, as in [YS02, SVA04, You05, YKNP06, JCL$^+$09, JLS12, BDL$^+$12]. Applications of this range over numerous domains, e.g. biological [JCL$^+$09, PGL$^+$13], hybrid [DDL$^+$12, EGF12, Lar12, ZPC13] or cyber-physical [CZ11, BBB$^+$12, DDL$^+$13, KJL$^+$16] systems. There are numerous tools for solving this problem [SVA05b, BDL$^+$12, BHH12, JLS12, BCLS13], where [BHH12] even already handles a modest form of nondeterminism.

The problem we have described here in Section 4.1 differs from Markov chains with bounded-horizon objectives in two ways: firstly, we consider infinite-horizon reachability and secondly our model exhibits nondeterminism. Both of these pose fundamental challenges.

Infinite-horizon properties but no nondeterminism

When model checking infinite-horizon properties (also known as unbounded), we require a way to terminate the simulation after a finite time and still reliably evaluate the infinite-horizon specification based on this finite simulation. We group previous approaches into three categories, according to the knowledge they require.

- White-box: The first approach for model checking infinite-horizon properties was described in [SVA05a], but it depends on knowledge of the value that it wants to compute, as observed in [HJB$^+$10]. Similarly, in order to give guarantees, the approaches proposed in [LP08] and [YCZ10, Section 3.2] must compute the second eigenvalue of the Markov chain, which is as hard as the verification problem itself.

- Grey-box: The approach of [HJB$^+$10] leverages the base idea of [SVA05a], but modifies the key procedure that decides whether a formula holds with positive probability. It constructs a Markov chain with the same topology, but every transition probability distribution is uniform. Using that the specification can be decided by simulations

[4]This paragraph is based on [Kre16].

of bounded length. The approach of [YCZ10, Section 3.1] performs a reachability analysis on the underlying topology of the Markov chain and then transforms it so that all potentially infinite simulations are eliminated.

Note that both of these approaches require processing the whole state-space to construct modified Markov chains, whereas our grey-box approach (see Section 4.1.3) works on a partial model and only uses the local information about the topology to terminate simulations. This is particularly important as this preprocessing of the whole state-space counteracts the purpose of statistical model checking: tackling the state-space explosion problem by avoiding the necessity to have the whole model in memory. This practical disadvantage is also experimentally confirmed in [DHKP17, Section 5].

Recently, [BGJS20] have proposed learning a model of a Markov chain based on the topology and simulations. They derive a PAC guarantee. The underlying theory is quite similar to our approach in the grey-box setting, see Section 4.1.3 and [AKW19a, Section 3.6]. A difference is that they use the Chen bound [Che15] instead of the Hoeffding bound [Hoe63], which "requires fewer samples than the other sequential algorithms" [BGJS20, Page 5]. Moreover, they not only treat reachability, but the whole computational tree logic (CTL), and provide an impossibility result for linear temporal logic (LTL).

- Black-box: The approach in [RP09] does not assume any knowledge of the system and instead only performs simulations. However, its usage is quite limited: it is applicable only to ergodic Markov chains because it relies on the existence of the steady-state distribution (observed in [DHKP17]); moreover, the practical performance of the algorithm heavily relies on the Markov chain being monotone, but it is not discussed how monotonicity of a Markov chain can be determined (observed in [YCZ10]).

  The basis of our approach is given in [BCC$^+$14, DHKP17], requiring not only the power to simulate but also knowledge of the minimum transition probability $p_{\min}$. Using this, we can infer a bound on the probability that a simulation still is in the transient part of the state-space after some number of steps; hence, we can give an upper bound on the length of the simulation, after which the probability to not be in the recurrent part is smaller than our confidence $\varDelta$. While the a-priori step bound from [BCC$^+$14] is infeasible, the faster method of [DHKP17] reduces simulation lengths to be practically useful. As we discuss below Definition 6, knowledge of $p_{\min}$ is a relatively light assumption in many realistic scenarios [DHKP17].

Nondeterminism but finite-horizon properties

Now, after discussing how to deal with infinite-horizon properties, we turn to the second problem: extending the system to include nondeterminism. Firstly, we consider the setting where the properties are finite-horizon (also known as bounded) and hence no sophisticated stopping criterion is necessary. One approach to deal with nondeterminism is to give it a probabilistic semantics, as for timed automata in [DLL$^+$11a, DLL$^+$11b, Lar13]. By doing this, the problem is effectively reduced to a Markov chain.

Alternatively, [HMZ$^+$12] employs reinforcement learning for MDPs with bounded linear temporal logic objectives. However, “at any given point we cannot quantify how close to optimal the candidate scheduler is” [HMZ$^+$12, Page 5] and the algorithm “does not in general converge to the true maximum [...] A further problem is that [the algorithm] will eventually produce erroneous results” [DLST15, Page 3]. Moreover, it only uses memoryless strategies, which are not sufficient for the considered objective.

Algorithms with guarantees were proposed in [Lit94, SLW$^+$06, LP12]. They all deal with discounted reward objectives[5] and employ learning techniques. [Lit94] provides a convergent algorithm for solving concurrent SGs. Both [SLW$^+$06] and [LP12] study MDPs and obtain a PAC guarantee, where [SLW$^+$06] is model-free while [LP12] is model-based.

Further, [EGF15] solves stochastic hybrid systems[6] for properties that can be encoded in stochastic satisfiability modulo theory. It employs bounded model checking to obtain a probabilistic guarantee on the result, and the algorithm could be straightforwardly extended to yield a PAC guarantee.

Furthermore, [HBKS21] studies combinations of a finite-horizon reward and multiple cost objectives in constrained MDPs and provides two model-based approaches, one offline like us and one online. Similarly, [WT21] considers SGs with a combination of sure safety[7] and discounted reward. Dealing with the safety objective in [WT21] is achieved by assuming the grey-box setting and never taking an action that leaves the safe region. For the reward objectives, both the online approach of [HBKS21] and [WT21] require a transition to be seen a certain number of times before they can start updating the estimates, similar to [BCC$^+$14]. This required number is a lot smaller than in [BCC$^+$14], but still infeasibly large: for an SG with 100 states, 1 action per state and a confidence bound Δ and error tolerance ε of 0.1, both approaches require more than 10^9 samples before the first update. Increasing the state space size to 10^6, the first update requires more than 10^{17} samples. The online approach of [HBKS21] only starts giving the PAC guarantee after performing a number of simulations that is slightly smaller than for the offline approach, but still too large. In contrast, our grey-box method can immediately give guarantees and return estimates within minutes, see Example 9 and Section 4.1.4.1.

Nondeterminism and infinite-horizon properties

Finally, we look at both complications, i.e. systems with nondeterminism and infinite-horizon properties. Most of the papers cited in this section apply some form of learning, e.g. reinforcement or delayed Q-learning. However, one line of research used a different approach: The works [LST14, DLST15, DHS18] sample uniformly from the space of strategies; this relies on a a compact representation of strategies as the seeds of a pseudo-random number generators. The considered strategies can even be history-dependent since memoryless strategies are insufficient for several of the system-specification combinations in the

[5]Note that solving a discounted objective basically is equivalent to terminating the simulation with the discount factor in every step, and hence when approximating such an objective can be considered bounded.

[6]These are MDPs with some continuous state-variables.

[7]The paper considers almost-sure safety. This is equivalent to sure safety because if there was any path leaving the safe region, it would do so after a finite time and thus have a non-zero probability. Hence, both in sure and almost-sure safety, no path may leave the winning region.

papers. Note that the first presentations of the approach only dealt with bounded linear temporal logic [LST14, DLST15], but was extended to transient properties[8] in [DHS18]. This approach gives useful lower bounds on the value when good strategies are frequent because then the chances of sampling a close to optimal scheduler are high.

Now we turn to approaches that deal with the nondeterminism using learning techniques. Already in [LN81], a learning algorithm was given for zero-sum SGs, which converges to the optimal pure strategies; however, it did not deal with simulations and how to terminate them, but rather assumed that given two strategies, it can get the outcome of the SG.

[BT00] considers SGs with an undiscounted reward objective and provides a PAC learning algorithm. Unlike us, they assume ergodicity of the SG under any pair of strategies. Intuitively, this means that the whole SG is a single BSCC and independent of the strategies every state is reached almost-surely. Thus, they only perform a single simulation and thereby learn the whole SG almost-surely.

In [FT14], the problem of terminating simulations is solved in a similar fashion to [LP08, YCZ10] by requiring (1) the mixing time T of the MDP, which is as hard to compute as the values we are looking for. The algorithm then yields PAC bounds in time polynomial in T (which in turn can of course be exponential in the size of the MDP). Moreover, the algorithm requires (2) the ability to restart simulations also in non-initial states, (3) it only returns the strategy once all states have been visited sufficiently many times, and thus (4) requires the size of the state-space $|S|$. This contradicts a key reason to use SMC, namely to avoid exploring the whole state space.

In contrast, the basis of our algorithm, [BCC$^+$14] does not require the assumptions (1), (2) and (3). In particular, instead of assuming knowledge of the mixing time T, it requires only (a bound on) the minimum transition probability $p_{\min}$ to be able to terminate simulations. However, as mentioned before, e.g. in Section 4.1.4.1, the approach is entirely theoretical as the required step bounds for the simulations are infeasible. Note that since our approach is model-based and uses the smarter techniques from [DHKP17] for terminating simulations, it does not even require assumption (4) anymore and still can terminate simulations in a reasonable time. Also, in contrast to [FT14] our algorithm can terminate early, while [FT14] needs at least T steps, i.e. a number exponential in the size of the MDP.

When not aiming for a PAC guarantee, but only for convergence without any bound on the current error, deep reinforcement learning has been applied to MDP with many objectives, see e.g. [HPS$^+$19, Has20]. These works aim to make formal verification practically applicable by using deep learning techniques and hence "quantification of the sub-optimality of the policy generated by these methods is out of the scope of this work" [Has20, Page 59]. Note that [Has20] is a PhD-thesis summarizing a large body of work, including also algorithms for MDPs with continuous state and action spaces.

As a final note on SMC, we mention that SMC (in particular Monte-Carlo-Tree-Search) can be combined with the simulation-based algorithm from Section 3.1.3, as suggested in [ABKS18]. This yields a spectrum of algorithms from Monte-Carlo-Tree-Search without any guarantees to the full-knowledge algorithm with hard guarantees. The combined algorithms can outperform both basis algorithms as well as traditional (non-simulation-based) value iteration [ABKS18].

[8] Also called constrained reachability: avoid a set of states while trying to reach a goal. By making all avoid states absorbing, it is also equivalent to classical infinite-horizon reachability.

Learning strategies online

Our work in this section assumes that we are in an offline setting, where we can first simulate the system to learn an optimal strategy before having to apply it. A related branch of research — online learning of strategies — considers the setting where we want to optimize performance while we are running the system. The key difference is that in our setting we can explore all states to get a precise estimate of how good or bad they are, while in the other setting we want to avoid bad states. Our paper [EKKW21] finds out when we are stuck in a "bad" cycle and restarts the system, building on the techniques from [DHKP17]. In [KPR18, KMMP20], the approach learns strategies for unknown MDPs online. In [KPR18], they optimize mean-payoff while satisfying an ω-regular objective; in [KMMP20], they provide PAC guarantees for both a black-box and a grey-box setting. Both of these works assume we are inside a single EC. In a more general reinforcement-learning context, shielding [ABE$^+$18] allows satisfying a safety specification while learning an optimal strategy.

4.2 Bounded-parameter stochastic games

In this section, we consider systems where the transition probabilities are not given precisely, but by an interval. In literature, there exist many names for these kinds of systems: Markov chains with this property are called *interval Markov chains* [CSH08, BLW13] or *interval valued Markov chains* [JL91, KU02, SVA06]; and MDPs are called *interval MDPs* [HM18, HHH$^+$19b], *MDP with uncertain transition probabilities or uncertain MDPs* [SJ73, NG05, WTM12] or, the name that we adopt in this thesis, *bounded-parameter MDPs* [GLD00, TB07, WK08, FDdB14]. For our technical discussion, we choose to only use the word *bounded-parameter system*, where systems can be Markov chains, MDPs or SGs. We motivate this choice in Remark 3.

We have adjusted the presentation of our paper [WMK19] as follows: in Section 4.2.2, we extract a general construction for reducing bounded-parameter SGs to normal SGs from [WMK19, Theorem 1 and Remark 3]. The actual focus of [WMK19] on bounded-parameter MDPs with ω-regular objectives serves as a showcase of the generality and the advantages of this construction in Section 4.2.3. We start in Section 4.2.1 by recalling relevant notions from related work, while slightly extending and reformulating them to serve as a good basis for our general construction.

4.2.1 State of the art

We give the definition of a bounded-parameter SG, which is a natural extension of the definition of bounded-parameter MDP in e.g. [GLD00].

Definition 7 (Bounded-parameter stochastic game (bSG)). *A bounded-parameter stochastic game (bSG) is a tuple* $\mathcal{G} := (\mathsf{S}, \mathsf{S}_\square, \mathsf{S}_\bigcirc, \mathsf{A}, \mathsf{Av}, \underline{\delta}, \overline{\delta})$*, where* $\mathsf{S}, \mathsf{S}_\square, \mathsf{S}_\bigcirc, \mathsf{A}$ *and* Av *are as in Definition 1 (definition of SGs), and* $\underline{\delta}, \overline{\delta} : \mathsf{S} \times \mathsf{A} \times \mathsf{S} \to [0,1]$ *are lower and upper bounds on the transition function.*

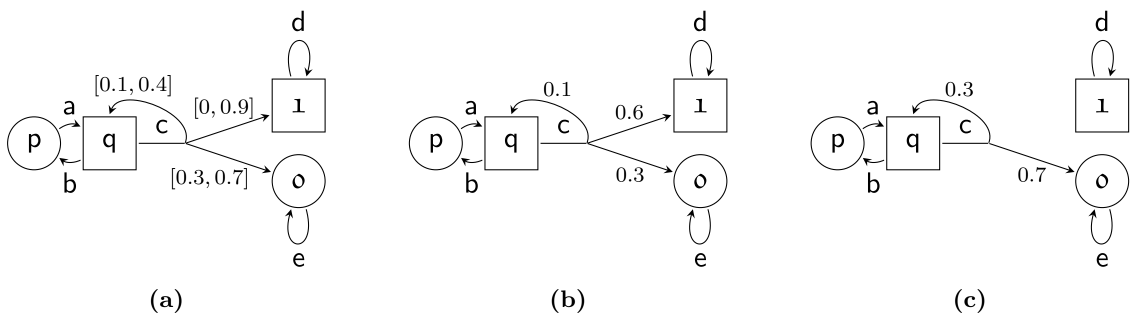


Figure 4.3: (a): Our running example of a bSG for Section 4.2. (b/c): A consistent instantiation under best-/worst-case semantics.

We call an SG $G = (\mathsf{S}, \mathsf{S}_\square, \mathsf{S}_\bigcirc, \mathsf{A}, \mathsf{Av}, \delta)$ *consistent with* a bSG $\mathcal{G} = (\mathsf{S}, \mathsf{S}_\square, \mathsf{S}_\bigcirc, \mathsf{A}, \mathsf{Av}, \underline{\delta}, \overline{\delta})$, denoted $G \in \mathcal{G}$, if and only if the transition function δ of G is in the interval defined by $\underline{\delta}$ and $\overline{\delta}$, i.e. for all states $s, s' \in \mathsf{S}$ and action $a \in \mathsf{Av}(s)$ we have $\underline{\delta}(s, a, s') \leq \delta(s, a, s') \leq \overline{\delta}(s, a, s')$.

Note that $\underline{\delta}$ and $\overline{\delta}$ do not map to probability distributions. Further, note that a necessary requirement for the existence of SGs consistent with bSG is the following: for all states $s, s' \in \mathsf{S}$ and action $a \in \mathsf{Av}(s)$ we have $\underline{\delta}(s, a, s') \leq \overline{\delta}(s, a, s')$; and $\sum_{t \in \mathsf{S}} \underline{\delta}(s, a, t) < 1$ as well as $\sum_{t \in \mathsf{S}} \overline{\delta}(s, a, t) > 1$. We assume that this requirement is satisfied, i.e. there exist consistent SGs for every bSG we consider.

A bSG $\mathcal{G}$ generalizes bounded-parameter MDPs (bMDP) and bounded-parameter Markov chains (bMC) in the usual way, i.e. it is a bMDP if all states belong to one player ($\mathsf{S}_\square = \emptyset$ or $\mathsf{S}_\bigcirc = \emptyset$), and it is a bMC if all states s have only a single action ($|\mathsf{Av}(s)| = 1$).

Example 12 (bSG)**.** *Figure 4.3a shows the running example of a bSG for Section 4.2. As before, it is a modification of the SG from Figure 2.1. Here, we use intervals instead of precise transition probabilities. Note that for actions* a, b, d *and* e, *we do not depict the interval* $[1, 1]$.

For action c, *we do not have the precise probability distribution that assigns* $1/3$ *to every successor as before. Instead, we have intervals that describe possible instantiations. Note that there are uncountably many consistent instantiations because the intervals contain uncountably many real numbers.*

We show two consistent instantiations in Figures 4.3b and 4.3c. They are consistent because the chosen probabilities add up to 1, i.e. they are a proper distribution over successors and because all chosen probabilities lie within the given intervals. We highlight that it is possible for the underlying graph to change when instantiating the probabilities, since transitions can disappear, such as $(\mathsf{q}, \mathsf{c}, \mathtt{1})$ *in Figure 4.3c.* △

There are two important **semantic distinctions** to be made when dealing with bounded-parameter systems. We first give a formal definition of the semantics before providing an example and some intuition. The semantics decide how the probabilities are picked from the intervals. We can view the intervals as an additional source of nondeterminism and hence we need an interval-player who resolves this nondeterminism. This player can use memoryless or history-dependent strategies (once-and-for-all or at-every-step) and be helpful or adversarial (best-case or worst-case).

Definition 8 (Interval-player strategy)**.** *An interval-player strategy π in a bSG $\mathcal{G}$ is a function that for a given finite path and choice of action (by either Maximizer or Minimizer) returns a distribution over successor states that is consistent with the intervals. Formally, $\pi : ((\mathsf{S} \times \mathsf{A})^* \times \mathsf{S}) \times \mathsf{A} \mapsto \mathcal{D}(\mathsf{S})$, such that for all $\rho \in \mathsf{Paths}$, all $s, s' \in \mathsf{S}$ with the last state of ρ being s, and all $a \in \mathsf{Av}(s)$ we have $\underline{\delta}(s,a,s') \leq \pi(\rho,a)(s') \leq \overline{\delta}(s,a,s')$.*

A bSG $\mathcal{G}$ together with a pair of Maximizer and Minimizer strategies (σ,τ) and an interval-player strategy π induces a Markov chain[9] *$\mathcal{G}^{(\sigma,\tau,\pi)}$ and, given a starting state $s \in \mathsf{S}$, a unique probability measure $\mathbb{P}^{(\sigma,\tau,\pi)}_{s,\mathcal{G}}$ on infinite paths.*

Definition 9 (Semantics)**.**

- Once-and-for-all or at-every-step. *In the once-and-for-all semantics, the interval-player strategy is memoryless. This is equivalent to choosing a single consistent SG $G \in \mathcal{G}$. In the at-every-step semantics, the interval-player strategy uses memory.*

- Best-case or worst-case. *In the best-case semantics, the interval-player strategy is the supremum over all strategies, i.e. it is maximizing the objective. In the worst-case semantics, it is the infimum.*

Example 13 (Semantics)**.** *Consider again the bSG from Figure 4.3a and let the objective be to maximize the chance of reaching* `1`*.*

Under best-case semantics, the interval-player will choose the probabilities as in Figure 4.3b, since it minimizes the chance of reaching `0` *and assigns the greatest possible probability to reaching* `1`*. Under worst-case semantics, the interval-player can choose the probabilities as in Figure 4.3c, thereby reducing the chance of reaching the goal to 0. Note that in both cases, picking the probabilities is non-trivial and does not only amount to using extremal values of the intervals because of the interdependencies between the intervals and the necessity to be consistent.*

So far, this example used once-and-for-all semantics, as we fixed the probabilities and selected a single consistent SG. Under at-every-step semantics, the interval-player can choose new probabilities every time action `c` *is played. For a reachability objective, this is not necessary. Thus, to exemplify the necessity of these semantics, we consider a different objective: being in state* `q`*, we want to loop once in* `q` *and afterwards reach* `1`*. This is a linear temporal logic objective, specifying a temporal relation between different events.*

Under best-case semantics, an optimal strategy of the interval-player is to first maximize the chances of looping by picking $\delta(q,c,q) = 0.4$ and some consistent probabilities for both other transitions. Upon reaching `q` *for the second time, the interval-player switches to picking probabilities as in Figure 4.3b. This strategy uses memory and takes advantage of the fact that the interval-player can modify the choice at every step.* $\triangle$

[9] This Markov chain can have infinite state-space since the interval-player strategy can be history-dependent. We refer to [BK08, Section 10.6] for details on how to obtain a Markov chain by applying a strategy to a nondeterministic system.

We give some facts and intuition about the semantics. The best-case semantics can be understood as a "design-challenge", where we are interested in building our system and finding the optimal assignment of all transition probabilities, as in [BLW13]. The worst-case semantics is about robustness in the face of uncertainty. In other words: we want to compute a value that can be obtained, no matter what probabilities are picked from the intervals. This view is taken if our focus is robustness or model-checking, see e.g. [SVA06, CSH08, WTM12, PLSS13, HHH$^+$19b].

Once-and-for-all semantics is "included" in at-every-step semantics because the interval-player can choose not to use memory. Further, if memoryless strategies are sufficient for the interval-player, once-and-for-all and at-every-step semantics yield the same value and optimal strategies. This is the case for the reachability objective in Example 13 and is proven for other objectives in [PLSS13, BDL$^+$17, HM18], as well as Corollary 1 in Section 4.2.3. However, there are cases where memory is necessary to achieve the optimal value, which also affects the complexity to solve the problem, see e.g. [CSH08]. This was exemplified with the linear temporal logic objective in Example 13.

Remark 3 (Naming)**.** *These semantics have appeared under different names in the literature. Once-and-for-all and at-every-step were used in [BDL$^+$17]. Alternatives are stationary uncertainty model and time-varying uncertainty model from [NG05] or uncertain Markov chain and interval Markov decision process from [SVA06]. Even though the last two names were reused in several works [CSH08, CK15, DC18], we refrain from using them, as they can lead to confusion: uncertain is a very broad term that can also refer to more general uncertainties than just intervals [NG05, WTM12] and interval Markov decision process is also used to denote bMDPs in [HM18, HHH$^+$19b].*

Instead of best-case and worst-case, literature typically uses interval-value function [KU02, GLD00, WK08], optimistic or pessimistic criterion [TB07], maximal and minimal probability [BDL$^+$17] or lower and upper bound [DC18]. We do not use the word interval-value function because we want to differentiate whether the interval-player is maximizing or minimizing, and we refrain from using maximal/minimal probability or lower/upper bound as these words have been used throughout Chapter 3 with a different meaning.

The word interval-player is not used in literature, but the idea of having a player resolve the intervals is often used. This player is called nature [NG05, PLSS13, HHH$^+$19b], environment [WTM12] or adversary [LAB15]. These words evoke the feeling that the player is random or adversarial, as the papers usually only consider the worst-case semantics. Hence, since we also want to treat the best-case semantics, we use the more general term interval-player.

Our goal is to reduce model checking of bounded-parameter systems to model checking of normal systems. Thus, we need a way to get rid of the interval-player while still capturing all uncountably many possible interval-player strategies. The following key observation allows us to that: even though there are uncountably many ways of picking consistent probability distributions, they can be expressed as the combination of *finitely many* distribution, which are called *basic feasible solutions* [SVA06, Section 6.1].[10]

[10]Our definition of BFS is based on [SVA06, Definition 4], but slightly adjusts notation and defines BFS for state-action pairs instead of only states because we consider bSGs instead of only bMCs as in [SVA06].

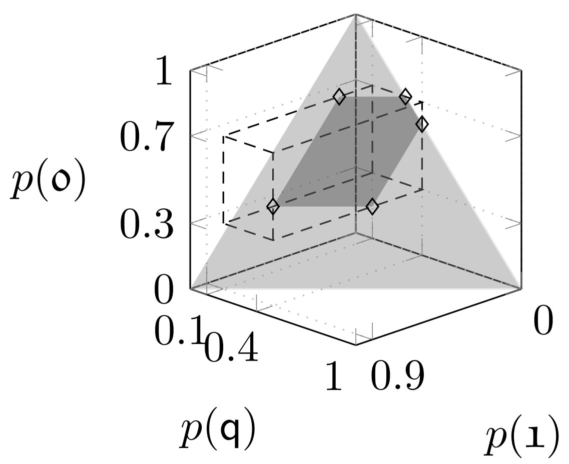


Figure 4.4: Visualization of the BFS of the state-action pair (q, c) in the bSG from Figure 4.3a. Based on [WMK19, Figure 2b].

Definition 10 (Basic feasible solution (BFS))**.** *A distribution $p_{(s,a)} \in \mathcal{D}(\mathsf{S})$ is a basic feasible solution (BFS) for a state-action pair (s, a) if and only if it is* consistent *with the intervals, i.e.*

$$\underline{\delta}(s, a, s') \le p_{(s,a)}(s') \le \overline{\delta}(s, a, s') \text{ for all } s' \in \mathsf{S}.$$

and there exists a set $\mathsf{S}' \subseteq \mathsf{S}$ with $|\mathsf{S}'| \ge |\mathsf{S}| - 1$ and for all $s' \in \mathsf{S}'$ either $\underline{\delta}(s, a, s') = p_{(s,a)}(s')$ or $\overline{\delta}(s, a, s') = p_{(s,a)}(s')$.

We denote the set of all BFS of a state-action pair (s, a) by $\mathsf{BFS}(s, a)$.

In other words: we have three requirements for a basic feasible solution, namely (I) it must be a distribution (sum up to 1), (II) it must be consistent with the intervals and (III) it must coincide with the bounds of the intervals in at least $|\mathsf{S}| - 1$ places. The following example gives a graphical interpretation.

Example 14 (BFS)**.** *We exemplify the BFS of the state-action pair (q, c) of our running example in Figure 4.4. It shows a coordinate system, where every point in it corresponds to a vector of probabilities for going to q, $\mathsf{1}$ or o. The coloured regions indicate the restrictions imposed by the requirements.*

Requirement (I) prescribes that the resulting vector must be a distribution, thus restricting the possible solutions to the light grey plane. Intuitively, the corners of the triangle correspond to surely picking one of the successors, and the plane contains all convex combinations of these three points.

Requirement (II) is the consistency with the intervals, which is depicted as the dashed box. The intersection of the light grey plane and the box forms the dark grey plane. Every point in this intersection is a consistent solution for instantiating the intervals. The diamonds marking the corner points of the dark grey plane are the BFS of the state-action pair (q, c).

Note that by virtue of being on the border of the dashed box, every BFS fulfils Requirement (III), namely that the chosen probability is equal to one of the extremal values in at least all but one dimensions. △

By construction, every possible solution can be expressed as a convex combination of BFS [SVA06, Proposition 1]. Moreover, the number of BFS is bounded by $|\mathsf{S}| * 2^{|\mathsf{S}|-1}$ [SVA06, Lemma 6], i.e. at most exponential. Since there is no proof given in [SVA06], we think it is instructive to provide it here: the definition of BFS requires the probability for all but one successor state to be fixed to one of two extremal values. To find all BFS, we have to consider every one of n states to not be fixed. For each of them, the remaining $n-1$ states have exactly two possible values, resulting in 2^{n-1} possibilities per state. In the worst case, $n = |\mathsf{S}|$, if a state-action pair can reach all states. Realistically, $n = |\mathsf{Post}(s,a)|$ will often be a lot smaller.

4.2.2 Contribution: Reducing from bounded-parameter to standard systems

We utilize the idea of BFS from previous work to provide a general construction for reducing bounded-parameter systems to standard ones. We saw in the previous Section 4.2.1 that for every state-action pair, we can describe all possible instantiations of the intervals as a mixture of finitely many BFS. Hence, we can capture all possible instantiations by the following reduction: introduce a new state for every state-action pair and set its actions to be exactly the BFS. Formally, for a bSG $\mathcal{G} = (\mathsf{S}, \mathsf{S}_\Box, \mathsf{S}_\bigcirc, \mathsf{A}, \mathsf{Av}, \underline{\delta}, \overline{\delta})$, let $\mathsf{S}_{\mathsf{new}} := \{s_{(s,a)} \mid s \in \mathsf{S}, a \in \mathsf{Av}(s)\}$ be the set of new states. Then we define the transformed SG $G' := (\mathsf{S}', \mathsf{S}'_\Box, \mathsf{S}'_\bigcirc, \mathsf{A}', \mathsf{Av}', \delta')$ as follows.

- $\mathsf{S}' := \mathsf{S} \cup \mathsf{S}_{\mathsf{new}}$
- $\mathsf{S}'_\Box := \begin{cases} \mathsf{S}_\Box \cup \mathsf{S}_{\mathsf{new}} & \text{best-case semantics} \\ \mathsf{S}_\Box & \text{worst-case semantics} \end{cases}$
- $\mathsf{S}'_\bigcirc := \begin{cases} \mathsf{S}_\bigcirc & \text{best-case semantics} \\ \mathsf{S}_\bigcirc \cup \mathsf{S}_{\mathsf{new}} & \text{worst-case semantics} \end{cases}$
- $\mathsf{A}' := \mathsf{A} \cup \{p \in \mathcal{D}(\mathsf{S}) \mid s \in \mathsf{S}, a \in \mathsf{Av}(s), p \in \mathsf{BFS}(s,a)\}$.
- $\mathsf{Av}'(s) := \begin{cases} \mathsf{Av}(s) & \text{for all } s \in \mathsf{S} \\ \mathsf{BFS}(s,a) & \text{for all } s_{(s,a)} \in \mathsf{S}_{\mathsf{new}} \end{cases}$
- for every original state $s \in \mathsf{S}$ and $a \in \mathsf{Av}(s)$, set $\delta'(s,a,s') := 0$ for $s' \in \mathsf{S}$ and $\delta'(s,a,s_{(s,a)}) := 1$.
- for every new state $s_{(s,a)} \in \mathsf{S}_{\mathsf{new}}$, $p \in \mathsf{Av}'(s_{(s,a)})$ and $s' \in \mathsf{S}$, set $\delta'(s_{(s,a)}, p, s') := p(s')$.

This construction makes the interval-player explicit by adding a new state for the choice of the interval-player strategy. Since this new state has exactly the BFS as actions, the interval-player can choose every possible instantiation of the intervals. The new states belong to either the Maximizer or the Minimizer, depending on whether we are in best- or worst-case semantics.

The following Theorem 5 is our central claim: every behaviour of a bSG can be captured by the transformed SG. It is a generalization of [WMK19, Theorem 1]. To make this claim formally, we provide a way to map paths of the transformed SG back to paths in the original bSG.

Let ρ be a path in the transformed SG. By construction, it is an element of $(\mathsf{S} \times \mathsf{A} \times \mathsf{S}_{\mathsf{new}} \times (\mathsf{A}' \setminus \mathsf{A}))^* \times \mathsf{S}$, i.e. original and new state-action pairs alternate. We define $\mathsf{project}(\rho)$ to be the projection of this path to only elements of S and A. Concretely, for every subsequence $(s, a, s_{(s,a)}, p, s')$, we remove $s_{(s,a)}$ and p. The resulting sequence still is a path because $p(s') > 0$ and since $p \in \mathsf{BFS}(s, a)$, we also have $s' \in \mathsf{Post}(s, a)$ for some δ consistent with $\underline{\delta}$ and $\overline{\delta}$.

Theorem 5 (Every bSG can be transformed to an equivalent SG)**.** *Let $\mathcal{G}$ be an arbitrary bSG, (σ, τ) an arbitrary pair of Maximizer and Minimizer strategies and π an interval-player strategy. Further, let G' be the SG constructed from $\mathcal{G}$ as just described.*

We can construct a pair of Maximizer and Minimizer strategies (σ', τ'), such that the probability measures in both induced Markov chains are equal, i.e. for every $s \in \mathsf{S}$ and $X \subseteq \mathsf{Paths}_{G'}$

$$\mathbb{P}^{(\sigma,\tau,\pi)}_{s,\mathcal{G}}[\{\mathsf{project}(\rho) \mid \rho \in X\}] = \mathbb{P}^{(\sigma',\tau')}_{s,G'}[X]$$

Proof. Assume without loss of generality that $\mathsf{S}_{\mathsf{new}} \subseteq \mathsf{S}'_{\Box}$. Then $\tau' = \tau$ and for every path $\rho \in (\mathsf{S} \times \mathsf{A} \times \mathsf{S}_{\mathsf{new}} \times (\mathsf{A}' \setminus \mathsf{A}))^* \times \mathsf{S}_{\Box}$ in the new SG that ends in an original Maximizer state, we let $\sigma'(\rho) = \sigma(\mathsf{project}(\rho))$ be the same as in the original SG. The difference between the original and the new strategy is that Maximizer has to mimic the interval-player strategy in the new states $\mathsf{S}_{\mathsf{new}}$. Consider an arbitrary state-action pair (s, a). By [SVA06, Proposition 1], every distribution $\delta(s, a)$ that the interval-player chooses can be expressed as a combination of $\mathsf{BFS}(s, a)$. These are exactly the actions of Maximizer in the new state $s_{(s,a)}$ that is surely reached upon playing a in s. Hence, Maximizer can choose $\sigma'(s_{(s,a)})$ such that the resulting distribution over successor states equals $\delta(s, a)$. Then the resulting probability measure on paths projected to only the original states is equal, as there is one sure step and one step with the same distribution. For more details on the final step of the proof, we refer the interested reader to [SVA06, Proposition 2]. $\Box$

The central idea of letting a player choose the BFS in order to capture all possible instantiations of the intervals was introduced in [SVA06] and has been reused in several works since then. The contribution of Theorem 5 is to state the result in full generality: independent of objectives because it talks about the underlying probability measure; independent of semantics because it captures all behaviours of the interval-player; and for bSGs, not only bMDPs or bMCs.

Note that bSGs are the most general notion required as they are "the end of a chain" in the following sense: applying the construction to bMCs adds a player, effectively making them MDPs (which possibly is the intuition behind the term "IMDP semantics"). Applying the construction to bMDPs under worst-case semantics[11] also adds a player, effectively making them SGs. But when applying the construction to bSGs, there already are two players. Hence under both best- and worst-case semantics, the result of the transformation is an SG. Even when considering more complex systems such as concurrent bSGs, the

[11]This assumes it is an MDP with only Maximizer states. The dual statement holds for an MDP with only Minimizer states under best-case semantics.

construction remains basically the same: for every pair of actions, we add a new state to resolve the intervals, but we do not have to add a player.

We exemplify the **advantages of our general construction** in the next Section 4.2.3, as follows.

- We use the independence of the specification to derive Theorem 6 by a straightforward reduction to SGs.
- We use the independence of semantics of Theorem 5 and the natural way to capture them in the transformation to derive Corollary 1, showing that at-every-step and once-and-for-all semantics yield the same optimal values and strategies.
- We use the understanding of bounded-parameter systems gained from this general viewpoint to derive a more efficient algorithm for a special case, see Theorem 7.

4.2.3 Contribution: Application to ω-regular objectives

The motivation of our paper [WMK19] was to solve bMDPs with ω-regular objectives. Given the discussion of the previous Section 4.2.2, we know that we can transform a bMDP into an exponentially larger SG (as there are exponentially many BFS). Hence, we can apply solution methods for SGs with ω-regular objectives, e.g. [CH06, Algorithm 2]. Thus, we obtain the following theorem (see [WMK19, Theorem 2]):

Theorem 6. *Given a bMDP (and even bSG) with an ω-regular objective under best- or worst-case semantics, we can compute the optimal strategies as well as the value function in time exponential in the size of the transformed SG, which is, in turn, exponential in the size of the original bMDP (or bSG).*

Moreover, we know that memoryless pure strategies are sufficient for SGs with ω-regular objectives [CH12, Theorem 2]. Thus, we can give the following corollary of Theorem 5 (see [WMK19, Corollary 1]).

Corollary 1. *Given a bSG and an ω-regular objective, the optimal strategies and values are the same under the once-and-for-all and at-every-step semantics.*

Note that this result might be confusing at first sight: in some settings, ω-regular objectives require memory, but the corollary states that no memory is needed. This is because of a technical intricacy of the standard approach for model checking ω-regular properties: the objective can be specified on labels of the states of the system, in which case memory is required for optimal strategies. The solution approach is to transform the objective into an automaton and then construct the product between the system and the objective-automaton, see e.g. [BK08, Chapter 4]. In this product, the satisfaction of the objective depends only on the states of the system and no memory is required anymore. We exemplify this on the linear temporal logic objective that we already used in Example 13. For a more formal description, we refer to the appendix of [WMK19].

Example 15 (Memoryless strategies for ω-regular objectives)**.** *We consider again our running example of a bSG with the objective of looping in* $\mathtt{q}$ *once and then reaching* $\mathtt{1}$*. This* *objective can be captured by an ω-regular expression, namely $\mathsf{q}(\mathsf{q})^*(\mathsf{1})^\omega$. An automaton*[12] *for this objective has three states: the initial state s_1; the state s_2 that is reached after seeing the first q, where we loop upon seeing further q's; and the accepting state s_3 that is reached when the system enters state $\mathsf{1}$.*

Taking the product of this automaton for the objective and the system, i.e. our running example bSG, effectively creates three copies of the states S of the bSG by taking the product $\mathsf{S} \times \{s_1, s_2, s_3\}$. In the state (q, s_1) of the product, the interval-player uses the strategy that improves the chances of looping in q. If a loop occurs, the automaton proceeds to state s_2, thus switching the state of the product to (q, s_2). Note that, even though the system looped in q, the product has made progress, since we reach the new state (q, s_2) where the objective has been updated. In this state, the interval-player now maximizes the chances of reaching $\mathsf{1}$.

In summary, we see that memory is required when considering only the system since in state q two different decisions have to be taken. But when considering the product of system and objective, the memory is effectively part of the state-space and thus memoryless strategies suffice. Corollary 1 refers to the product of the system and objective. △

We highlight that both Theorem 6 and Corollary 1 rely on the generality of our approach. We did not have to apply specialized reasoning for bSGs, but could immediately reduce to SGs and transfer known results from literature. However, this generality comes with a drawback: the exponential blow-up in size when transforming the bSG to an SG. If there are not many actions and, more importantly, not many successors per state-action pair, this blow-up can be negligible in practice. However, we are also interested in solutions with better time- and space-complexity.

In particular, consider a bMDP with only Maximizer states under best-case semantics. Here, the resulting system after the transformation still is an MDP, which admits a polynomial solution algorithm. So we have to avoid the exponential blow-up of the transformation in order to obtain a polynomial solution algorithm for bMDP.

- Instead of adding a new state for every action, we can add the new actions to the same state, since all belong to the same player Maximizer.
- Since we choose the probabilities, we can avoid transitions disappearing. Thus, maximal end components (MECs) of the bMDP can be computed on the bMDP without transforming it, since this qualitative analysis of the graph does not rely on the precise transition probabilities, but only on edges being present or not. We can use [HM18, Algorithm 3] which runs in polynomial time [HM18, Proposition 6].
- The qualitative analysis of whether a MEC is winning can also be performed on the bMDP, see Step 1c) of our procedure in [WMK19, Section IV.C].
- Finally, computing the probability to reach the MECs can also be done in polynomial time [PLSS13, Theorem 4.1]. Intuitively, the constraints of the intervals can be added to the linear program that is used to solve the MDP.

[12]Note that the proposed automaton rejects words that cannot become part of its language by blocking. For example, upon seeing o, the automaton blocks, since there is no transition for o, and thus the objective is not satisfied.

Thus, we obtain an algorithm with better space- and time-complexity (see [WMK19, Theorem 3]).

Theorem 7. *Given a bMDP with an ω-regular objective under best-case semantics, we can compute the optimal strategy as well as the value function in time polynomial in the size of the bMDP.*

4.2.4 Related work

We first give an overview of the literature on bounded-parameter systems, i.e. Markov chains and MDPs with interval-valued transition functions. Afterwards, we mention several other notions of limited information that have been considered.

Bounded-parameter systems

Systems with intervals instead of exact transition probabilities were studied under a variety of names, with different semantics and objectives, and in several research communities. We found works from journals or conferences about formal methods [JL91] and model checking [SVA06, PLSS13, BDL$^+$17], control theory [WTM12, DC18], operations research [SJ73, NG05], planning [Buf05] and artificial intelligence [GLD00, WK08]. In the following, we aggregate papers according to their model (Markov chain or MDP), give their semantics (as distinguished in Definition 9) and mention their objective, possibly complemented by a comment on special extensions.

Interval Markov chains (IMC, called bMC in our technical sections) have been introduced independently by [JL91] and [KU02]. Both works use once-and-for-all semantics. The former [JL91] considers bisimulation of interval systems, so it does not distinguish between best- or worst-case. The latter [KU02] computes best- and worst-case step-bounded reachability values.

One work uses an uncommon combination of semantics: in [BLW13], the authors try to find the optimal probabilities to satisfy a linear temporal logic (LTL) objective, i.e. they consider best-case and once-and-for-all semantics. Their algorithm "converges, but not necessarilly [sic] to a global optimum." [BDL$^+$17, Page 2].

The remaining works we mention in the area of IMCs all consider both the once-and-for-all and at-every-step semantics, where [BDL$^+$17] even shows that these semantics are equivalent when considering reachability. There are works on model checking of IMCs in worst-case semantics with probabilistic computational tree logic (PCTL) [SVA06, CK15] and ω-PCTL objectives [CSH08], establishing complexity results. Note that unlike the others, [CK15] considers open intervals. Moreover, both worst- and best-case probabilities are computed for reachability in [BDL$^+$17] and for ω-regular objectives in [DC18].

Bounded-parameter Markov decision processes (bMDP) were already considered in the 70s [SJ73], but the term bMDP was introduced by [GLD00]. For a PhD-thesis summarizing research on modelling and performance analysis of bMDPs, see [Has17a].

Several works use the once-and-for-all semantics [SJ73, GLD00, TB07, WK08], as they quantify over all models and select one. Many others use a nature player or adversary to pick the probabilities; when this other player uses memory, we are in the at-every-step semantics [Buf05, WTM12, FDdB14, LAB15, HHH$^+$19b]. Both these semantics occur in [NG05]. Moreover, in [PLSS13, HM18] the authors even prove that they are the same because the interval-player can play optimally using only memoryless strategies.

Both best- and worst-case are considered in the setting of discounted reward [SJ73, GLD00], mean-payoff [TB07] and expected total reward [WK08]. Only best-case semantics is investigated in [HM18], giving a guaranteed-precision algorithm for reachability. Only worst-case semantics is investigated with the objective being stochastic shortest path [Buf05], expected total cost [NG05], linear temporal logic (LTL) [WTM12], probabilistic computational tree logic (PCTL) [PLSS13, LAB15], discounted reward [FDdB14] and combinations of expected rewards, reachability and ω-regular properties [HHH$^+$19b]. Note that [NG05, WTM12, PLSS13] consider more general uncertainties than just intervals.

One fundamental difficulty of interval systems is that transitions can "disappear" if their probability is set to 0 [Buf05, DC18]. This complicates the qualitative analysis of the graph (cycle-detection). In [CSH08, HHH$^+$19b], the authors also consider positive uncertainty, where every probability has to be greater than 0.

Finally, we mention that different measures of uncertainty can be considered and are surveyed in [Wal96]. The paper concludes that: "Each of the four measures seems to be useful in special kinds of problems, but only lower and upper previsions [(i.e. intervals)] appear to be sufficiently general to model the most common types of uncertainty" [Wal96, Page 1].

Other notions of limited information

The problem of not having precise information about every part of the model has been considered for decades from different viewpoints. Our contributions relate to the setting where we can obtain many simulations of the system, but know very little apart from that (see the previous Section 4.1); and (in this Section 4.2) we consider the setting where we know everything except the precise transition probabilities, on which we only assume to have intervals. We now comment on several other notions that have appeared in literature and point to publications that can serve as an entry-point to that area of research.

Firstly, we mention **parametric models**. These lift the assumption that the intervals on all transitions are independent. Instead, they consider models where several transitions can depend on the same unknown, but fixed parameter.[13]

We observed two approaches for dealing with parametric models: on the one hand, parametric model checking or parameter synthesis consider finding valuations of the parameters that satisfy a specification. In essence, they apply model checking to families of systems. See [Daw04, HKM08] for early works on this; [BDL$^+$17] for a comparison of IMCs, parametric Markov chains and parametric IMCs; [CDP$^+$17] for an application in biochemical systems; and [Jun20] for a recent and extensive overview.

On the other hand, given a parametric model and the possibility to run simulations in it, one can try to infer the actual parameter values. This corresponds to model checking the particular given system, with the added complication that the probabilities are unknown; in contrast to our simulation-based approach in Section 4.1, in this setting information about one transition also propagates to other transitions depending on the same

[13]Parametric verification should not be confused with the model checking of parameterized systems [CHVB18, Chapter 21], which contain arbitrary numbers of components.

parameter. This approach was considered in different research communities, including operations research [WKR13], foundations of software engineering [LBB$^+$18] as well as verification [MA20].

A variant of the parametric verification is a problem called **scenario-based verification**. It assumes that there is an underlying distribution of the parameters which can be sampled from. Then it considers the related question of what percentage of a number of sampled models is expected to satsify the specification [CJJ$^+$20].

Another generalization of interval systems are **constraint Markov chains** [CDL$^+$11]. In these, a given constraint function defines for every possible transition function whether it is permitted or not. In [CDL$^+$11], linear and polynomial constraints are considered, as they have desirable properties for closure and parallel composition of constraint Markov chains. Very similar is the model of Markov decision process with imprecise parameters, where the transition probabilities are also restricted by linear inequalities [WE94].

Further, instead of or additionally to assuming that the transition probabilities are unknown, one can also consider **unknown rewards**. This was already done in [Sil63, Section II] and [GLD00], but also more recently, see e.g. [HBK17, GHS18, BHL19], as well as [Fan21] for a recent practical case study. Similarly, when looking at continuous-time MDPs, one can consider unknown transition rates [KCS02].

Furthermore, there are **partially observable models**, most commonly partially observable MDPs (POMDPs) [Åst65, KLC98]. In these, the limited information is not on the transition function, but on the state-space. Instead of being able to know the current state, we only get an observation about the state; this observation is shared between multiple states, thus revealing only partial information. It is undecidable whether there exist strategies that satisfy a specification with a given probability [MHC03]; still, finding approximate solutions has been very actively investigated, see e.g. [RGT05, Spa12]. Note that there is a close connection between finding observation-based strategies in POMDPs and parameter synthesis, as well as concurrent SGs and software product lines [Jun20, Chapter 7].

A subclass of POMDPs are **multi-model** or **multi-environment** MDPs (MEMDPs). The basic idea of this setting is that we are given a concrete, finite set of possible transition functions and rewards, as opposed to the parameters with infinite domain in parameter synthesis. Previous work has considered the question of finding a strategy for all instantiations of the transition function [RS14]; for a given weight distribution over the transition functions [SKD21]; or, in [Sil63, Section I] and [CCK$^+$20], for a single run in a fixed MEMDP, inferring the environment from the history. A motivation for considering MEMDPs is that "several verification problems that are undecidable for partially observable MDPs are decidable for MEMDPs" [RS14, Page 1] and that "the specific structure of MEMDPs allows for more efficient algorithmic analysis, in particular for faster belief updates" [CCK$^+$20, Page 1].

Partially observable SGs are typically called **games with incomplete information**, see e.g. [DR11]. For an analysis of these and variations of them, see [Sor03]. Further, there are **perspective games**, where players can view only the parts of the system that they own [KV19, KS21].

Finally, we mention **modal transition systems**, introduced by [LT88] and recently surveyed by [Kre17]. In these, we get qualitative information about the transitions: they must be there (corresponding to an interval with $\underline{\delta} > 0$) or they may be there (corresponding to an interval with $\underline{\delta} = 0$). We refer to [Kre17] for information about their applications and extensions.

4.3 Summary, discussion and outlook

Summary In this chapter, we studied stochastic games (SG) with limited information. In Section 4.1, we can only simulate the system. This can either be because we do not know the full system, but only a lower bound on the transition probabilities (black-box) or only the local topology (grey-box); or because the system is so large that even though we know it precisely (white-box), non-statistical algorithms are infeasible. In all cases, we provide a statistical model checking algorithm with a PAC guarantee. It is the first one for SG and the first practically applicable one in the black-box setting. Moreover, it is an anytime algorithm which we can stop at any time during the computation and obtain an estimate of the current precision and confidence.

In Section 4.2, we give a general transformation from bounded-parameter systems to normal ones. The advantage of the transformation is its independence of the objective and the semantics. However, the transformed normal system is exponentially larger. Still, using reasoning based on the understanding we derived from our general procedure, we obtain a procedure with optimal complexity for a special case.

Discussion In a sense, the partial model that is built from the simulations in Section 4.1 is a bounded-parameter system. It is not immediately obvious because we define $\tilde{\delta}$ using the one-sided variant of the Hoeffding bound, and thus the resulting intervals always have an upper limit of 1. Our algorithm deals with this by using specialized Bellman equations (4.1) and (4.2). To have the same behaviour in a bSG, we could add a transition to the goal and one to a sink, both with interval $[0, 1]$. Then the upper bound (best-case semantics) always assigns all remaining probability to the transition going to the goal and the lower bound (worst-case semantics) to the sink. Alternatively, we can use the two-sided variant of the Hoeffding bound, which immediately yields intervals on the probabilities. Then, the BVI-phase basically amounts to transforming the bSG resulting from the simulation-phase into an SG and solving it by standard algorithms. This allows us to clearly identify whether the BVI-phase does not converge due to BVI needing more iterations or due to the intervals being too coarse. Previously, we could not distinguish these cases well. This combination — learning a bounded-parameter system from simulations — is used by [SSJP22], as well as our own work in Chapter 7. Learning a system from data in order to model check it has also been considered in e.g. [MCJ$^+$12, WSYP18, WSQJ21].

Outlook There are many interesting directions for future work. Firstly, we can further investigate the **combination of the approaches** we just discussed in order to learn models from real data. Note that depending on whether the learning is offline or online, further methods such as shielding [ABE+18] during the simulations become necessary, see also the last paragraph of Section 4.1.5.

Secondly, we discuss extensions of the **SMC-approach**. **Practically**, there is a huge potential for improvements of the algorithm. The number of simulations per simulation-phase is not only theoretically interesting but also an important hyper-parameter affecting the practical performance of the algorithm. Further, the bounds on the transition probabilities are usually very conservative. Using another bound, e.g. the Chen bound [Che15] as in [BGJS20], could improve this. Furthermore, we can distribute Δ not uniformly, but based on some heuristic to recognize which transitions are *relevant*. For a discussion of relevance, we refer to Section 3.1.3. Moreover, we mention that we can combine statistical model checking with importance sampling or importance splitting, see [LST16] for an overview. These techniques allow getting good estimations of rare events, i.e. transitions with low probability.

Given a good implementation and choice of hyper-parameters of the SMC algorithm, we can further compare it with non-statistical algorithms yielding hard guarantees such as bounded value iteration. From this, we can derive practical advice on when to use SMC and when a non-statistical algorithm; currently, the advice given in [ADKW20] only relates to Markov chains. Moreover, the more practical this approach is, the better the chances for using it in a model checking pipeline.

Theoretically, we can extend our algorithm to other objectives. An extension to ω-regular objectives relies on one key change: when considering ω-regular objectives there are no obvious goal or sink states. Thus, we have to detect that we have reached an EC as described in Section 4.1.2.2 and then investigate whether the objective can be satisfied in this EC. For this, only qualitative knowledge of the transitions is necessary. An orthogonal idea is to investigate the number of simulations per simulation-phase to not only prove the existence of a bound but the exact number.

Thirdly, we give a direction for future work on **bounded-parameter systems**. Here, our general construction allows to apply algorithms for normal systems to their bounded-parameter counterparts. Additionally, there is hope for finding algorithms with optimal complexity by investigating and improving the naive, exponential approach, as we did for Theorem 7.

Further, one might consider other semantics for the interval-player than only best- or worst-case. Assuming some kind of fairness would result in a probability distribution over BFS and could easily be incorporated without blowing up the bSG. Alternatively, one might use things such as Minimax Regret Optimisation [RLH21], which not only considers the worst-case but additionally tries to optimize for all instantiations.

Finally, we refer to our extensive discussion of related work considering different settings with limited information in Sections 4.1.5 and 4.2.4, as well as the system and specification modifications in Sections 1.5.2 and 1.5.3. Insights and ideas from this thesis can potentially be transferred to those other settings.

5 Expressive specification formalisms: Multiple objectives

Motivation So far, we have given algorithms to verify that a system satisfies a specification which is modelled as *one* objective. However, we are often interested in multiple, possibly conflicting objectives, e.g. stay safe and maximize a reward while minimizing cost or energy consumption. Again, we focus on the most fundamental case: a generalized-reachability objective. This problem is not only practically relevant but also theoretically: the precise decidability of generalized-reachability stochastic games is still open.

Problem statement Similarly to Chapter 3, we are given a stochastic game (SG) G and consider reachability. The difference is that instead of one set of goal states, in this chapter we have a generalized-reachability objective, which is an n-tuple $\mathcal{T} = (\mathsf{F}_1, \dots, \mathsf{F}_n)$ of goal state sets $\mathsf{F}_i \subseteq \mathsf{S}$. Thus, instead of values, we now compute vectors of values, where every component corresponds to one objective. For a vector $\vec{\mathsf{V}}$, we denote by $\vec{\mathsf{V}}_i$ its i-th component.

We are interested in the function $\mathbb{A} : \mathsf{S} \to 2^{[0,1]^n}$, which maps every state to the set of vectors achievable from it. Formally, given an SG G and a generalized-reachability objective $\mathcal{T}$, for all states $s \in \mathsf{S}$ (in particular the given initial state s_0) we want to compute

$$\mathbb{A}(s) := \{\vec{\mathsf{V}} \mid \exists\sigma.\forall\tau.\forall i \in 1, \dots, n : \mathbb{P}^{\sigma,\tau}_{s,G}[\Diamond \mathsf{F}_i] \geq \vec{\mathsf{V}}_i\}$$

It is currently not known whether finding the exact function $\mathbb{A}$ is decidable, but we show that and how it can be approximated with arbitrary precision $\varepsilon > 0$.

Note that we only consider a conjunction of objectives and that we require that Maximizer fixes the strategy first. See Section 5.3 for a discussion of the implications of this as well as possible alternatives.

Chapter outline In Section 5.1, we start by introducing some notation and geometric notions, as well as recalling the previous solution. Section 5.2 describes our algorithm for approximating the set of achievable vectors from the paper [ACK+20] (Appendix F). Finally, in Section 5.3 we discuss related work (including the papers [CKWW20, WW21] mentioned in Section 1.6) and in Section 5.4 we summarize and discuss future directions.

5.1 State of the art

We illustrate the important concepts mainly by means of examples and refer to [ACK+20] for the completely formal treatment. To speak about sets of achievable vectors, the following definition is very useful: Given a set $X \subseteq [0,1]^n$ of n-dimensional vectors, its *downward closure* is defined as $dwc(X) := \{y \mid \exists x \in X : y \leq x\}$.

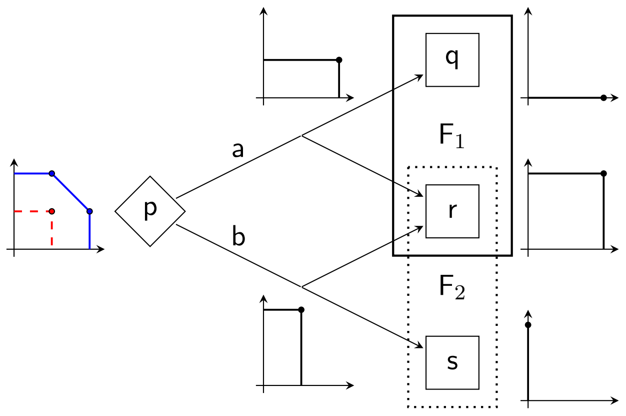


Figure 5.1: An SG with a generalized-reachability objective and the resulting sets of achievable vectors. The goal sets are $\mathsf{F}_1 = \{\mathsf{q},\mathsf{r}\}$ and $\mathsf{F}_2 = \{\mathsf{r},\mathsf{s}\}$. The transition probabilities of actions a and b are uniform, i.e. $1/2$. We do not depict the self loops of states q, r and s. The state p is not drawn as box (Maximizer) or circle (Minimizer), but rather as a diamond because we consider both cases: the blue solid line depicts the set of achievable vectors if it is a Maximizer state and the red dashed line if it is a Minimizer state.

Example 16 (Generalized-reachability objective)**.** *Consider the example in Figure 5.1 of an SG with two goal sets F_1 and F_2. The sets of achievable vectors are shown for all states, as well as the state-action pairs (p,a) and (p,b). Since we have two objectives, we can depict them in a two-dimensional coordinate system where the horizontal axis corresponds to F_1 and the vertical axis to F_2.*

The state r is in both goal sets, so it reaches both goals surely. Thus, every vector in $\mathbb{A}(\mathsf{r}) = dwc(\{(1,1)\})$ is achievable. Geometrically, this corresponds to a square in the first quadrant of the 2-dimensional coordinate system; in higher dimensions, the achievable vectors of a state in all goal sets form an n-dimensional cube.

Each of the states q and s is in one of the goal sets, thus we have $\mathbb{A}(\mathsf{q}) = dwc(\{(1,0)\})$ and $\mathbb{A}(\mathsf{s}) = dwc(\{(0,1)\})$. Both of these correspond to an axis of the coordinate system.

By playing action a, the set $\mathsf{F}_1 = \{\mathsf{q},\mathsf{r}\}$ is surely reached, but F_2 only with a probability of $1/2$. This corresponds to the rectangle $dwc(\{(1,0.5)\})$. Dually, we obtain the rectangle $dwc(\{(0.5,1)\})$ for action b.

For state p, we depict two sets, depending on whether p is a Maximizer or Minimizer state. The blue solid line corresponds to Maximizer and the red dashed line to Minimizer. On the one hand, Minimizer obtains the intersection of the available actions. This is because we cannot rely on any target being reached with a probability of more than 0.5. On the other hand, Maximizer obtains not only the union, but even the convex hull of the union. This is because Maximizer can use a randomized strategy to obtain vectors such as $(0.75,0.75)$ that are not achievable by either action alone, but by a convex combination, in this case playing both actions with a probability of $1/2$.

We see that even in this simple example, there are trade-offs between the objectives, since reaching one goal can make it impossible to reach another. Knowing all achievable vectors is useful because then we can (i) select a trade-off that we like based on full information

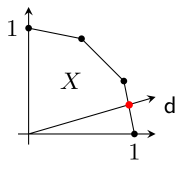


Figure 5.2: Example showing a Pareto-frontier of a set X, a direction d, and the point of intersection of d with the frontier, depicted as ● in distance $X[\mathsf{d}]$ from the origin. Taken from [ACK+20, Figure 1].

about what is possible and (ii) decide whether a certain combination of thresholds for all objectives is achievable. △

We want to be able quantify the best trade-off in a certain *direction*, a certain "ratio" of the objectives. Intuitively, a direction is a ray from the origin into the (n-dimensional) first quadrant. As such, we may represent it with any vector $\vec{v} \neq \vec{0}$ on that ray. Then all vectors $\lambda \cdot \vec{v}$ for any $\lambda \in \mathbb{R}_{>0}$ are equivalent and represent the same direction. For instance, direction $\mathsf{d} = [(1,0,0)]$ denotes the x-axis and it is equal to $[(\lambda,0,0)]$ for any $\lambda > 0$. Formally, a direction $\mathsf{d} = [\vec{v}]$ is the set $\{\lambda \cdot \vec{v} \mid \lambda \in \mathbb{R}_{>0}\}$. We denote by $\mathsf{D} = \{[\vec{v}] \mid \vec{v} \in \mathit{dwc}(\{\vec{1}\})\}$ the set of all directions (in the first quadrant).

Given a set X of vectors and a direction d, X *evaluated in direction* d is the (Euclidean) length of the vector from the origin to the farthermost intersection of X and d, denoted

$$X[\mathsf{d}] := \sup\{||\vec{x}|| \mid \vec{x} \in X, \mathsf{d} = [\vec{x}]\}$$

with the usual $\sup \emptyset = 0$. Figure 5.2 illustrates an evaluation of a direction on an achievable set.

Note that a set of achievable points X is uniquely identified by its *Pareto-optimal* vectors, i.e. those $x \in X$ such that there is no $y \in X$ with $x \leq y$. Intuitively, we cannot increase one component x_i of a Pareto-optimal vector without decreasing some other component x_j. In particular, in Example 16 we could describe many achievable sets as the downward closure of a single vector. Considering Figure 5.2, we see that it can also be possible that the Pareto-optimal vectors are the whole "upper right border" of the set. For technical reasons, we always want to consider the whole border in all directions. Thus, we define the *Pareto-frontier* of a set X as

$$\mathcal{P} := \{\vec{x} \mid \mathsf{d} \in \mathsf{D}, \mathsf{d} = [\vec{x}], X[\mathsf{d}] = ||\vec{x}||\}$$

We also define the Pareto-frontier of a state s as $\mathcal{P}(s) := \mathcal{P}(\mathbb{A}(s))$. Note that we do not know whether the set $\mathbb{A}(s)$ is closed and hence whether the Pareto-frontier is actually part of it or not — more formally: whether the suprema that are computed when evaluating $\mathbb{A}(s)[\mathsf{d}]$ are elements of $\mathbb{A}(s)$. But since our goal is to *approximate* the set of achievable vectors, we can equivalently approximate $\mathcal{P}(s)$ for all states.

Previous work [CFK+13b] gives an algorithm based on value iteration (VI) to approximate the Pareto-frontier from below. The idea is similar to single-dimensional VI, see Section 3.1: we start with a lower bound on the set of achievable vectors, i.e. every state can achieve nothing and goal states achieve 1 in the dimension of all goal set that they are part of. Then we iteratively apply a variant of Bellman updates (see Equations (3.3) and (3.4) for the single-dimensional case and Equations (5.1) and (5.2) below for the multi-dimensional extension) and hence backpropagate the values through the SG. In the i-th iteration, the lower bound contains exactly those vectors that are achievable within i steps. Thus, the algorithm converges to the Pareto-frontier in the limit [CFK$^+$13b, Theorem 4].

$$\mathsf{L}_n(s) := \begin{cases} \mathsf{conv}(\bigcup_{a \in \mathsf{Av}(s)} \mathsf{L}_n(s,a)) & \text{if } s \in \mathsf{S}_\Box \\ \bigcap a \in \mathsf{Av}(s) \mathsf{L}_n(s,a) & \text{if } s \in \mathsf{S}_\bigcirc \end{cases} \tag{5.1}$$

$$\mathsf{L}_n(s,a) := \sum_{s' \in \mathsf{S}} \delta(s,a,s') \cdot \mathsf{L}_{n-1}(s') \tag{5.2}$$

In Equations (5.1) and (5.2), we omit several technical details: the formal definitions of scaling a set by a constant, summing of sets and taking the convex hull conv (see [ACK$^+$20, Section 2.4]) and the corner cases of a state itself or successor states being a goal states, in which case we set the corresponding dimension to 1 (see [ACK$^+$20, Section 2.6]). Note that the intuition of these multi-dimensional Bellman equations is the same as for the single-dimensional ones and the same that we sketched at the end of Example 16.

5.2 Contribution: Approximating the Pareto-frontier in generalized-reachability stochastic games

The contribution of this chapter is the multi-dimensional analogue of Section 3.1.2: previous work gives a VI-algorithm that converges in the limit, but cannot quantify its progress in general. In particular, [CFK$^+$13b, Theorem 4] only talks about stopping SGs, i.e. SGs without end components (EC). Thus, we extend the algorithm by adding a convergent sequence of over-approximations, turning it into an anytime algorithm with guaranteed precision. For this, we need to treat the ECs by using a multi-dimensional extension of *deflating*. As a result, we obtain the central theorem, see [ACK$^+$20, Theorem 5.4]:

Theorem 8. *Given an SG G with a generalized-reachability objective $\mathcal{T}$, we can compute convergent monotonic under- and over-approximations L and U of the Pareto-frontier $\mathcal{P}(s)$ for every state $s \in \mathsf{S}$. Thus, for every $\varepsilon > 0$ there exists an iteration i such that for every $s \in \mathsf{S}$ and direction $\mathsf{d} \in \mathsf{D}$ we have*

$$\mathsf{U}_i(s)[\mathsf{d}] - \mathsf{L}_i(s)[\mathsf{d}] < \varepsilon.$$

In other words: we can ε-approximate the Pareto-frontier (and hence the set of achievable vectors) to arbitrary precision.

We highlight the key difficulties that we had to overcome in order to achieve this theorem. In essence, we want to reduce the multi-dimensional case to the single-dimensional case. For this, we can consider any single direction in which we evaluate our approximants and then apply deflating as usual. However, as we have an infinite continuum of directions, we have to group them into finitely many regions. The most technically involved part of the problem are these regions, in particular the projective geometry necessary to describe them, their relation to deflating and the argument that they can effectively be computed. We illustrate the ideas on a two-dimensional example.

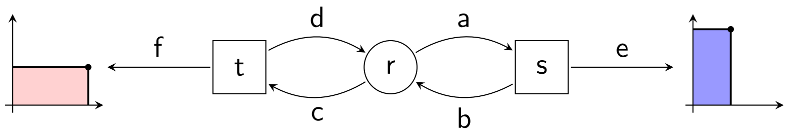


Figure 5.3: Example of an EC where we need to distinguish regions in order to deflate.

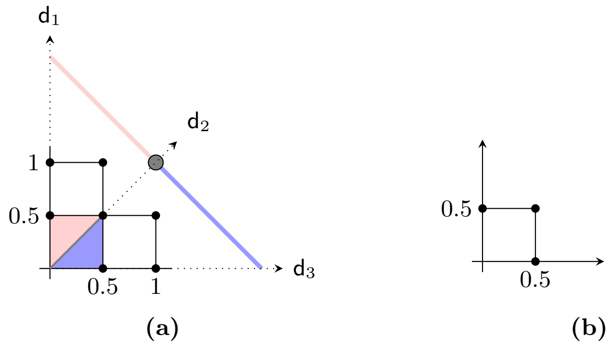


Figure 5.4: Visualizing (a) the regions for state r and (b) the result of deflating. Based on [ACK+20, Figure 4].

Example 17 (Multi-dimensional deflating)**.** *Consider the SG in Figure 5.3, which is the same as Figure 3.2. The difference is that this time the values on the exits are not numbers, but Pareto-frontiers. Recall that in the single-dimensional case, depending on the relation of the exits, Minimizer used different strategies, and thus different parts of the maximal end component (MEC) had to be deflated, see Example 7.*

In the multi-dimensional case, Minimizer's strategy changes depending on the direction in which we evaluate the Pareto-frontier. Figure 5.4a shows both Pareto-frontiers of the exits in a single coordinate system and highlights three directions d_1*,* d_2 *and* d_3*. For every direction between* d_1 *and* d_2*, Minimizer prefers the exit* f*, and thus state* r *will play action* c*. Dually, for every direction between* d_2 *and* d_3*, exit* e *is preferred by Minimizer, so an optimal strategy will play* b *in* r*. In direction* d_2*, both actions are optimal.*

Note that this explanation has effectively decomposed the set of all directions into regions where Minimizer's strategy is the same. Thus, we can apply deflating, reducing to the Pareto-frontier in the red region between d_1 *and* d_2 *to that of exit* f*; and in the blue region between* d_2 *and* d_3 *to that of exit* e*. Technically, we also have to deflate the single direction* d_2*, which fits in with the other segments because it was at the intersection of the Pareto-frontiers. Similarly, direction* d_1 *and* d_2 *add the corner points on the axes.*

The result is depicted in Figure 5.4b. Intuitively, this result is correct, as depending on which goal set Maximizer prefers, Minimizer can always restrict the resulting value to 0.5 by restricting Maximizer to the exit that does not reach this goal surely.

Note that within a region, Minimizer's strategy stays constant. A sufficient criterion for this is that for all directions in the region, the ordering of the actions according to their value in that direction is the same. Thus, we can decompose the set of all directions into regions by splitting it at all places where the Pareto-frontiers of the actions intersect. In Figure 5.4a, the only intersection is direction d_2*.* △

When having more dimension than two, the same idea for finding regions still works. Since all Pareto-frontiers are finite, the intersection of finitely many Pareto-frontiers also yields finitely many directions in which we have to split. To effectively represent these, we use *simplicial complexes* and *triangulation*, as exemplified in [ACK$^+$20, Figure 5 and 6] and described in [ACK$^+$20, Algorithm 3 and Sections 3.3 and 4.3].

5.3 Related work

We first give some general comments about the computation and usage of Pareto-frontiers. Then we focus on the generalized-reachability objective and discuss in detail its decidability, its computational and strategy complexity and its determinacy.

Next, we mention variants of multi-objective specifications that go beyond optimizing expectation. Afterwards, we give an extensive overview of other works considering multiple objectives, roughly grouping them into those more interested in computational and strategy complexity on the one hand and those more interested in practical algorithms for computation or approximation on the other hand. Finally, we list tools and case studies.

General comments about computation and usage of Pareto-frontiers

In general, Pareto-frontiers can be approximated by a number of vectors polynomial in the size of the system [PY00, Theorem 1] and often also in polynomial time [PY00, Theorem 2]. To support our argument that a Pareto-frontier is a good foundation for making a decision in a multi-objective setting, we point out that deciding which Pareto-optimal outcome to use is the topic of a whole area of research called *multiple criteria decision making*. According to an extensive book on this topic, the goal is "understood as helping a human decision maker (DM) in considering the multiple objectives simultaneously and in finding a Pareto-optimal solution that pleases him/her the most" [BDMS08, Page 2]. To support a decision maker, we can use techniques for describing the cartography of the problem [BGMR18] and visualizing trade-offs [LPLSA17, HJKQ20].

Generalized-reachability objectives

To highlight the increase in difficulty when considering generalized-reachability SGs, we report their properties in relation to single-objective SGs and Markov decision processes (MDPs).

Decidability: The decision problem is to decide whether a given vector is achievable. While SGs with a single reachability objective [Con92] and generalized-reachability MDPs [EKVY08] are decidable, it is not known whether generalized-reachability SGs are decidable. We know decidable subclasses, namely if the SG is determined [BF16b, Theorem 3] or if it is stopping and there are only two objectives [BF16b, Theorem 6]; and we know undecidable subclasses, namely if we restrict to pure strategies [CFK$^+$13b, Theorem 3]. Moreover, by our work [ACK$^+$20] we know that the approximation problem is decidable, i.e. if we want to decide whether an ε-approximation of a given vector can be achieved.

Strategy complexity: For simple SGs, memoryless pure strategies suffice to achieve the optimal value [Con92]. Already in generalized-reachability MDPs with absorbing goal states (i.e. goal states with only one action that self-loops surely), we need randomization [EKVY08, Theorem 3.2]; without the assumptions that goals are absorbing, we additionally need finite memory [EKVY08, Theorem 5.1]. When moving to generalized-reachability SGs, we require infinite memory even if the SG is stopping and all goal states are absorbing [CFK$^+$13b, Theorem 2].

Computational complexity: Deciding simple SGs is in UP $\cap$ co-UP [CF11, Theorem 4]. Generalized-reachability MDPs are PSPACE-complete [RRS17, Theorem 2]. However, the hardness comes mainly from the number of objectives; the solution time is polynomial in the size of the MDP [EKVY08, Theorem 5.1]. As decidability is open for generalized-reachability SGs, of course we cannot give computational complexity. Still, we want to point out two things: firstly, one might wonder why the idea of [CH08] for proving that value iteration takes at most exponentially many steps cannot be extended to multiple objectives. The reason for this is that the infinite memory allows the strategies to introduce arbitrarily small probabilities and thus we cannot give a lower bound on the progress that an update of value iteration makes.

However, if we consider approximation, we should be able to apply this idea again, since infinite memory is not necessary in that case. Thus, we conjecture that the approximation-algorithm should terminate within an exponential number of iterations.

Algorithms: For simple SGs, we have three classes of algorithms, namely value iteration, strategy iteration and quadratic programming. For generalized-reachability MDPs, there is an approach based on linear-programming [EKVY08], but the authors argue that value or strategy iteration can equivalently be used [EKVY08, Section 6]. For generalized-reachability SGs, we only know of the convergent value iteration algorithm from [CFK$^+$13b] which we extended to be an anytime algorithm with precision guarantees. Note that an approach based on strategy iteration does not seem promising because strategies can require infinite memory.

We mention the close connection between this Chapter 5 and Section 3.1: both of them develop a practical stopping criterion for value iteration. Thus, we can possibly extend improvements mentioned in Section 3.1.4, the related work section of the single-dimensional case, to the multi-objective setting.

Determinacy and its implications: Recall that a game is determined if the order in which the players fix their strategy does not matter. While simple SGs are determined, see e.g. [Con93, Section 1.1.2], generalized-reachability SGs are not [CFK$^+$13b, Theorem 1].[1] This means that there are two problems to be considered: the “lower-value” problem, where we are interested in what a single strategy can guarantee against all strategies of the opponent; and the “upper-value” problem, where we are interested in what we can guarantee by reacting optimally to every strategy of the opponent. Instead of changing the order in which we quantify the strategies, we can also switch from considering conjunctions of objectives to considering disjunctions, as explained in [WW21, Section 5]. The upper-value problem, i.e. answering disjunctive queries, is fundamentally easier than the lower-value problem that we address in this chapter. In particular, memoryless pure strategies

[1]This theorem relies on reducing the objective to a *precise* query, which assumes that we mix objectives in which we maximize and minimize. For an example showing non-determinacy even when only maximizing, we refer to [WW21, Section 1].

are sufficient and the problem is NP-complete [CFK$^+$13b, Theorem 7 and Corollary 1]. We refer to our work [WW21] for an in-depth discussion of the computational and strategy complexity, also addressing the cases of deterministic strategies or non-stochastic games.

In general, the question of considering conjunctions, disjunctions or general Boolean combinations of objectives is interesting and has massive implications, as we discuss at the end of the section on complexity.

Variants of the multi-objective specifications

Most of the works we mention optimize the expectation of the objectives. Alternatively, one can consider **percentile** queries, which require that the value of an objective is better than a given threshold with a specified probability (percentile) [FKR95, RRS17, BGMR18]. Very similar are **quantiles**, where we combine several thresholds (on probability or reward) with the optimization of an expectation [UB13, BDD$^+$14, BDK14a, HJKQ20]. See [BDK$^+$14b] for an experience report from interdisciplinary research projects where, among others, the idea of using quantiles arose.

A corner case of percentiles are the *satisfaction semantics* for mean-payoff objectives, where we maximize the probability that the reward is above a threshold [BBC$^+$14, CKK17]. Energy objectives, introduced in [CdAHS03] are similar in spirit because they also require the value to be above a threshold; however, they do not consider the long-run average, but have to stay above the threshold for the whole run.

Another special case of percentiles is when some goals have to be achieved *surely*. This is called **beyond worst-case** paradigm [RRS15, CR15, AKV16, BFRR17, BRR17], where one constraint is surely satisfied and afterwards other performance criteria are considered.[2]

Further, one can also consider combinations of **expectation and variance** [BCFK17, PSB22] or **ratios** between objectives [Bai15, BD18]. Also, in artificial intelligence, another notion of optimality instead of Pareto-optimality has been used, namely **Lorenz-optimality** [PWGH13]. In literature on decision making, approaches exist to immediately get well-balanced trade-offs instead of the whole Pareto-frontier [OPW13].

Furthermore, we can assume a strict preference order over the objectives, which is also called **lexicographic preferences** [Fis74]. This idea assumes that one objective (e.g. saving a human) is strictly more important than others (e.g. minimizing energy consumption). In our paper [CKWW20] we treat combinations of multiple reachability and safety objectives with lexicographic ordering. For further works with lexicographic preferences, we refer to the related work of [CKWW20]. Here, we only mention that the beyond worst-case paradigm essentially also is lexicographic: it first optimizes the sure objectives and then the strictly less important, quantitative ones.

Complexity

The papers in this section investigate decidability or computational or strategy complexity.

MDPs were considered with multiple ω-regular objectives [EKVY08] and generalized-mean-payoff objectives under satisfaction or expectation semantics, as well as combina-

[2]In some of these papers, the underlying graph is considered once as a deterministic game for those objectives that need to be satisfied surely and once as an MDP for the other performance criteria.

tions of both semantics [Cha07a, BBC$^+$14, CKK17]. Further, combinations of energy, mean-payoff and parity were investigated in [CD11] and combinations of discounted reward objectives in [CMH06, CFW13]. Lastly, we list papers analysing the complexity of variants of multi-objective model checking: combinations of percentile queries with many kinds of quantitative payoffs (including mean-payoff and discounted reward) [RRS17]; combinations of expectation and variance [BCFK17]; and beyond worst-case analysis for mean-payoff [CR15], total reward [BFRR17], ω-regular objectives [BRR17] or a combination of an ω-regular objective with mean-payoff [AKV16].

Deterministic games were considered with generalized-mean-payoff objectives [BR15] and combinations of mean-payoff with ω-regular [CRR14] or energy objectives [CDHR10, BHRR19], as well as windowed mean- and total-payoff [CDRR15]. In particular, multi-dimensional total-payoff games are undecidable [CDRR15, Theorem 2].

Stochastic games were investigated with combinations of total reward [CFK$^+$13b], generalized-mean-payoff objectives [BKTW15, CD16] and combinations of mean-payoff with ω-regular [CDGO14] or energy objectives [VCD$^+$15]. Considering lexicographic preferences, we again refer to the related work of [CKWW20].

Recently, multiple mean-payoff objectives were also considered on Vector Addition Systems with States (VASS) [CHO20].

So far, we have mainly talked about conjunctions of objectives, where the goal is to satisfy several requirements. Disjunctions are the topic of the already mentioned [WW21]. Considering **general boolean combinations** of objectives often is a lot harder. We know solutions for total reward objectives can be approximated for Markov chains [HKL17], stopping SGs [CFK$^+$13b] and controllable multi-chain games [BKW18], but computability in these is still open. Boolean combination of mean-payoff objectives are undecidable [Vel15], but a relevant subclass is EXPTIME-complete [BHR16]. We know algorithms for Boolean combinations of mean-payoff and Büchi [GH10] and Boolean combinations of ω-regular objectives [BGR20].

Algorithms

First, we mention that multi-objective model checking is also prevalent in other fields, see e.g. [MBB07] for a paper on multiagent planning and [VDB$^+$11, RVWD13] for surveys from machine learning and from artificial intelligence, respectively.

From the formal methods side, a lot of work has been invested into a **compositional approach** for model checking MDPs [EKVY08], probabilistic automata [FKN$^+$11, FKP12, KNPQ13], Markov automata [QJK17, QK21] and SGs [SK16, BKW18]. The objectives of all cited works include expected reward or cost and additionally sometimes mean-payoff, ratios or ω-regular objectives. We highlight that [FKP12, QK21] report huge improvements in the scalability of algorithms approximating Pareto-frontiers. Similarly, quantiles for reachability and expected cost are efficiently computed in [HJKQ20]. Orthogonally, [DKQR20] investigates multiple expected reward objectives when restricted to simple strategies, i.e. pure ones with finite memory.

Multi-objective model checking was also considered in settings with **limited information** (see also Chapter 4), for example using many simulations to obtain a probably approximately correct (PAC) guarantee [JJD$^+$16, WT21], optimizing a single run in the system [KPR18], considering bounded-parameter systems [HHH+19b] or partially observable systems [DDG+10].

Tools and case studies

Multi-objective model checking is supported by multi-gain [BCFK15] (only for mean-payoff), PRISM-games [KPW16][3] the MODEST TOOLSET, as utilized in e.g. [HJKQ20], and there are prototypical implementations in STORM [QK21].

Case studies include cost-time-quality trade-off for optimizing access to the World-Wide Web [PY00], delay-cost trade-offs in the Mariposa database system [PY01], autonomous driving [CKSW13], electric vehicle charging [BGMR18], energy-aware network systems [BCD+18] and further case studies collected in [SK16, Section 6].

5.4 Summary, discussion and outlook

Summary In this chapter, we have considered SGs with a conjunction of reachability objectives. We gave an anytime algorithm for approximating the Pareto-frontier to arbitrary precision and thereby proved that the approximation problem is decidable.

Discussion The result currently is quite theoretical because the computation of the regions is described on a very high level. An implementation of the necessary geometrical notions is involved, especially since the Pareto-frontiers can become very large.

Still, the theoretical and algorithmic contributions are a significant step forward because — as in the single-objective case — reachability is underlying many other objectives.

Outlook The most immediate open question is whether generalized-reachability SGs are not only approximately decidable but also precisely.

However, since practically we are more interested in approximations and finite-strategy solutions anyway, the computational complexity of our algorithm should be investigated, as well as the relation between a bound on the memory and the quality of the result.

Focussing even more on practical progress, we plan to implement the algorithm, possibly combining it with simulation-based methods as in Section 3.1.3.

Of course, we are also interested in more expressive formalisms for systems and specifications, e.g. concurrent SGs or reward-based objectives. We refer to the concepts listed in Sections 1.5.2 and 1.5.3, as well as the discussion of variants of multi-objective specifications in Section 5.3.

[3]Note that we cite not the most recent version of PRISM-games because version 2.0 was the one that introduced the support for multiple objectives.

6 Explainable output

Motivation So far, we have focussed on computing optimal strategies, which prescribe how an agent has to behave in order to satisfy a specification. However, these strategies are typically very large and incomprehensible, taking the form of a long list of state-action pairs.

We want to find a way to represent these strategies *concisely* and *explainably*. There are two key motivations for this: firstly, we want to be able to use the strategy in practice, i.e. store it on a device with limited memory and execute it efficiently. Hence, we need a concise representation because systems and strategies are very large. This is related to the challenge of scalability, see Section 1.2. Secondly, related to the challenge of reliability, we want to be able to understand the strategy that results from solving our system in order to validate it. Hence, it has to be small enough that it becomes human-readable. Moreover, the representation should be explainable.

In the past decade, there have been several works using the formalism of *decision trees* for this task of strategy representation. They are well-suited for it because (i) they allow keeping the original guarantees of the strategies, (ii) they significantly reduce the size of the strategies (iii) they can easily be transformed into executable code, but also naturally be interpreted by humans.

Problem statement We are given a memoryless pure strategy $\sigma : \mathsf{S} \to 2^{\mathsf{A}}$. Note that this is a nondeterministic (also known as permissive) strategy which recommends a set of optimal actions. Further, we highlight that this strategy can be the result of any kind of procedure and we are not restricted to the systems or specifications addressed in this thesis, nor do we require that the strategy gives guarantees.

We want to represent this given strategy using a decision tree that is small, i.e. consists of few nodes; and that is explainable, i.e. there is some intuitive explanation or deeper reasoning behind the decisions of the tree.

Moreover, we not only want to solve this problem theoretically but offer a well-designed, easy-to-use tool that performs the task.

Chapter outline In Section 6.1, we describe the state of the art for representing strategies as decision trees. Then in Section 6.2, we describe the tool dtControl[1] that we developed for this task. We describe all components of the tool but focus on those features that were added in the newest version of the tool, dtControl 2.0. This is because this chapter is based on [AJK+21a] (Appendix G), the publication where the extensions of the second version of dtControl are described. Afterwards, in Section 6.3 we provide related work on decision trees, alternative forms of strategy representation and other ways of pursuing explainability. As usual, we finish with a discussion and an outlook in Section 6.4.

[1]Available at `dtcontrol.model.in.tum.de`

6.1 State of the art

For this chapter, we assume that our state-space is a cross-product of state-variables. This means that states are not monolithic elements $s \in \mathsf{S}$, but rather tuples $(s_1, \dots, s_n) \in \mathsf{S}_1 \times \cdots \times \mathsf{S}_n$, where S_i is the domain of the i-th state-variable. We highlight that almost all case studies have a state-space that is structured as a tuple of state-variables like this, and we exemplify it in our running Example 18.

Further, we slightly adapt the definition of strategy: firstly, we consider nondeterministic (also called permissive) strategies. The reasons are twofold: when executing the strategy, it can be useful to permit all actions that are safe and not arbitrarily restrict to a particular one; and, as we explain in Section 6.2.3, it is easier to reduce the size of nondeterministic strategies when representing them as decision trees. For more information on permissive strategies, see e.g. [BJW02, BMOU11, DFK$^+$15]. Secondly, we restrict our strategies to be memoryless and pure because only these kinds of strategies can be represented by decision trees well. Note that this is a relevant class of strategies, as they are optimal for objectives such as reachability, safety and expected reward; we can even consider ω-regular objectives after taking the product of state-space and system, see Example 15 in Section 4.2.3.

Definition 11 (Nondeterministic strategy)**.** *A nondeterministic (memoryless pure) strategy is a function $\sigma : \mathsf{S} \to 2^{\mathsf{A}}$ mapping every state to a set of available actions.*

Example 18 (Motivating example)**.** *This example is based on [LMT15]. Assume that we are controlling a car that is driving in a lane behind another car, see Figure 6.1. We do not consider more cars or lane-switching, but only focus on driving behind the other car in a safe distance. Our actions are braking, accelerating or doing nothing. Our state-space consists of triples (v_e, v_f, d), where $v_e, v_f, d \in \mathbb{Q}$ are the current valuations of the state-variables: **v**elocity of our car (**e**go), **v**elocity of the car in **f**ront and **d**istance between the cars.*

This abstraction of controlling a car is very simple, especially since we bound the velocities at around 60 km/h. Still, the resulting strategy as produced by Uppaal Stratego [DJL$^+$15] consists of more than 300,000 state-action pairs and has a size of 418 MB. This is potentially too large to fit into the memory of the controller of the car, especially when considering more realistic domain sizes of the velocities; and already now it is completely incomprehensible for a human.

However, the strategy has some inherent structure. For example, if the distance d becomes smaller than 10 meters, then we have to brake. Already with such a simple rule, we can group together many states. In this case, we capture all triples where the last component is smaller than 10. Depending on the domain size of the velocities, this can already be hundreds or thousands of states.

*This idea of having "if-then-else"-rules that partition the state-space can be encoded well in **decision trees**. These are binary trees where the inner nodes are labelled by a predicate on the state-variables such as "$d < 10$"; and the leaf nodes are labelled by sets of actions such as "{brake}". A decision tree prescribes an action for a given state as follows: start at the root node, evaluate the predicate and go to the left or right child depending on whether the predicate is true or false. Repeat this process until you reach a leaf node, which gives you the set of allowed actions.* △

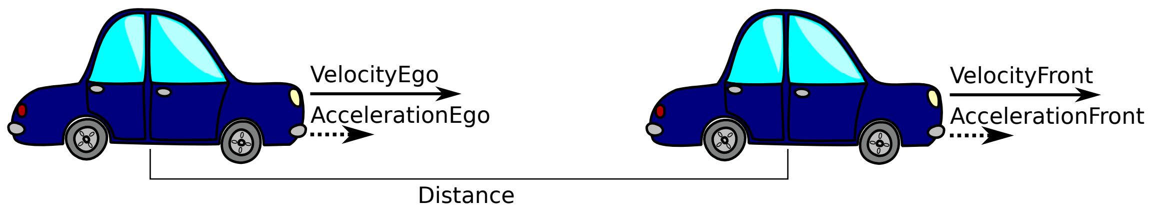


Figure 6.1: A graphical depiction of the case study used for our running example. Taken from [LMT15, Figure 1].

Definition 12 (Decision tree (DT)). *A decision tree (DT) is a full binary tree (every node has exactly* 0 *or* 2 *children) where every inner node is labelled with a predicate* $\mathsf{P} : \mathsf{S} \mapsto \{\mathit{true}, \mathit{false}\}$ *and every leaf node is labelled with a set of actions* 2^{A}*.*

This definition of DT is based on [Mit97], but already tailored to our setting. Constructing the optimal DT for representing a strategy is an NP-complete problem [HR76]. Hence, the following heuristic algorithm for DT construction is commonly used.

- Base case: If all states $s \in \mathsf{S}$ agree on their set of actions, i.e. we have $\sigma(s) = B$, then return a leaf node with label B.
- Recursive case: Otherwise, split the states. For this, select a predicate P and construct an inner node with label P. Then recursively construct a DT for the subset of states $\{s \in \mathsf{S} \mid \mathsf{P}(s)\}$ where the predicate is true and another DT for the subset where it is false. The resulting DTs are the children of the inner node with label P.

By exhaustively running this algorithm, we obtain a DT that exactly represents original strategy. Thus, any guarantee we have on the input strategy is preserved. For example, if the strategy keeps us in a safe set of states where we can avoid crashing into the car in front of us, it still does so when represented as DT. We mention that this usage of DTs for strategy representation is fundamentally different from the usual usage of DTs in machine learning, where a DT should not memorize the whole data it is given (called *overfitting*), but instead generalize to other problems.

Using the recursive algorithm, we can represent strategies more concisely and explainably. However, there are several questions to be addressed which greatly affect *how* concise and explainable the resulting DT is.

(a) What is the *domain* of the predicates?

(b) How do we *select* a predicate from this domain?

(c) How do we take advantage of the *nondeterminism* of the strategy?

There are state of the art answers to these questions, as well as extensions that are part of the contribution. We refrain from first answering the questions in this section and describing the extensions in the next, i.e. grouping the information according to how new it is. Instead, we group the information according to the questions first, and then inside the sections clearly mark what is new and old. We believe that this is more accessible and readable.

6.2 Contribution: dtControl 2.0

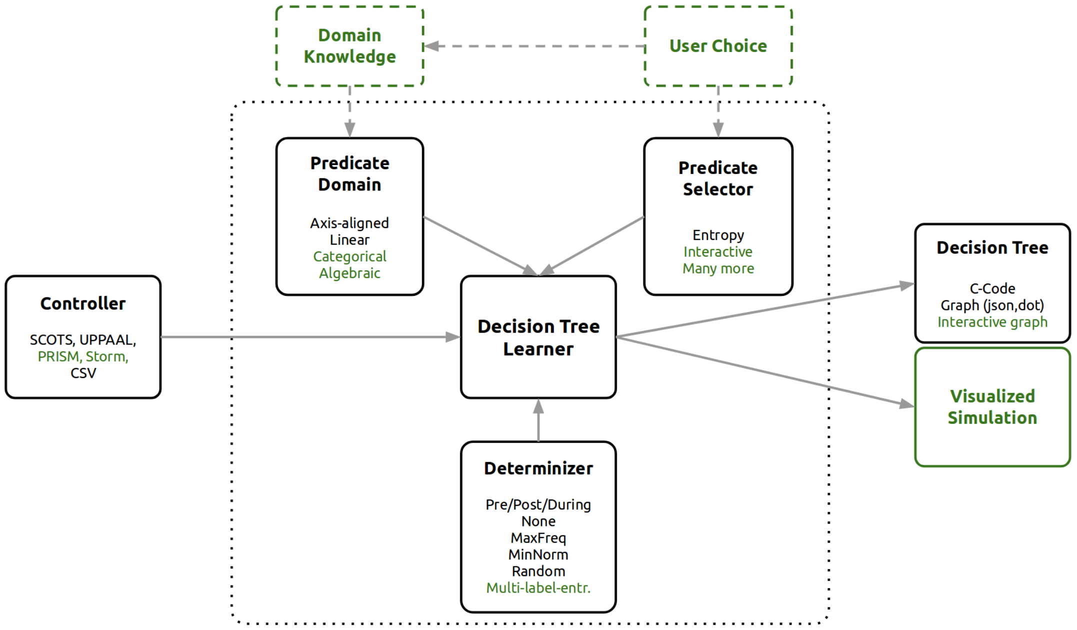


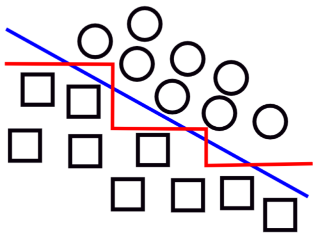
Figure 6.2: An overview of the components of the tool dtControl 2.0, showing software architecture and workflow. Everything depicted in black was already part of the first version of dtControl [AJJ+20a], namely the basic layout and some possibilities for the hyper-parameters. The green colour marks the features that are the contribution of the extension to dtControl 2.0. Based on [AJK+21a, Figure 2].

Figure 6.2 shows the architecture and workflow of our tool dtControl, which we developed to transform strategies (also called controllers) represented as list of state-action pairs to more concise and explainable DTs. It adheres to the intuition we just described in the state of the art section: an input strategy (controller) is given to the Decision Tree Learner of dtControl. We have to set the following hyper-parameters of the tool: The Predicate Domain (Question (a)), the Predicate Selector (Question (b)) and the Determinizer, which takes advantage of the nondeterminism of the strategy to reduce the size of the DT (Question (c)). Everything depicted in black was already part of the first version of dtControl [AJJ+20a], namely the basic layout and some choices for the hyper-parameters. The green colour marks the features that are the contribution of the extension to dtControl 2.0.

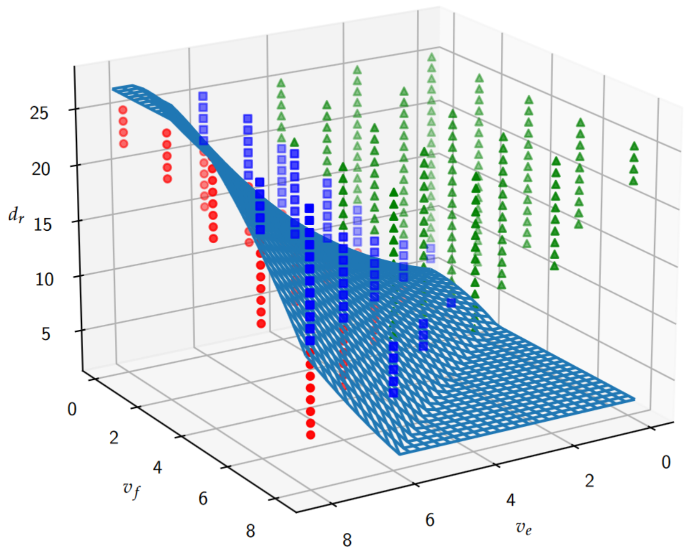
6.2.1 Predicate domain

We first illustrate that the predicate domain is very important for the size and explainability of the resulting DT by means of an example.

Example 19 (Illustrating more powerful predicates)**.** *Figure 6.3a illustrates a state-space with two state-variables, namely the horizontal and vertical coordinate of a data-point. A point being a circle or box indicates which action is allowed in a certain state. To represent*



(a)



(b)

Figure 6.3: (a) An illustration of the usefulness of linear predicates. Taken from [Jac20, Figure 5.2]. (b) An illustration of the usefulness of algebraic predicates on the real data from the car-controlling example. Taken from [Jün21, Figure 6.1].

this strategy using only axis-aligned splits, we would need 5 predicates, as indicated by the red line. In contrast, having one linear predicate suffices to completely partition the state-space.

However, in real examples, linear predicates often still are not powerful enough. Consider again the case study of controlling a car to keep a safe distance as described in Example 18. The kinematic equations governing the movement of objects under acceleration use quadratic terms. Hence, more powerful predicates are necessary to partition the state-space well. Figure 6.3b shows a part of the state-space of the case study. The colour of a data point indicates what set of actions is allowed: red corresponds to only braking, blue to braking or doing nothing and green to also allowing acceleration. The blue hyperplane illustrates a predicate perfectly splitting of all data-points where only braking is allowed. △

The example shows that different predicate domains impact the number of splits and hence the number of nodes in a DT. We now list the predicate domains supported by dtControl 2.0 and point out their advantages and disadvantages.

Axis-aligned predicates [Mit97] have the form $s_i \leq c$, where s_i is a state-variable (e.g. distance) and c is a rational constant. These predicates have the advantage that there are only finitely many, since the domain of every state-variable is finite.[2] Hence, they are also easy to compute, as we can just try out all of them. These predicates are the ones that are commonly used in machine learning, possibly because of their simplicity. However,

[2]Technically, there are finitely many *equivalence classes* of predicates. For example, if the domain of a state-variable s_i contains only the values 1 and 2, then $s_i \leq c$ are equivalent for all $c \in [1, 2)$.

they are also least expressive, since they can only use a single state-variable. The term axis-aligned comes from this fact, as in two dimensions, such a split always partitions the state-space along a line parallel to an axis of the underlying coordinate system, for example like the red lines in Figure 6.3a. In general, it is a hyperplane orthogonal to the axis of its corresponding state-variable.

Linear predicates (also known as oblique [MKSB93]) have the form $\sum_i s_i \cdot a_i \leq c$, where s_i and c are as before and a_i are rational coefficients. They have the advantage that they are able to combine several state-variables which can lead to saving linearly many splits, see Figure 6.3a. The disadvantage of these predicates is that it is highly non-trivial to find out the best coefficients a_i This is why heuristics were introduced to determine a good set of predicates to try out in [MKSB93, AJJ$^+$20a]. However, heuristically determined coefficients and combinations of variables can impede explainability.

Algebraic predicates (new feature of dtControl 2.0) have the form $f(s) \leq c$, where f is any mathematical function over the state-variables and c is a rational constant. It can use elementary functions such as exponentiation, logarithm or trigonometric functions. Coming up with these predicates currently requires domain knowledge and expert guidance. More concretely, a human provides a predicate template based on knowledge about the model at hand. In the car example, these templates are based on an understanding of the kinematic equations. Then dtControl 2.0 uses heuristics to find a good instantiation of the template. For more information on this process, we refer to [AJK$^+$21a, Section 4.2]. We highlight that, even though the predicate is complex, it is still explainable, as it is based on an actual understanding of the case study.

Example 20 (Reducing DT sizes with more powerful predicates)**.** *Consider again our car controlling running example where the strategy is a list of more than* 300, 000 *state-action pairs. Using only axis-aligned splits, dtControl returns a DT of size* 869*, which is already a reduction by three orders of magnitude. Linear predicates halve the size again, allowing to construct a DT with* 369 *nodes. While this is much more concise, it is still hardly explainable (possibly with a lot of manual work, assisted by the automated support described in Section 6.2.4). But with algebraic predicates, we are able to reduce the size to* 11 *nodes! Here, we can clearly identify the task of every predicate and explain its physical meaning, thus validating that the strategy indeed makes sense. See [Jün21, Chapter 4] for an explanation of the predicates.* △

Categorical predicates (new feature of dtControl 2.0) are special predicates for state-variables that do not have numerical domain (such as a velocity ranging over rationals), but rather a discrete, categorical domain. For example, a state-variable for colour can have values {red, blue, green}. Naively, one could treat these variables as numerical, assigning a (nonsensical) number to each value; or one could just make a full case distinction on all possible values of the domain. Our tool dtControl 2.0 in fact supports more sophisticated algorithms from literature for grouping similarly behaving values of the state-variable together, see [AJK$^+$21a, Section 4.1].

Categorical state-variables are common in case studies from the quantitative verification benchmark set [HKP$^+$19], and hence support for this benchmark set was only properly added in dtControl 2.0. Our evaluation of our DT construction on this benchmark set shows that the DT often is one or two orders of magnitudes smaller than the list of state-action pairs. Still, there also are case studies where in fact we do not reduce the size or where another representation, binary decision diagrams, resulted in a smaller (though not more explainable) representation. We refer to [AJK$^+$21a, Section 7] for more information.

6.2.2 Predicate selection

In general, predicate selection works as follows: given a set of predicates, we construct the resulting partitioning of the state-space for all of them. Then, we use an *impurity measure* to judge how similar the states in each of them are. If the strategy is the same for all states in one partition, its impurity is 0; on the other hand, if the strategy differs for all states in one partition, then the impurity is maximal. Thus, we want to use a predicate that minimizes impurity.

There are many impurity measures considered in literature, ranging from the well-known entropy introduced by Shannon [Sha48] to very particularly tuned measures such as the area under the receiver-operator curve [ABC$^+$19]. In its first version, dtControl supported only entropy. Now, in dtControl 2.0, we added many other measures, as detailed in [AJK$^+$21b, Appendix D]. However, our evaluation showed that entropy typically performs best.

Moreover, dtControl 2.0 has two new features affecting predicate selection which support experts in designing algebraic predicates: firstly, we offer the new functionality to assign **priorities** to the predicate generating algorithms. Priorities are rational numbers between 0 and 1. The impurity of every predicate is divided by the priority of the algorithm that generated it. For example, a user can use axis-aligned splits with priority 1 and a linear heuristic with priority $1/2$. Then the more complicated linear predicate is only chosen if it is at least twice as good (in terms of impurity) as the easier-to-understand axis-aligned split. Note that if a complicated predicate is able to split the data completely, its impurity is 0, and thus it is infinitely better than every predicate that does not completely separate the classes.

A predicate with priority 0 is only considered after all predicates with non-zero priority have failed to split the data. This allows the user to give just a few predicates from domain knowledge, which are then strictly preferred to the automatically generated ones, but which need not suffice to construct a complete DT for the controller.

Secondly, we added **interactive predicate selection**. This gives an expert full control over the DT generation, allowing to prefer predicates that are explainable over those that optimize the impurity. At all times, our graphical user interface shows some statistics and metadata, e.g. minimum, maximum and step size of the state-variables in the current node, a few automatically generated predicates for reference and all predicates generated from domain knowledge. The user can specify new predicates and is immediately informed about their impurity. Upon selecting a predicate, the split is performed and the user continues in the next node.

The user can also first construct a DT using some automatic algorithm and then restart the construction from an arbitrary node. Using the interactive predicate selection, the user can handcraft an optimized representation, or at any point decide that the rest of the DT should again be constructed automatically.

6.2.3 Determinizer

In this context of strategy representation, *determinizing* a strategy means picking a subset of the allowed actions for every state. Formally, determinizing a nondeterministic strategy $\sigma : \mathsf{S} \to 2^{\mathsf{A}}$ constructs a new (possibly still nondeterministic) strategy σ' such that for all states $s \in \mathsf{S}$ we have $\emptyset \subsetneq \sigma'(s) \subseteq \sigma(s)$. While this looses some information, the resulting controller still only selects optimal actions and thus preserves the guarantees of the strategy.[3] Moreover, it can drastically reduce the size of the strategy.

Example 21 (Determinizing our running example)**.** *Consider our running example of controlling a car. Since our only objective is safety — avoiding a state where the distance between the cars is too low, it is a valid strategy to "not drive". By picking the brake action in all states, we ensure that we cannot hit the front car. This reduces the size of the strategy to* 1 *node.*

Technically, in the formalisation of the case study, the brake action is not allowed when we already are at minimum velocity, since then we stay neutral. Hence, the resulting DT has 3 *nodes: the inner node asking whether we are at minimum velocity and two leaves, prescribing to stay neutral if we are and to brake if we are not.* △

We see that determinizing can greatly reduce the size of a controller, leading to a result even smaller than that we achieved using the algebraic predicates while still preserving guarantees. Note that, in this particular example, the strategy is definitely safe, but we actually have a second requirement, namely that we want to make progress when driving a car. This additional requirement is not encoded in the strategy and thus not satisfying it is not a fault of dtControl. If we want to satisfy this requirement as well, we have to options: firstly, we can analyse the trade-off between optimality and size by determinizing to varying degrees as in [AKL$^+$19]. Secondly, instead of only synthesizing a strategy for the safety objective, we could consider the following lexicographic multi-objective specification: first maximize safety and then maximize the travelled distance among the safe strategies. The resulting strategy will be less nondeterministic, but immediately more performant.

The determinization procedures in dtControl arose from practically trying around with the underlying ideas, which already led to very useful procedures. In [AJK$^+$21a, Section 6], we give a more theoretical treatment of the matter and from that derive an improved determinization procedure. We refer to [AJK$^+$21a, Section 6] and [AJK$^+$21b, Appendix E] for the technical details. Here, we only mention the results of the experimental evaluation [AJK$^+$21a, Section 7]: the new determinization procedure is strictly better and often produces DTs that are slightly smaller.

6.2.4 Further improvements of the user interface

We greatly improved the usability of our tool. While the first version of dtControl only had a command line interface, dtControl 2.0 additionally offers a graphical user interface.

[3]This is true when considering safety objectives. For some other objectives like reachability, it might lead to a violation of the original guarantees. For example, consider a strategy that allows both a self-looping and a non-self-looping action at a particular state. If the determinizer decides to restrict to the self-looping action, the reachability specification may be violated in the determinized strategy. However, this problem can be addressed when synthesizing the strategy by ensuring that every action makes progress towards the target.

This is extremely important, as the tool supports experts in coming up with algebraic predicates. For this, we offer interactive exploration of a DTs, the interactive predicate selection mentioned in Section 6.2.2 and a visualized simulation of a case study under the DT strategy, highlighting the path to the leaf node prescribing the action.

Moreover, we added support for the strategy files as output by the probabilistic model checkers Storm [DJKV17] and PRISM [KNP11]. Thus, additionally to the controller synthesis tools from hybrid system, Uppaal [LPY97] and Scots [RZ16], we now also support common model checkers for probabilistic verification.

Note that since the first version of dtControl we also offer to read a very straightforward csv-format. Thus, users of other tools can still use dtControl by transferring their strategy files into csv-format; this however requires writing their own scripts. Moreover, dtControl has a very modular structure and is easily extendible with parsers for other strategy formats, as was done for strategies resulting from the MODEST toolset [HH14] in the recent Bachelor's thesis [Wal21].

6.3 Related work

Decision trees for strategy-representation

The idea of representing a strategy more concisely while keeping its guarantees was already discussed in [Gir13], even suggesting a determinization procedure. However, the paper describes the data structure as "inspired by algebraic decision diagrams" [Gir13, Page 8], while actually using something more akin to a DT, and it lacks a proper construction algorithm, only stating that "we can split (typically using a hyperplane)" [Gir13, Page 8]. The actual ancestor of dtControl is the paper [BCC$^+$15], which for the first time explicitly used DTs for strategy-representation in the context of formal methods. There, the DT represented a *liberal* strategy and gave an ε-guarantee. This was addressed in [BCKT18], where the strategy was precisely stored in the DT in order to keep its full guarantees. On the one hand, this algorithm was extended to infinite-state safety games [NM19]. On the other hand, it was further tuned in [ABC$^+$19], where linear classifiers and new impurity measures were proposed; and in [AKL$^+$19], most notably adding a good heuristic for determinization. Finally, with the first version of dtControl [AJJ$^+$20a], a mature tool for strategy representation using DTs was developed. The tool is still under active development, often in the form Bachelor's theses [Akm19, Jac20, Wei20, Jün21]. Some of the major progress was reported in the paper summarized in this chapter [AJK$^+$21a]. Recently, a Bachelor's thesis [Wal21] extended dtControl with a parser for strategy files coming from the MODEST toolset [HH14]. This thesis was completely independent from all authors of the dtControl-papers, highlighting the modularity and ease-of-use of our tool.

Without the restriction of keeping guarantees, DTs have been used for strategy representation in artificial intelligence and machine learning, for example during reinforcement learning algorithms [CK91, PH98]. In particular, they were used in policy iteration [BDG95], dynamic programming [BD96, KP99] and reinforcement learning [KK99] with the additional assumption that the transition function of the considered system can be factored as a dynamic Bayesian network. Moreover, DTs were also used to represent strategies for Body Sensor Networks [LPTR10], with the main goal being a reduction in size to allow to store the strategies on embedded devices (Body Sensors).

Other uses of decision trees in formal methods and beyond

In [BCC+15], the motivation for concise and explainable representation was to explain counterexamples, i.e. strategies leading to unwanted behaviour. For further information on counterexample generation, see Section 1.5.4.

DTs were used to synthesize invariants [GNMR16] and piece-wise functions in [NSM16, END+18]. Note that the DT construction algorithm for the infinite safety games [NM19] builds on these methods.

DTs originated in the area of machine learning, in particular for classification tasks. For an introduction to that setting, we refer to [Mit97], and for a description of the relation between that setting and ours, we refer to [Jac20, Chapter 2 and Sections 3.2 and 5.1]. We refer to [Mur98] for a survey on the use of DTs in many different disciplines.

Alternative data structures for strategy representation

The main alternative for DTs are **binary decision diagrams (BDDs)** [Bry86], which also can be used to represent strategies as a graph of predicates on the state-variables. There are three major differences. (i) The underlying graph of BDDs is not a tree, but a directed acyclic graph. This allows them to reuse parts of the graph. (ii) The predicates of the BDD work on the bits of the input state-variables. This makes them easier to evaluate than those of a DT, but it is a big disadvantage for explainability. A predicate of the form "the third bit of distance is 0" is hardly explainable and longer chains of such predicates, mixing several state-variables, are not human-understandable. (iii) The leaf nodes of BDDs are binary. Hence, to encode a strategy as a BDD, we cannot store a map $\mathsf{S} \to 2^{\mathsf{A}}$ as for DTs, but rather have to store a map $\mathsf{S} \times 2^{\mathsf{A}} \to \{0, 1\}$.[4] This is not only harder to understand, but also more difficult to evaluate: instead of giving the current state to the strategy and getting an action, we have to give state-action pairs to the BDD until one of them is evaluated to 1. This can be mitigated by Algebraic Decision Diagrams (ADD) [BFG+97], also called multi-terminal BDD (MTBDD) [FMY97], or the more recent lattice-valued BDDs [GKG+10].

BDDs and MTBDDs are at the core of symbolic model checking algorithms, where they are used for representing both systems and strategies [BCM+92, CMCH96, dAKN+00, KNP04]. Moreover, they are used to represent strategies in planning [CRT98] or hybrid-systems controllers [JDT10, RZ16]. Inspired by the usage of DTs in [BDG95, BD96] for factored systems, [HSHB99, SHB00] use MTBDDs in reinforcement learning for strategy synthesis. Concise (but not explainable) strategy representation using BDDs was explicitly investigated in [PILM09, ZVJ18], where the latter even considers determinization procedures, similar to us. Note that both the problem of finding an optimal DT [HR76] or an optimal variable ordering for BDDs [BW96], are NP-complete, as is finding an optimal

[4]We mention that several paper using DTs, e.g. [BCC+15, BCKT18, NM19, ABC+19], represented the strategy as map to $\{0, 1\}$, even though that is suboptimal.

determinization for MTBDDs [ZVJ18]. We conjecture that the latter result also extends to DTs.

In our papers [AKL+19, AJJ+20a, AJK+21a], we always compared the DTs we generated with BDDs. For BDD-construction, we used reordering heuristics. To avoid reporting very suboptimal local minima, we additionally constructed several BDDs with randomly chosen initial variable orderings. Through all our papers [AKL+19, AJJ+20a, AJK+21a] as well as previous work [BCC+15, BCKT18], experimental results show that the BDDs typically contain more nodes than DTs. Moreover, they are sensitive to the choice of initial ordering [AKL+19, Table 1].

There are various other formalisms for representing **history-dependent strategies**, for example Mealy machines or AIGER circuits, as used by the tool Strix [MSL18]. Moreover, one can always employ more complex formalisms from machine learning, the most popular of which are neural networks. However, getting any kind of guarantees on the behaviour of the networks is very hard and a topic of active research [BP20, BP22]. Hence, they are not suitable for the task we consider, as we want our strategy representation to keep the guarantees. Additionally, neural networks are not explainable. There exist approaches to distil more explainable DTs from a neural network [FH17, AAHL20].

Explainability throughout many fields

The relevance of explainability in the formal methods community has risen in the past years. Many developments are summarized in the Isola track introduction "From Verification to Explanation" [BH20]. More recently, causality has been suggested as a metric for explaining systems [BFPZ22].

Looking to other fields, we see an immense amount of work done in artificial intelligence and machine learning, mostly under the keywords **eXplainable Artificial Intelligence (XAI)**. There are many recent surveys trying to categorize this huge body of work, see e.g. [GSC+19, ARS+20, TG21, GSC21, MBSG22] or the special issue [MHAH22]. Similarly, reliability of analysis of big data has been investigated [SFO19, SKK21]. In a highly cited paper [Rud19], Cynthia Rudin postulates replacing inexplicable models for high-stakes decision making with interpretable models.

Generally, our methods can no longer only focus on size and performance, but they also have to take into account our ability to understand their results. However, while size and performance are easy to quantify, it is difficult to come up with a clear metric for explainability, but it has been tried, see e.g. [BF16a]. We refer to [HDM+11] for an empirical evaluation of the comprehensibility of different representations and [PLGM16] for an extensive, in-depth study of classification-tree comprehensibility.

The fact that DT are explainable seems widely accepted. They are used for explaining neural networks [FH17, AAHL20], as already mentioned above. Further applications include intrusion detection systems [MTSS21], bioinformatics [CLRT11, SKPK12] or medicine [PKSR02]. Two groups of researchers developed a more explainable variant of DTs, called *cascading* DT [ZSP20, DHD+20]. While the aim and the name are similar, the resulting concept for cascading DT differs.

6.4 Summary, discussion and outlook

Summary We have presented dtControl 2.0. It is a mature, modular and performant tool used for representing memoryless nondeterministic strategies as DTs. The resulting representation is several orders of magnitude more concise and more explainable than the original list of state-action pairs.

The particular contributions of the appended paper [AJK$^+$21a] are, most importantly, (i) the addition of algebraic and categorical predicates, (ii) a graphical user interface which allows for exploring and interactively building DTs and (iii) a better determinization procedure, based on an improved theoretical understanding.

Discussion In the following, we discuss how dtControl already achieves its goals in certain ways and where we see the most room for improvement. Our first motivation was to represent strategies **concisely**, to fit them on embedded devices with limited memory; the size reduction by several orders of magnitude definitely is useful in this direction. We recall a related advantage from [AJJ$^+$20a, Section 6]: a better **quantization** of the state-space. In closed-loop systems, the information about the state of the system is measured by sensors, transferred over a channel to the controller which then decides the next action. Typically, these measurements are uniformly quantized, i.e. the state-space is discretized and partitioned into equally sized subsets. We can improve this by taking advantage of the fact that a DT effectively partitions the state-space according to the allowed actions. This partitioning groups many states together when compared to the uniform one. Thus, a sensor can use fewer bits to inform the controller about the state of the system. Concretely, in the example depicted in [AJJ$^+$20a, Figure 2 (b)], a uniform quantizer partitions the state-space into 271 different observations and hence uses 9 bits to inform the controller of the current state. By using dtControl instead, we see that we can partition the state-space into 6 observations, one for every possible leaf node of the DT. The sensor only has to inform the controller in which of these 6 sets the system currently is, and hence only 3 bits need to be transferred per time unit. Thus, dtControl can help in constructing efficient coder-controllers with a better-than-uniform quantization.

One more practical motivation of our approach was to allow **fast execution** of strategies. A DT can be encoded in a chain of if-then-else statements, and dtControl can output a DT as C-code. However, so far we have not run experiments to compare the memory footprint or execution time of this with BDDs or lookup tables. For BDDs, their memory footprints were investigated in [ZVJ18] and both memory and execution times in [PILM09]. Possibly, enabling dtControl to output DTs in a hardware description language could improve the resulting execution time.

Apart from conciseness, our second motivation was **explainability**. In this direction, we can report two success stories: firstly, we were able to detect a bug in the model of our running example. This shows that investigating the resulting strategy is useful for validating the model and that such validation is necessary.

Secondly, engineers designing a bin-picking robot used dtControl to debug the PRISM model of their robot, as reported in [KGAK22]. Essentially, this turned the modelling into a counter-example guided loop, where the engineer specified the PRISM model, represented the resulting strategy as a DT, analysed it and then updated the model according to unwanted behaviour he saw in the strategy. Compressing the strategies with millions of state-action pairs to DTs with thousands of nodes allowed the engineer to detect many mistakes in the model.

This success story highlights again that model validation is necessary, and additionally shows that even DTs of very large size are understandable enough to be useful for experts. It also supports the argument that requiring expert guidance for the development of algebraic predicates is an acceptable restriction.

It is interesting to consider the **theoretical limits** of representing strategies as DTs. Finding an optimal DT is NP-complete [HR76]. We now give a limit on the minimum size of DTs [Jün21]: consider a strategy that recommends n different action sets, where n potentially is exponential in the number of actions. Without determinizing, a DT for this strategy needs at least $2n - 1$ nodes, since it is a binary tree with n leaf nodes. This is independent of whether the tree is a balanced binary tree or a "stairway", separating one label completely with every predicate.[5] Finding the best possible determinization is probably NP-complete because it is NP-complete for MTBDDs [ZVJ18]. As a trade-off between computing the best determinization (which is infeasible) and not determinizing, we use heuristics. We can further be inspired by methods from machine learning which try to keep as many actions as possible[Rea08, TKV08]. Moreover, we could consider liberal strategies with ε-guarantees as in [BCC+15], or reduce the size of the strategy by only representing its *core* [KM20].

Outlook There are many practical improvements that we plan for dtControl. Firstly, we want to improve the generation of algebraic predicates by

- improving the user interface to support experts better in guiding the generation of algebraic predicates.
- automating the generation process, only requiring basic domain knowledge and not suggestions of predicates. In this direction, we can report a negative result: starting with the Kinematics equations and considering well-typed combinations of them, coming up with the "right" predicate requires so many combinations of basic expressions that the whole search space is infeasibly large [Jün21, Chapter 5]. However, using sophisticated machine learning and prettifying could help [Jün21, Chapter 6], even though it is at the risk of generating predicates which are too complex to be explained.

There are many further heuristics we can try in order to construct smaller DTs.

- Using a decision diagram, i.e. reusing subtrees instead of copying them. This seems like it would mimic an advantage of BDDs. However, our preliminary analysis showed that this seldom helps in our case studies.
- Using lookahead during the DT construction, as in [EM03, BCKT18].

[5]Note that, depending on the situation, either of these could be more explainable. If there is a natural way of finding a stairway, this is easy to follow. However, the predicates required for this form might be too complex.

- Using evolutionary algorithms [BBCF12] which perform a global search and can avoid local minima. However, to retain safety guarantees we would have to take further measures, as these are not included in these algorithms so far. Still, the DTs constructed this way could serve as heuristics, inspiring the predicate generation or tree structure.
- Adding an approach based on uniform random sampling of the space of all DTs. This requires a way to sample uniformly from the space of all predicates. Given that, such an approach can be quite performant and provide statistical guarantees.
- Improve the performance of the DT construction by parallelising. For example, checking the entropy of all considered predicates is completely independent and can be parallelised.

To judge the effectiveness of our heuristics, it could be worthwhile to implement an approach for finding the optimal determinization and optimal DT. Most probably this will not scale well and thus we will only be able to compare on small or medium case studies.

We encourage the reader to visit our website `https://dtcontrol.model.in.tum.de/`, which contains not only the tool (as an easy-to-install pip-package or source code) but also documentation and tutorials. We hope that dtControl will help users in many different fields to represent their strategies concisely and explainably.

7 Application: Defending against advanced persistent threats

Motivation The motivation for this chapter is threefold.

- We improve previous techniques for the concrete security problem of defending against advanced persistent threats.
- We show that it can be useful to modify the formalisms which are used to model a problem. In particular, turn-based games are easier to analyse than concurrent ones and multi-objective analysis is more informative than merging multiple objectives into a single utility function.
- We combine several algorithms from previous chapters, showcasing their usefulness and the interplay between them.

Problem statement We want to compute the optimal strategies of an attacker and defender in a setting where dynamic information flow tracking is used to defend against an advanced persistent threat.

From previous work, we know that we can analyse advanced persistent threats using concurrent stochastic games with a complex utility function and limited information.

We want to modify this formalisation so that we can equivalently analyse a turn-based stochastic game (SG) with multiple, but simpler objectives. Then we can apply algorithms from Chapter 4 and 5 to compute Pareto-frontiers and optimal strategies.

Chapter outline We first recall the state of the art solution in Section 7.1: what are advanced persistent threats and how do we model the defence against them as stochastic game. Then we summarize how we modified the formalism to facilitate easier and more meaningful analysis in Section 7.2, provide and algorithm and experimentally evaluate our approach. This is based on our paper [WGMK21] (Appendix H). Finally, we discuss related work in Section 7.3 and conclude in Section 7.4.

7.1 State of the art

Advanced persistent threats (APT), see e.g. [GP$^+$14], are a type of attack where an attacker gains illegitimate access to a system and remains undetected for a long period of time. APTs have emerged as a serious security threat for many organizations and industries, including national defense, manufacturing, and finances [Col12, VG13, LCMF18].

While these attacks are difficult to detect, there still exist ways to do it. One way relies on the fact that APTs create information flows which are recorded in the system log file. *Dynamic Information Flow Tracking (DIFT)* [SLZD04] is used to analyse this recorded information and detect APTs. DIFT operates by tagging suspicious input/output data channels, tracing the propagation of the tagged information through the system and then performing security analysis on these flows to detect attacks. However, this introduces memory and performance costs on the system, especially when it involves tracking and analysing a large number of innocuous flows [JLD+17].

Several papers [MSC+18, SXC+18, MMH+19, SMB+19, MSA+20] have modelled the problem of defending against APTs using DIFT as a concurrent game between two players, Attacker and Defender. The Attacker tries to gain access to secret information in the system, while the Defender decides which flows to track. Defender has several partially contradicting objectives: detecting the attacker, forcing the Attacker to abort the attack, preventing the Attacker from accessing secret information and minimizing the cost on the system. We refer to [MSA+20, Section II and III] for an extensive description of the resulting game model. Throughout the papers, the exact game model has varied, as we describe in Section 7.3. Our work builds on [MMH+19], where the problem is modelled as an acyclic concurrent non-zero-sum SG with limited information. For a formal definition of this kind of system, see [WGMK21, Definition 2].

We believe that this formalisation is suboptimal, for several reasons.

- Concurrent games are hard to analyse because even when only considering reachability, optimal strategies do not exists and ε-optimal strategies require randomization [dAHK07]. Thus, we want to model our system as turn-based game, if possible.
- All of the mentioned works all use a single-dimensional reward function that encodes several objectives. This is suboptimal, as the outcome of the analysis greatly depends on the exact choice of utilities assigned to each objective. Thus, instead of relying on a good choice of these numbers, we want to explicitly model the problem with a multi-dimensional reward function. This more interpretable formulation allows us to answer questions such as "What cost does the defender need to pay to achieve the maximum probability of trapping the attacker?", "What is the highest probability to trap the attacker by a defence cheaper than 10?" or what the possible trade-offs are.

  Note that this way of encoding the objectives also makes them *zero-sum*, i.e. the objectives of the players are exactly contrary to each other. This allows us to avoid the more involved solution concept of Nash-equilibria; instead, we compute the set of all achievable vectors, as in Chapter 5.
- Previous solutions often relied on machine learning techniques [MMH+19, SMA+19a] which are not reliable. Thus, we want to use an algorithm that gives guarantees on its result. As we are in a setting with limited information, we want to provide a *probably approximately correct (PAC) guarantee.*

Moreover, the analysis in [MMH+19] was limited to acyclic (hence stopping) SGs.

7.2 Contribution: Guaranteed trade-offs in dynamic information flow tracking games

Our contribution is twofold: firstly, we prove that we can model the problem as a turn-based SG with a multi-dimensional objective function. Secondly, we show how to solve these games in the presence of limited information, namely unknown transition probabilities. We obtain an approximation of the set of achievable vectors.

7.2.1 Adjusting the formalisation

There are two parts to adjusting the formalisation of the problem: the concurrency of the system and the incomprehensible, utility-based objective function.

To transform the **concurrent system** into a turn-based one, we used the fact that the SGs modelling a DIFT defense against an APT attack have a very particular structure. In every state (except for a few absorbing special states), the Defender has the choice to try to trap the Attacker or not. If the Defender does not trap or if the trapping was unsuccessful, the Attacker decides the next state of the system. If the trapping was successful, a special absorbing state is reached, which models that the Attacker was detected.

The Defender's choice of whether to trap or not does not affect the Attacker's strategy. Intuitively, Attacker was aware of the risk of being detected when entering the current state and has no way of affecting the probability of being detected in the next step. Thus, instead of having the players make their choice concurrently, we can first let Defender decide whether to trap or not and afterwards let the Attacker decide where to move. Thus, we can transform a concurrent SG modelling this specific problem into a turn-based one. We prove this formally in [WGMK21, Appendix II].

For the **objective-function**, we observe that there are four distinct objectives that the players have.

- Destination: Attacker wants to reach a destination, thus obtaining secret information. Defender wants to avoid this.
- Detection: Defender wants to detect the Attacker. Attacker wants to avoid this.
- Dropout: Defender wants Attacker to drop out of the system and abort the attack. Attacker wants to avoid this. Note that, when considering utilities, this state is better for Attacker than being detected, and we have to numerically specify how much better. Using a multi-objective function, we can separate the concerns.
- Cost: Defender wants to minimize the cost that the DIFT incurs. Attacker is either indifferent to this or malicious and wants to maximize it.

We see that all these objectives are in fact zero-sum. They were only non-zero-sum in previous work because they were combined into a single utility function.

Also, we see that it is possible to track each objective separately and not merge the payoffs into a single function. Our solution concept thus is not a single value, but a vector of values. To obtain a single function as previously, we can choose a weight for each component, effectively specifying a ratio between them; this was called *direction* in Chapter 5. These weights are effectively set in [MMH$^+$19, Section IV] by choosing $\alpha_k, \beta_k, \sigma_k$ etc. Thus, our analysis indeed not only solves the previous problem, but additionally provides more information, since it informs us about all possible Pareto-optimal outcomes, not just one outcome for a particular ratio.

Table 7.1: Properties of the considered case studies and runtime of our algorithm. Based on [WGMK21, Table 1].

Example	Size	Cyclicity	Time taken (s)
Random	10	No	7.78
	100	No	11.90
Random	10	Yes	8.29
	100	Yes	17.63
ScreenGrab	9	no	7.95
NationState	30	yes	8.40

7.2.2 Algorithm

Our algorithm proceeds in three steps: first it simulates the model to learn the unknown probabilities with a certain confidence, using a grey-box variant of the algorithm in Chapter 4.1. Then, we transform the resulting bounded-parameter SG to a normal SG using the general approach described in Section 4.2. Finally, we use the algorithm from [CFK$^+$13b] to compute the Pareto-frontier in the multi-objective SG we obtained. For more details, we refer to [WGMK21, Section IV] or the respective sections of this dissertation.

Note all of the algorithms we use do not restrict the considered SG to be acyclic, so we have also lifted this requirement of [MMH$^+$19].

7.2.3 Experimental results

We have implemented our algorithm and ran it on two real case studies as well as several random graphs. The results are summarized in Table 7.1. We see that the algorithm runs in reasonable time on the real case studies, finishing within a few seconds. Moreover, even on the larger random models it terminates quickly, giving hope that it will also scale to larger problems when they occur in practice.

Further, we analysed the resulting Pareto-frontiers, see [WGMK21, Figure 4]. As the models were very small, we were able to validate that the computed values and trade-offs make sense. Moreover, we gathered the following insights about the objective function.

- Defender cannot ensure that Attacker aborts the attack. Since we are interested in defending against a worst-case Attacker, we have to assume the Attacker is risk-seeking and prefers chances of incurring costs and reaching the destination over a safe dropout. Thus, we can omit this dimension from the analysis.

- The probability to detect the Attacker is 1 minus the probability of the Attacker to reach the goal. Again, a worst-case Attacker is risk-seeking, so the only chance of preventing the destination being reached is to detect the Attacker.

7.3 Related work

We mention that the related works in Sections 4.1.5, 4.2.4 and 5.3 are relevant for this chapter, too. This is because the algorithms we use here are combinations of those from previous chapters. In the following, we recall developments in the analysis of the concrete security problem of APTs and defending against them using DIFT.

Advanced persistent threats

APTs pose a huge risk for companies and governments alike, stealing terabytes of sensitive data and causing immense financial loss [VG13, GP+14, LCMF18]. Thus, even though successful large-scale APTs have only started being reported on in the past decade, APTs have received a lot of attention, see the book [Col12], the surveys [VG13, GP+14, CDH14, LCMF18, AMCH19] and the special issue [CSYY18]. These reports analyse many attacks that have occurred in the past 15 years and discuss why they were possible, what they achieved and what has to be changed in order to defend against them. Their conclusions are alarming: "the old way of doing security is no longer going to scale against a game-changing adversary, the APT." [Col12, Page xv] and "[n]o single current protection approach alone can efficiently defeat APT, and thus research effort is required to further investigate this area" [CSYY18, Page 1].

Dynamic information flow tracking

Since APTs are so sophisticated and varied, protection against them cannot rely on traditional security methods anymore. One necessary defence mechanism is to monitor the entire network and look for suspicious activity, as described in [VG13, GP+14, AMCH19] or [Col12, Page 23].

DIFT as a way to monitor spurious information flows was introduced in [SLZD04]. It has been improved since then, for example by using efficient parallelization [JKKP13] or by investigating the trade-off between collecting a detailed log and minimizing runtime overhead [JLD+17].

Based on this, a string of works modelled the interaction of an APT attack and a DIFT defence using games. This approach of using games is common in security, see e.g. [LW05, RES+10]. In [HZ18, SXC+18, MSC+18], the problem is modelled as a *non-stochastic* concurrent game with imperfect information. Multiple attackers were considered in [SMA+19b], and an alternative model of a zero-sum Stackelberg semi-Markov game in [SAM+20]. Concurrent *stochastic* games were used in [SMB+19, SMA+19a, MMH+19, MSA+20]. Many of these papers use a single utility function to capture the objective, often making the game non-zero-sum.

7.4 Summary, discussion and outlook

Summary It is important to defend against APTs and one way of doing so is by using DIFT. We have shown how to modify the formalism for modelling this problem so that it is (i) not a concurrent game, but an easier-to-analyse turn-based one; (ii) not using a single, incomprehensible utility function, but a multi-dimensional objective allowing for analysing trade-offs and (iii) we have combined our algorithms to give a guaranteed solution for the resulting model.

Discussion Over the past few years, many models have been proposed for modelling APT versus DIFT games. We do not claim to have found an optimum, but we do believe that our contribution can help this string of research. In particular, as discussed in Section 7.2.3, we can essentially ignore two dimensions of the objective function.

In general, we believe it is relevant to expand the reach of formal methods. We as a field should bring our guaranteed algorithms and our knowledge of modelling formalisms to the communities that are practically solving problems, like the security community in this chapter. Even though from the algorithmic side, the contribution of this chapter is a straightforward combination of existing algorithms, it requires knowledge of and experience with these algorithms and the involved formalisms.

Outlook The algorithms we concatenate to achieve our results are in fact suboptimal: we only get a guarantee of convergence in the limit for non-stopping SGs, since we rely on [CFK$^+$13b, Theorem 4] for solving the turn-based SG with a combination of reachability and reward objectives. While this was not relevant in the case studies, it is still desirable to extend the contribution of Chapter 5 to also provide an ε-guarantee on reward objectives. This would allow to giving an anytime guarantee, as well as a PAC guarantee on non-stopping SGs.

In general, we hope to repeat the story of this paper [WGMK21]: find a practical problem, improve the formalism that is used to model it and apply algorithms from formal methods to compute solutions that are more informative and reliable.

8 Conclusion

We have given an overview of the state of the art techniques for solving stochastic games, with a particular focus on doing so reliably. This reliability comes in the form of giving guarantees on the precision of our algorithms (Chapter 3), more expressive formalisms for modelling the given problem (Chapter 4 and Chapter 5) and allowing to validate a model (Chapter 6).

Our overview has taken us on a tour through all parts of the model checking paradigm, investigating which formalisms can be used to model the system (Chapter 4) and the specification (Chapter 5), how to use and validate the result (Chapter 6) and how to apply this to a real case study, in particular by choosing the right formalism (Chapter 7). We have encountered problems of theoretical interest, e.g. whether there exists a polynomial algorithm for solving simple stochastic games or whether generalized-reachability stochastic games are decidable; and very practical problems, e.g. how to improve the scalability of our methods by using statistical model checking or machine learning approaches like simulation-based algorithms and decision trees. All chapters include an extensive overview of related work, surveying the current literature and thus also giving many potential directions for future work.

In conclusion, I hope that this dissertation serves as a good introduction to the topics I encountered during my PhD, as well as an inspiration for further research. The overall goal is to help agents — people — make good decisions reliably.

Bibliography

[AAHL20] Parand Alizadeh Alamdari, Guy Avni, Thomas A. Henzinger, and Anna Lukina. Formal methods with a touch of magic. In *FMCAD*, pages 138–147. IEEE, 2020. doi:10.34727/2020/isbn.978-3-85448-042-6_21.

[AAK15] Shaull Almagor, Guy Avni, and Orna Kupferman. Repairing multi-player games. In *CONCUR*, volume 42 of *LIPIcs*, pages 325–339. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CONCUR.2015.325.

[AAK17] Rimmi Anand, Divya Aggarwal, and Vijay Kumar. A comparative analysis of optimization solvers. *Journal of Statistics and Management Systems*, 20(4):623–635, 2017. doi:10.1080/09720510.2017.1395182.

[Aar16] Scott Aaronson. P =? NP. In John Forbes Nash Jr. and Michael Th. Rassias, editors, *Open Problems in Mathematics*, pages 1–122. Springer, 2016. doi:10.1007/978-3-319-32162-2_1.

[AB] Scott Aaronson and Chris Bourke. The complexity zoo. URL `http://cse.unl.edu/~cbourke/latex/ComplexityZoo.pdf`. Accessed: 15.06.2022.

[ABC$^+$19] Pranav Ashok, Tomás Brázdil, Krishnendu Chatterjee, Jan Kretínský, Christoph H. Lampert, and Viktor Toman. Strategy representation by decision trees with linear classifiers. In *QEST*, volume 11785 of *Lecture Notes in Computer Science*, pages 109–128. Springer, 2019. doi:10.1007/978-3-030-30281-8_7.

[ÁBD$^+$14] Erika Ábrahám, Bernd Becker, Christian Dehnert, Nils Jansen, Joost-Pieter Katoen, and Ralf Wimmer. Counterexample generation for discrete-time Markov models: An introductory survey. In *SFM*, volume 8483 of *Lecture Notes in Computer Science*, pages 65–121. Springer, 2014. doi:10.1007/978-3-319-07317-0_3.

[ABD$^+$15] Rajeev Alur, Rastislav Bodík, Eric Dallal, Dana Fisman, Pranav Garg, Garvit Juniwal, Hadas Kress-Gazit, P. Madhusudan, Milo M. K. Martin, Mukund Raghothaman, Shambwaditya Saha, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *Dependable Software Systems Engineering*, volume 40 of *NATO Science for Peace and Security Series, D: Information and Communication Security*, pages 1–25. IOS Press, 2015. doi:10.3233/978-1-61499-495-4-1.

[ABE$^+$18] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *AAAI*, pages 2669–2678. AAAI Press, 2018. URL `https://ojs.aaai.org/index.php/AAAI/article/view/11797`.

[ABKS18] Pranav Ashok, Tomás Brázdil, Jan Kretínský, and Ondrej Slámecka. Monte carlo tree search for verifying reachability in Markov decision processes. In *ISoLA (2)*, volume 11245 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 2018. doi:10.1007/978-3-030-03421-4_21.

[ACD$^+$17] Pranav Ashok, Krishnendu Chatterjee, Przemyslaw Daca, Jan Kretínský, and Tobias Meggendorfer. Value iteration for long-run average reward in Markov decision processes. In *CAV (1)*, volume 10426 of *Lecture Notes in Computer Science*, pages 201–221. Springer, 2017. doi:10.1007/978-3-319-63387-9_10.

[ACK$^+$20] Pranav Ashok, Krishnendu Chatterjee, Jan Kretinsky, Maximilian Weininger, and Tobias Winkler. Approximating values of generalized-reachability stochastic games. In *LICS*, pages 102–115. ACM, 2020. doi:10.1145/3373718.3394761.

[ACS19] David Auger, Pierre Coucheney, and Yann Strozecki. Solving simple stochastic games with few random nodes faster using Bland's rule. In *STACS*, volume 126 of *LIPIcs*, pages 9:1–9:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.9.

[AD14] Matthias Althoff and John M. Dolan. Online verification of automated road vehicles using reachability analysis. *IEEE Trans. Robotics*, 30(4):903–918, 2014. doi:10.1109/TRO.2014.2312453.

[ADKW20] Pranav Ashok, Przemyslaw Daca, Jan Kretinsky, and Maximilian Weininger. Statistical model checking: Black or white? In *ISoLA (1)*, volume 12476 of *Lecture Notes in Computer Science*, pages 331–349. Springer, 2020. doi:10.1007/978-3-030-61362-4_19.

[AdMS21] David Auger, Xavier Badin de Montjoye, and Yann Strozecki. A generic strategy improvement method for simple stochastic games. In *MFCS*, volume 202 of *LIPIcs*, pages 12:1–12:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.12.

[AH20] Guy Avni and Thomas A. Henzinger. A survey of bidding games on graphs (invited paper). In *CONCUR*, volume 171 of *LIPIcs*, pages 2:1–2:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.2.

[AHdA$^+$08] Parosh Aziz Abdulla, Noomene Ben Henda, Luca de Alfaro, Richard Mayr, and Sven Sandberg. Stochastic games with lossy channels. In *FoSSaCS*, volume 4962 of *Lecture Notes in Computer Science*, pages 35–49. Springer, 2008. doi:10.1007/978-3-540-78499-9_4.

[AJJ+20a] Pranav Ashok, Mathias Jackermeier, Pushpak Jagtap, Jan Kretínský, Maximilian Weininger, and Majid Zamani. dtControl: decision tree learning algorithms for controller representation. In *HSCC*, pages 17:1–17:7. ACM, 2020. doi:10.1145/3365365.3382220.

[AJJ+20b] Pranav Ashok, Mathias Jackermeier, Pushpak Jagtap, Jan Kretínský, Maximilian Weininger, and Majid Zamani. dtControl: decision tree learning algorithms for controller representation. In *HSCC*, pages 30:1–30:2. ACM, 2020. doi:10.1145/3365365.3383468.

[AJK+21a] Pranav Ashok, Mathias Jackermeier, Jan Kretinsky, Christoph Weinhuber, Maximilian Weininger, and Mayank Yadav. dtControl 2.0: Explainable strategy representation via decision tree learning steered by experts. In *TACAS (2)*, volume 12652 of *Lecture Notes in Computer Science*, pages 326–345. Springer, 2021. doi:10.1007/978-3-030-72013-1_17.

[AJK+21b] Pranav Ashok, Mathias Jackermeier, Jan Kretínský, Christoph Weinhuber, Maximilian Weininger, and Mayank Yadav. dtControl 2.0: Explainable strategy representation via decision tree learning steered by experts. *CoRR*, abs/2101.07202, 2021. URL `https://arxiv.org/abs/2101.07202`.

[AK20] Shaull Almagor and Orna Kupferman. Good-enough synthesis. In *CAV (2)*, volume 12225 of *Lecture Notes in Computer Science*, pages 541–563. Springer, 2020. doi:10.1007/978-3-030-53291-8_28.

[AKL+19] Pranav Ashok, Jan Kretínský, Kim Guldstrand Larsen, Adrien Le Coënt, Jakob Haahr Taankvist, and Maximilian Weininger. SOS: safe, optimal and small strategies for hybrid Markov decision processes. In *QEST*, volume 11785 of *Lecture Notes in Computer Science*, pages 147–164. Springer, 2019. doi:10.1007/978-3-030-30281-8_9.

[Akm19] Safa Mert Akmese. *Generating Richer Predicates for Decision Trees.* Bachelor's thesis, Technical University of Munich, 2019.

[AKV16] Shaull Almagor, Orna Kupferman, and Yaron Velner. Minimizing expected cost under hard boolean constraints, with applications to quantitative synthesis. In *CONCUR*, volume 59 of *LIPIcs*, pages 9:1–9:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.9.

[AKW19a] Pranav Ashok, Jan Kretinsky, and Maximilian Weininger. PAC statistical model checking for Markov decision processes and stochastic games. In *CAV (1)*, volume 11561 of *Lecture Notes in Computer Science*, pages 497–519. Springer, 2019. doi:10.1007/978-3-030-25540-4_29.

[AKW19b] Pranav Ashok, Jan Kretínský, and Maximilian Weininger. PAC statistical model checking for Markov decision processes and stochastic games. *CoRR*, abs/1905.04403, 2019. URL `http://arxiv.org/abs/1905.04403`.

[AL09] Husain Aljazzar and Stefan Leue. Generation of counterexamples for model checking of Markov decision processes. In *QEST*, pages 197–206. IEEE Computer Society, 2009. doi:10.1109/QEST.2009.10.

[Alu15] Rajeev Alur. *Principles of cyber-physical systems.* MIT press, 2015. URL `https://mitpress.mit.edu/books/principles-cyber-physical-systems`.

[AM09] Daniel Andersson and Peter Bro Miltersen. The complexity of solving stochastic games on graphs. In *ISAAC*, volume 5878 of *Lecture Notes in Computer Science*, pages 112–121. Springer, 2009. doi:10.1007/978-3-642-10631-6_13.

[AMCH19] Adel Alshamrani, Sowmya Myneni, Ankur Chowdhary, and Dijiang Huang. A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities. *IEEE Commun. Surv. Tutorials*, 21(2):1851–1877, 2019. doi:10.1109/COMST.2019.2891891.

[Ami03] Rabah Amir. Stochastic games in economics and related fields: an overview. *Stochastic games and applications*, pages 455–470, 2003. doi:10.1007/978-94-010-0189-2_30.

[AP18] Gul Agha and Karl Palmskog. A survey of statistical model checking. *ACM Trans. Model. Comput. Simul.*, 28(1):6:1–6:39, 2018. doi:10.1145/3158668.

[ARS$^+$20] Alejandro Barredo Arrieta, Natalia Díaz Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. *Inf. Fusion*, 58:82–115, 2020. doi:10.1016/j.inffus.2019.12.012.

[Åst65] Karl Johan Åström. Optimal control of Markov processes with incomplete state information. *Journal of mathematical analysis and applications*, 10(1):174–205, 1965. URL `https://core.ac.uk/download/pdf/82498456.pdf`.

[AY17] Gürdal Arslan and Serdar Yüksel. Decentralized Q-learning for stochastic teams and games. *IEEE Trans. Autom. Control.*, 62(4):1545–1558, 2017. URL `https://doi.org/10.1109/TAC.2016.2598476`.

[Bai15] Christel Baier. Reasoning about cost-utility constraints in probabilistic models. In *RP*, volume 9328 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 2015. doi:10.1007/978-3-319-24537-9_1.

[BBB$^+$12] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Delahaye, and Axel Legay. Statistical abstraction and model-checking of large heterogeneous systems. *Int. J. Softw. Tools Technol. Transf.*, 14(1):53–72, 2012. doi:10.1007/s10009-011-0201-2.

[BBC$^+$14] Tomás Brázdil, Václav Brozek, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. Two views on multiple mean-payoff objectives in Markov decision processes. *Log. Methods Comput. Sci.*, 10(1), 2014. doi:10.2168/LMCS-10(1:13)2014.

[BBCF12] Rodrigo Coelho Barros, Márcio Porto Basgalupp, André Carlos Ponce de Leon Ferreira de Carvalho, and Alex Alves Freitas. A survey of evolutionary algorithms for decision-tree induction. *IEEE Trans. Syst. Man Cybern. Part C*, 42(3):291–312, 2012. doi:10.1109/TSMCC.2011.2157494.

[BBGT21] Thomas Brihaye, Véronique Bruyère, Aline Goeminne, and Nathan Thomasset. On relevant equilibria in reachability games. *J. Comput. Syst. Sci.*, 119:211–230, 2021. doi:10.1016/j.jcss.2021.02.009.

[BBS08] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Trans. Syst. Man Cybern. Part C*, 38(2):156–172, 2008. URL `https://doi.org/10.1109/TSMCC.2007.913919`.

[BCC$^+$14] Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Kretínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of markov decision processes using learning algorithms. In *ATVA*, volume 8837 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2014. doi:10.1007/978-3-319-11936-6_8.

[BCC$^+$15] Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Andreas Fellner, and Jan Kretínský. Counterexample explanation by learning small strategies in Markov decision processes. In *CAV (1)*, volume 9206 of *Lecture Notes in Computer Science*, pages 158–177. Springer, 2015. doi:10.1007/978-3-319-21690-4_10.

[BCD$^+$18] Christel Baier, Philipp Chrszon, Clemens Dubslaff, Joachim Klein, and Sascha Klüppelholz. Energy-utility analysis of probabilistic systems with exogenous coordination. In *It's All About Coordination*, volume 10865 of *Lecture Notes in Computer Science*, pages 38–56. Springer, 2018. doi:10.1007/978-3-319-90089-6_3.

[BCFK15] Tomás Brázdil, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. Multigain: A controller synthesis tool for MDPs with multiple mean-payoff objectives. In *TACAS*, volume 9035 of *Lecture Notes in Computer Science*, pages 181–187. Springer, 2015. doi:10.1007/978-3-662-46681-0_12.

[BCFK17] Tomás Brázdil, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. Trading performance for stability in Markov decision processes. *J. Comput. Syst. Sci.*, 84:144–170, 2017. doi:10.1016/j.jcss.2016.09.009.

[BCH$^+$16] Romain Brenguier, Lorenzo Clemente, Paul Hunter, Guillermo A. Pérez, Mickael Randour, Jean-François Raskin, Ocan Sankur, and Mathieu Sassolas.

Non-zero sum games for reactive synthesis. In *LATA*, volume 9618 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2016. doi:10.1007/978-3-319-30000-9_1.

[BCKT18] Tomás Brázdil, Krishnendu Chatterjee, Jan Kretínský, and Viktor Toman. Strategy representation by decision trees in reactive synthesis. In *TACAS (1)*, volume 10805 of *Lecture Notes in Computer Science*, pages 385–407. Springer, 2018. doi:10.1007/978-3-319-89960-2_21.

[BCLS13] Benoît Boyer, Kevin Corre, Axel Legay, and Sean Sedwards. Plasma-lab: A flexible, distributable statistical model checking library. In *QEST*, volume 8054 of *Lecture Notes in Computer Science*, pages 160–164. Springer, 2013. doi:10.1007/978-3-642-40196-1_12.

[BCM$^+$92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10ˆ20 states and beyond. *Inf. Comput.*, 98(2):142–170, 1992. doi:10.1016/0890-5401(92)90017-A.

[BCV21] Suguman Bansal, Krishnendu Chatterjee, and Moshe Y. Vardi. On satisficing in quantitative games. In *TACAS (1)*, volume 12651 of *Lecture Notes in Computer Science*, pages 20–37. Springer, 2021. doi:10.1007/978-3-030-72016-2_2.

[BD96] Craig Boutilier and Richard Dearden. Approximate value trees in structured dynamic programming. In *ICML*, pages 54–62. Morgan Kaufmann, 1996. URL `https://www.semanticscholar.org/paper/Approximate-Value-Trees-in-Structured-Dynamic-Boutilier-Dearden/ddb3bd5136cea13652a5eca9e286956844d440ac`.

[BD08] Nicola Bellomo and M Delitala. From the mathematical kinetic, and stochastic game theory to modelling mutations, onset, progression and immune competition of cancer cells. *Physics of Life Reviews*, 5(4):183–206, 2008. doi:10.1016/j.plrev.2008.07.001.

[BD18] Christel Baier and Clemens Dubslaff. From verification to synthesis under cost-utility constraints. *ACM SIGLOG News*, 5(4):26–46, 2018. doi:10.1145/3292048.3292052.

[BDD$^+$14] Christel Baier, Marcus Daum, Clemens Dubslaff, Joachim Klein, and Sascha Klüppelholz. Energy-utility quantiles. In *NASA Formal Methods*, volume 8430 of *Lecture Notes in Computer Science*, pages 285–299. Springer, 2014. doi:10.1007/978-3-319-06200-6_24.

[BDG95] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Exploiting structure in policy construction. In *IJCAI*, pages 1104–1113. Morgan Kaufmann, 1995. URL `http://ijcai.org/Proceedings/95-2/Papers/012.pdf`.

[BDK14a] Christel Baier, Clemens Dubslaff, and Sascha Klüppelholz. Trade-off analysis meets probabilistic model checking. In *CSL-LICS*, pages 1:1–1:10. ACM, 2014. doi:10.1145/2603088.2603089.

[BDK$^+$14b] Christel Baier, Clemens Dubslaff, Sascha Klüppelholz, Marcus Daum, Joachim Klein, Steffen Märcker, and Sascha Wunderlich. Probabilistic model checking and non-standard multi-objective reasoning. In *FASE*, volume 8411 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2014. doi:10.1007/978-3-642-54804-8_1.

[BDL$^+$12] Peter E. Bulychev, Alexandre David, Kim Guldstrand Larsen, Marius Mikucionis, Danny Bøgsted Poulsen, Axel Legay, and Zheng Wang. UPPAAL-SMC: statistical model checking for priced timed automata. In *QAPL*, volume 85 of *EPTCS*, pages 1–16, 2012. doi:10.4204/EPTCS.85.1.

[BDL$^+$17] Anicet Bart, Benoît Delahaye, Didier Lime, Éric Monfroy, and Charlotte Truchet. Reachability in parametric interval Markov chains using constraints. In *QEST*, volume 10503 of *Lecture Notes in Computer Science*, pages 173–189. Springer, 2017. doi:10.1007/978-3-319-66335-7_11.

[BDMS08] Jürgen Branke, Kalyanmoy Deb, Kaisa Miettinen, and Roman Slowinski, editors. *Multiobjective Optimization, Interactive and Evolutionary Approaches [outcome of Dagstuhl seminars]*, volume 5252 of *Lecture Notes in Computer Science*. Springer, 2008. doi:10.1007/978-3-540-88908-3.

[BF16a] Adrien Bibal and Benoît Frénay. Interpretability of machine learning models and representations: an introduction. In *ESANN*, 2016. URL `https://www.semanticscholar.org/paper/Interpretability-of-machine-learning-models-and-an-Bibal-Fr%C3%A9nay/464656fc6431f1db8b2e0b0b3093a5df1cb7958e`.

[BF16b] Romain Brenguier and Vojtech Forejt. Decidability results for multi-objective stochastic games. In *ATVA*, volume 9938 of *Lecture Notes in Computer Science*, pages 227–243, 2016. doi:10.1007/978-3-319-46520-3_15.

[BFG$^+$97] R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Algebraic decision diagrams and their applications. *Formal Methods Syst. Des.*, 10(2/3):171–206, 1997. doi:10.1023/A:1008699807402.

[BFPZ22] Christel Baier, Florian Funke, Jakob Piribauer, and Robin Ziemek. On probability-raising causality in Markov decision processes. In *FoSSaCS*, volume 13242 of *Lecture Notes in Computer Science*, pages 40–60. Springer, 2022. doi:10.1007/978-3-030-99253-8_3.

[BFRR17] Véronique Bruyère, Emmanuel Filiot, Mickael Randour, and Jean-François Raskin. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. *Inf. Comput.*, 254:259–295, 2017. doi:10.1016/j.ic.2016.10.011.

[BGJS20] Hugo Bazille, Blaise Genest, Cyrille Jégourel, and Jun Sun. Global PAC bounds for learning discrete time Markov chains. In *CAV (2)*, volume 12225 of *Lecture Notes in Computer Science*, pages 304–326. Springer, 2020. doi:10.1007/978-3-030-53291-8_17.

[BGMR18] Patricia Bouyer, Mauricio González, Nicolas Markey, and Mickael Randour. Multi-weighted Markov decision processes with reachability objectives. In *GandALF*, volume 277 of *EPTCS*, pages 250–264, 2018. doi:10.4204/EPTCS.277.18.

[BGR20] Raphaël Berthon, Shibashis Guha, and Jean-François Raskin. Mixing probabilistic and non-probabilistic objectives in Markov decision processes. In *LICS*, pages 195–208. ACM, 2020. doi:10.1145/3373718.3394805.

[BH20] Christel Baier and Holger Hermanns. From verification to explanation (track introduction). In *ISoLA (4)*, volume 12479 of *Lecture Notes in Computer Science*, pages 1–7. Springer, 2020. doi:10.1007/978-3-030-83723-5_1.

[BHH12] Jonathan Bogdoll, Arnd Hartmanns, and Holger Hermanns. Simulation and statistical model checking for modestly nondeterministic models. In *MMB/DFT*, volume 7201 of *Lecture Notes in Computer Science*, pages 249–252. Springer, 2012. doi:10.1007/978-3-642-28540-0_20.

[BHK19] Christel Baier, Holger Hermanns, and Joost-Pieter Katoen. The 10, 000 facets of MDP model checking. In *Computing and Software Science*, volume 10000 of *Lecture Notes in Computer Science*, pages 420–451. Springer, 2019. doi:10.1007/978-3-319-91908-9_21.

[BHK$^+$20] Carlos E. Budde, Arnd Hartmanns, Michaela Klauck, Jan Kretínský, David Parker, Tim Quatmann, Andrea Turrini, and Zhen Zhang. On correctness, precision, and performance in quantitative verification - QComp 2020 competition report. In *ISoLA (4)*, volume 12479 of *Lecture Notes in Computer Science*, pages 216–241. Springer, 2020. doi:10.1007/978-3-030-83723-5_15.

[BHL19] Giovanni Bacci, Mikkel Hansen, and Kim Guldstrand Larsen. Model checking constrained Markov reward models with uncertainties. In *QEST*, volume 11785 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 2019. doi:10.1007/978-3-030-30281-8_3.

[BHR16] Véronique Bruyère, Quentin Hautem, and Jean-François Raskin. On the complexity of heterogeneous multidimensional games. In *CONCUR*, volume 59 of *LIPIcs*, pages 11:1–11:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.11.

[BHRR19] Véronique Bruyère, Quentin Hautem, Mickael Randour, and Jean-François Raskin. Energy mean-payoff games. In *CONCUR*, volume 140 of *LIPIcs*, pages 21:1–21:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.21.

[BJW02] Julien Bernet, David Janin, and Igor Walukiewicz. Permissive strategies: from parity games to safety games. *RAIRO Theor. Informatics Appl.*, 36(3):261–275, 2002. doi:10.1051/ita:2002013.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008. URL `https://mitpress.mit.edu/books/principles-model-checking`.

[BKL$^+$17] Christel Baier, Joachim Klein, Linda Leuschner, David Parker, and Sascha Wunderlich. Ensuring the reliability of your model checker: Interval iteration for markov decision processes. In *CAV (1)*, volume 10426 of *Lecture Notes in Computer Science*, pages 160–180. Springer, 2017. doi:10.1007/978-3-319-63387-9_8.

[BKN$^+$19] Nikhil Balaji, Stefan Kiefer, Petr Novotný, Guillermo A. Pérez, and Mahsa Shirmohammadi. On the complexity of value iteration. In *ICALP*, volume 132 of *LIPIcs*, pages 102:1–102:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.102.

[BKTW15] Nicolas Basset, Marta Z. Kwiatkowska, Ufuk Topcu, and Clemens Wiltsche. Strategy synthesis for stochastic games with multiple long-run objectives. In *TACAS*, volume 9035 of *Lecture Notes in Computer Science*, pages 256–271. Springer, 2015. doi:10.1007/978-3-662-46681-0_22.

[BKW18] Nicolas Basset, Marta Z. Kwiatkowska, and Clemens Wiltsche. Compositional strategy synthesis for stochastic games with multiple objectives. *Inf. Comput.*, 261:536–587, 2018. doi:10.1016/j.ic.2017.09.010.

[BLW13] Michael Benedikt, Rastislav Lenhardt, and James Worrell. LTL model checking of interval Markov chains. In *TACAS*, volume 7795 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2013. doi:10.1007/978-3-642-36742-7_3.

[BMOU11] Patricia Bouyer, Nicolas Markey, Jörg Olschewski, and Michael Ummels. Measuring permissiveness in parity games: Mean-payoff parity games revisited. In *ATVA*, volume 6996 of *Lecture Notes in Computer Science*, pages 135–149. Springer, 2011. doi:10.1007/978-3-642-24372-1_11.

[BP20] Edoardo Bacci and David Parker. Probabilistic guarantees for safe deep reinforcement learning. In *FORMATS*, volume 12288 of *Lecture Notes in Computer Science*, pages 231–248. Springer, 2020. doi:10.1007/978-3-030-57628-8_14.

[BP22] Edoardo Bacci and David Parker. Verified probabilistic policies for deep reinforcement learning. In *NFM*, volume 13260 of *Lecture Notes in Computer Science*, pages 193–212. Springer, 2022. doi:10.1007/978-3-031-06773-0_10.

[BR15] Romain Brenguier and Jean-François Raskin. Pareto curves of multidimensional mean-payoff games. In *CAV (2)*, volume 9207 of *Lecture Notes in*

Computer Science, pages 251–267. Springer, 2015. doi:10.1007/978-3-319-21668-3_15.

[Bre16] Joachim Breitner. Visual theorem proving with the incredible proof machine. In *ITP*, volume 9807 of *Lecture Notes in Computer Science*, pages 123–139. Springer, 2016. doi:10.1007/978-3-319-43144-4_8.

[BRR17] Raphaël Berthon, Mickael Randour, and Jean-François Raskin. Threshold constraints with guarantees for parity objectives in Markov decision processes. In *ICALP*, volume 80 of *LIPIcs*, pages 121:1–121:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.121.

[BRT21] Véronique Bruyère, Jean-François Raskin, and Clément Tamines. Stackelberg-pareto synthesis. In *CONCUR*, volume 203 of *LIPIcs*, pages 27:1–27:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CONCUR.2021.27.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986. doi:10.1109/TC.1986.1676819.

[BT00] Ronen I. Brafman and Moshe Tennenholtz. A near-optimal polynomial time algorithm for learning in certain classes of stochastic games. *Artif. Intell.*, 121(1-2):31–47, 2000. doi:10.1016/S0004-3702(00)00039-4.

[Buf05] Olivier Buffet. Reachability analysis for uncertain SSPs. In *ICTAI*, pages 515–522. IEEE Computer Society, 2005. doi:10.1109/ICTAI.2005.106.

[BW96] Beate Bollig and Ingo Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Trans. Computers*, 45(9):993–1002, 1996. doi:10.1109/12.537122.

[CCK$^+$20] Krishnendu Chatterjee, Martin Chmelík, Deep Karkhanis, Petr Novotný, and Amélie Royer. Multiple-environment Markov decision processes: Efficient analysis and applications. In *ICAPS*, pages 48–56. AAAI Press, 2020. URL `https://ojs.aaai.org/index.php/ICAPS/article/view/6644`.

[CD11] Krishnendu Chatterjee and Laurent Doyen. Energy and mean-payoff parity Markov decision processes. In *MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 206–218. Springer, 2011. doi:10.1007/978-3-642-22993-0_21.

[CD16] Krishnendu Chatterjee and Laurent Doyen. Perfect-information stochastic games with generalized mean-payoff objectives. In *LICS*, pages 247–256. ACM, 2016. doi:10.1145/2933575.2934513.

[CdAH13] Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. Strategy improvement for concurrent reachability and turn-based stochastic safety games. *J. Comput. Syst. Sci.*, 79(5):640–657, 2013. doi:10.1016/j.jcss.2012.12.001.

[CdAHS03] Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource interfaces. In *EMSOFT*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 2003. doi:10.1007/978-3-540-45212-6_9.

[CDGO14] Krishnendu Chatterjee, Laurent Doyen, Hugo Gimbert, and Youssouf Oualhadj. Perfect-information stochastic mean-payoff parity games. In *FoSSaCS*, volume 8412 of *Lecture Notes in Computer Science*, pages 210–225. Springer, 2014. doi:10.1007/978-3-642-54830-7_14.

[CDH14] Ping Chen, Lieven Desmet, and Christophe Huygens. A study on advanced persistent threats. In *CMS*, volume 8735 of *Lecture Notes in Computer Science*, pages 63–72. Springer, 2014. URL `https://doi.org/10.1007/978-3-662-44885-4_5`.

[CDHR10] Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Generalized mean-payoff and energy games. In *FSTTCS*, volume 8 of *LIPIcs*, pages 505–516. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.505.

[CDL$^+$11] Benoît Caillaud, Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. Constraint Markov chains. *Theor. Comput. Sci.*, 412(34):4373–4404, 2011. doi:10.1016/j.tcs.2011.05.010.

[CDP$^+$17] Milan Ceska, Frits Dannenberg, Nicola Paoletti, Marta Kwiatkowska, and Lubos Brim. Precise parameter synthesis for stochastic biochemical systems. *Acta Informatica*, 54(6):589–623, 2017. doi:10.1007/s00236-016-0265-2.

[CDRR15] Krishnendu Chatterjee, Laurent Doyen, Mickael Randour, and Jean-François Raskin. Looking at mean-payoff and total-payoff through windows. *Inf. Comput.*, 242:25–52, 2015. doi:10.1016/j.ic.2015.03.010.

[CF11] Krishnendu Chatterjee and Nathanaël Fijalkow. A reduction from parity games to simple stochastic games. In *GandALF*, volume 54 of *EPTCS*, pages 74–86, 2011. doi:10.4204/EPTCS.54.6.

[CFK$^+$13a] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, David Parker, and Aistis Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods Syst. Des.*, 43(1):61–92, 2013. doi:10.1007/s10703-013-0183-7.

[CFK$^+$13b] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. On stochastic games with multiple objectives. In *MFCS*, volume 8087 of *Lecture Notes in Computer Science*, pages 266–277. Springer, 2013. doi:10.1007/978-3-642-40313-2_25.

[CFW13] Krishnendu Chatterjee, Vojtech Forejt, and Dominik Wojtczak. Multi-objective discounted reward verification in graphs and MDPs. In *LPAR*, volume 8312 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 2013. doi:10.1007/978-3-642-45221-5_17.

[CGNS19] Tristan Charrier, Sébastien Gamblin, Alexandre Niveau, and François Schwarzentruber. Hintikka's world: Scalable higher-order knowledge. In *IJCAI*, pages 6494–6496. ijcai.org, 2019. doi:10.24963/ijcai.2019/934.

[CH06] Krishnendu Chatterjee and Thomas A. Henzinger. Strategy improvement for stochastic Rabin and Streett games. In *CONCUR*, volume 4137 of *Lecture Notes in Computer Science*, pages 375–389. Springer, 2006. doi:10.1007/11817949_25.

[CH08] Krishnendu Chatterjee and Thomas A. Henzinger. Value iteration. In *25 Years of Model Checking*, volume 5000 of *Lecture Notes in Computer Science*, pages 107–138. Springer, 2008. doi:10.1007/978-3-540-69850-0_7.

[CH12] Krishnendu Chatterjee and Thomas A. Henzinger. A survey of stochastic ω-regular games. *J. Comput. Syst. Sci.*, 78(2):394–413, 2012. doi:10.1016/j.jcss.2011.05.002.

[CH14] Krishnendu Chatterjee and Monika Henzinger. Efficient and dynamic algorithms for alternating büchi games and maximal end-component decomposition. *J. ACM*, 61(3):15:1–15:40, 2014. doi:10.1145/2597631.

[Cha07a] Krishnendu Chatterjee. Markov decision processes with multiple long-run average objectives. In *FSTTCS*, volume 4855 of *Lecture Notes in Computer Science*, pages 473–484. Springer, 2007. doi:10.1007/978-3-540-77050-3_39.

[Cha07b] Krishnendu Chatterjee. *Stochastic ω-regular games*. PhD thesis, University of California, Berkeley, 2007. URL `https://digitalassets.lib.berkeley.edu/techreports/ucb/text/EECS-2007-122.pdf`.

[Che15] Jianhua Chen. Properties of a new adaptive sampling method with applications to scalable learning. *Web Intell.*, 13(4):215–227, 2015. doi:10.3233/WEB-150322.

[CHO20] Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Multidimensional long-run average problems for vector addition systems with states. In *CONCUR*, volume 171 of *LIPIcs*, pages 23:1–23:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.23.

[CHVB18] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018. doi:10.1007/978-3-319-10575-8.

[CJJ$^+$20] Murat Cubuktepe, Nils Jansen, Sebastian Junges, Joost-Pieter Katoen, and Ufuk Topcu. Scenario-based verification of uncertain MDPs. In *TACAS (1)*, volume 12078 of *Lecture Notes in Computer Science*, pages 287–305. Springer, 2020. doi:10.1007/978-3-030-45190-5_16.

[CK91] David Chapman and Leslie Pack Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *IJCAI*, pages 726–731. Morgan Kaufmann, 1991. URL `http://ijcai.org/Proceedings/91-2/Papers/018.pdf`.

[CK15] Souymodip Chakraborty and Joost-Pieter Katoen. Model checking of open interval Markov chains. In *ASMTA*, volume 9081 of *Lecture Notes in Computer Science*, pages 30–42. Springer, 2015. doi:10.1007/978-3-319-18579-8_3.

[CKK17] Krishnendu Chatterjee, Zuzana Kretínská, and Jan Kretínský. Unifying two views on multiple mean-payoff objectives in Markov decision processes. *Log. Methods Comput. Sci.*, 13(2), 2017. doi:10.23638/LMCS-13(2:15)2017.

[CKSW13] Taolue Chen, Marta Z. Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. Synthesis for multi-objective stochastic games: An application to autonomous urban driving. In *QEST*, volume 8054 of *Lecture Notes in Computer Science*, pages 322–337. Springer, 2013. doi:10.1007/978-3-642-40196-1_28.

[CKWW20] Krishnendu Chatterjee, Joost-Pieter Katoen, Maximilian Weininger, and Tobias Winkler. Stochastic games with lexicographic reachability-safety objectives. In *CAV (2)*, volume 12225 of *Lecture Notes in Computer Science*, pages 398–420. Springer, 2020. doi:10.1007/978-3-030-53291-8_21.

[CLRT11] Dongsheng Che, Qi Liu, Khaled Rasheed, and Xiuping Tao. Decision tree and ensemble learning algorithms with their applications in bioinformatics. *Software tools and algorithms for biological systems*, pages 191–199, 2011. URL `https://link.springer.com/chapter/10.1007/978-1-4419-7046-6_19`.

[CMCH96] Edmund M. Clarke, Kenneth L. McMillan, Sérgio Vale Aguiar Campos, and Vasiliki Hartonas-Garmhausen. Symbolic model checking. In *CAV*, volume 1102 of *Lecture Notes in Computer Science*, pages 419–427. Springer, 1996. doi:10.1007/3-540-61474-5_93.

[CMG14] Javier Cámara, Gabriel A. Moreno, and David Garlan. Stochastic game analysis and latency awareness for proactive self-adaptation. In *SEAMS*, pages 155–164. ACM, 2014. doi:10.1145/2593929.2593933.

[CMH06] Krishnendu Chatterjee, Rupak Majumdar, and Thomas A. Henzinger. Markov decision processes with multiple objectives. In *STACS*, volume 3884 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 2006. doi:10.1007/11672142_26.

[Col12] Eric Cole. *Advanced persistent threat: understanding the danger and how to protect your organization.* Newnes, 2012. URL `https://books.google.de/books?id=Y-CQmN5sEg8C&pg=PR7&lpg=PR1&ots=Hk_qVVjEWx&focus=viewport&dq=cole+advanced+persistent+threat&lr=&hl=de`.

[Con92] Anne Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992. doi:10.1016/0890-5401(92)90048-K.

[Con93] Anne Condon. On algorithms for simple stochastic games. In *Advances In Computational Complexity Theory*, volume 13 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 51–73. DIMACS/AMS, 1993. doi:10.1090/dimacs/013/04.

[CR15] Lorenzo Clemente and Jean-François Raskin. Multidimensional beyond worst-case and almost-sure problems for mean-payoff objectives. In *LICS*, pages 257–268. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.33.

[CRR14] Krishnendu Chatterjee, Mickael Randour, and Jean-François Raskin. Strategy synthesis for multi-dimensional quantitative objectives. *Acta Informatica*, 51(3-4):129–163, 2014. doi:10.1007/s00236-013-0182-6.

[CRT98] Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *AAAI/IAAI*, pages 875–881. AAAI Press / The MIT Press, 1998. URL `http://www.aaai.org/Library/AAAI/1998/aaai98-124.php`.

[CSH08] Krishnendu Chatterjee, Koushik Sen, and Thomas A. Henzinger. Model-checking omega-regular properties of interval Markov chains. In *FoSSaCS*, volume 4962 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 2008. doi:10.1007/978-3-540-78499-9_22.

[CSYY18] Jiageng Chen, Chunhua Su, Kuo-Hui Yeh, and Moti Yung. Special issue on advanced persistent threat. *Future Gener. Comput. Syst.*, 79:243–246, 2018. doi:10.1016/j.future.2017.11.005.

[CY95] Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995. doi:10.1145/210332.210339.

[CZ11] Edmund M. Clarke and Paolo Zuliani. Statistical model checking for cyber-physical systems. In *ATVA*, volume 6996 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2011. doi:10.1007/978-3-642-24372-1_1.

[DA98] Luca De Alfaro. *Formal verification of probabilistic systems.* PhD thesis, Stanford university, 1998. URL `https://exhibits.stanford.edu/cs/catalog/xz681dy7227`.

[dAHK07] Luca de Alfaro, Thomas A. Henzinger, and Orna Kupferman. Concurrent reachability games. *Theor. Comput. Sci.*, 386(3):188–217, 2007. doi:10.1016/j.tcs.2007.07.008.

[dAKN$^+$00] Luca de Alfaro, Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Roberto Segala. Symbolic model checking of probabilistic processes using MTBDDs and the Kronecker representation. In *TACAS*, volume 1785 of *Lecture Notes in Computer Science*, pages 395–410. Springer, 2000. doi:10.1007/3-540-46419-0_27.

[Daw04] Conrado Daws. Symbolic and parametric model checking of discrete-time Markov chains. In *ICTAC*, volume 3407 of *Lecture Notes in Computer Science*, pages 280–294. Springer, 2004. doi:10.1007/978-3-540-31862-0_21.

[DC18] Maxence Dutreix and Samuel Coogan. Satisfiability bounds for co-regular properties in interval-valued Markov chains. In *CDC*, pages 1047–1052. IEEE, 2018. doi:10.1109/CDC.2018.8619756.

[DDG$^+$10] Aldric Degorre, Laurent Doyen, Raffaella Gentilini, Jean-François Raskin, and Szymon Torunczyk. Energy and mean-payoff games with imperfect information. In *CSL*, volume 6247 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2010. doi:10.1007/978-3-642-15205-4_22.

[DDL$^+$12] Alexandre David, Dehui Du, Kim G. Larsen, Axel Legay, Marius Mikucionis, Danny Bøgsted Poulsen, and Sean Sedwards. Statistical model checking for stochastic hybrid systems. In *HSB*, volume 92 of *EPTCS*, pages 122–136, 2012. doi:10.4204/EPTCS.92.9.

[DDL$^+$13] Alexandre David, Dehui Du, Kim Guldstrand Larsen, Axel Legay, and Marius Mikucionis. Optimizing control strategy using statistical model checking. In *NFM*, volume 7871 of *Lecture Notes in Computer Science*, pages 352–367. Springer, 2013. doi:10.1007/978-3-642-38088-4_24.

[DFK$^+$15] Klaus Dräger, Vojtech Forejt, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Permissive controller synthesis for probabilistic systems. *Log. Methods Comput. Sci.*, 11(2), 2015. doi:10.2168/LMCS-11(2:16)2015.

[DG11] Decheng Dai and Rong Ge. Another sub-exponential algorithm for the simple stochastic game. *Algorithmica*, 61(4):1092–1104, 2011. doi:10.1007/s00453-010-9413-1.

[DHD$^+$20] Zihan Ding, Pablo Hernandez-Leal, Gavin Weiguang Ding, Changjian Li, and Ruitong Huang. CDT: cascading decision trees for explainable reinforcement learning. *CoRR*, abs/2011.07553, 2020. URL `https://arxiv.org/abs/2011.07553`.

[DHK$^+$20] Loris D'Antoni, Martin Helfrich, Jan Kretínský, Emanuel Ramneantu, and Maximilian Weininger. Automata tutor v3. In *CAV (2)*, volume 12225 of *Lecture Notes in Computer Science*, pages 3–14. Springer, 2020. doi:10.1007/978-3-030-53291-8_1.

[DHKP17] Przemyslaw Daca, Thomas A. Henzinger, Jan Kretínský, and Tatjana Petrov. Faster statistical model checking for unbounded temporal properties. *ACM Trans. Comput. Log.*, 18(2):12:1–12:25, 2017. doi:10.1145/3060139.

[DHS18] Pedro R. D'Argenio, Arnd Hartmanns, and Sean Sedwards. Lightweight statistical model checking in nondeterministic continuous time. In *ISoLA (2)*, volume 11245 of *Lecture Notes in Computer Science*, pages 336–353. Springer, 2018. doi:10.1007/978-3-030-03421-4_22.

[DJKV17] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A Storm is coming: A modern probabilistic model checker. In *CAV (2)*, volume 10427 of *Lecture Notes in Computer Science*, pages 592–600. Springer, 2017. doi:10.1007/978-3-319-63390-9_31.

[DJL$^+$15] Alexandre David, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marius Mikucionis, and Jakob Haahr Taankvist. UPPAAL STRATEGO. In *TACAS*, volume 9035 of *Lecture Notes in Computer Science*, pages 206–211. Springer, 2015. doi:10.1007/978-3-662-46681-0_16.

[DJW$^+$14] Christian Dehnert, Nils Jansen, Ralf Wimmer, Erika Ábrahám, and Joost-Pieter Katoen. Fast debugging of PRISM models. In *ATVA*, volume 8837 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2014. doi:10.1007/978-3-319-11936-6_11.

[DKQR20] Florent Delgrange, Joost-Pieter Katoen, Tim Quatmann, and Mickael Randour. Simple strategies in multi-objective MDPs. In *TACAS (1)*, volume 12078 of *Lecture Notes in Computer Science*, pages 346–364. Springer, 2020. doi:10.1007/978-3-030-45190-5_19.

[DLL$^+$11a] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, Danny Bøgsted Poulsen, Jonas van Vliet, and Zheng Wang. Statistical model checking for networks of priced timed automata. In *FORMATS*, volume 6919 of *Lecture Notes in Computer Science*, pages 80–96. Springer, 2011. doi:10.1007/978-3-642-24310-3_7.

[DLL$^+$11b] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Zheng Wang. Time for statistical model checking of real-time systems. In *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 349–355. Springer, 2011. doi:10.1007/978-3-642-22110-1_27.

[DLST15] Pedro R. D'Argenio, Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Smart sampling for lightweight verification of Markov decision processes. *Int. J. Softw. Tools Technol. Transf.*, 17(4):469–484, 2015. doi:10.1007/s10009-015-0383-0.

[dM21] Xavier Badin de Montjoye. A recursive algorithm for solving simple stochastic games. *CoRR*, abs/2110.01030, 2021. URL `https://arxiv.org/abs/2110.01030`.

[DMWG11] Peng Dai, Mausam, Daniel S. Weld, and Judy Goldsmith. Topological value iteration algorithms. *J. Artif. Intell. Res.*, 42:181–209, 2011. URL `https://www.jair.org/index.php/jair/article/view/10725`.

[DR11] Laurent Doyen and Jean-François Raskin. Games with imperfect information: theory and algorithms. In *Lectures in Game Theory for Computer Scientists*, pages 185–212. Cambridge University Press, 2011. URL `http://www.lsv.fr/~doyen/papers/Games_with_Imperfect_Information_Theory_Algorithms.pdf`.

[EGF12] Christian Ellen, Sebastian Gerwinn, and Martin Fränzle. Confidence bounds for statistical model checking of probabilistic hybrid systems. In *FORMATS*, volume 7595 of *Lecture Notes in Computer Science*, pages 123–138. Springer, 2012. doi:10.1007/978-3-642-33365-1_10.

[EGF15] Christian Ellen, Sebastian Gerwinn, and Martin Fränzle. Statistical model checking for stochastic hybrid systems involving nondeterminism over continuous domains. *Int. J. Softw. Tools Technol. Transf.*, 17(4):485–504, 2015. doi:10.1007/s10009-014-0329-y.

[EHL17] Arno Ehle, Norbert Hundeshagen, and Martin Lange. The sequent calculus trainer with automated reasoning - helping students to find proofs. In *ThEdu@CADE*, volume 267 of *EPTCS*, pages 19–37, 2017. doi:10.4204/EPTCS.267.2.

[EKKW21] Javier Esparza, Stefan Kiefer, Jan Kretínský, and Maximilian Weininger. Enforcing ω-regular properties in Markov chains by restarting. In *CONCUR*, volume 203 of *LIPIcs*, pages 5:1–5:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CONCUR.2021.5.

[EKKW22] Julia Eisentraut, Edon Kelmendi, Jan Kretínský, and Maximilian Weininger. Value iteration for simple stochastic games: Stopping criterion and learning algorithm. *Inf. Comput.*, 285(Part):104886, 2022. doi:10.1016/j.ic.2022.104886.

[EKR19] Julia Eisentraut, Jan Kretínský, and Alexej Rotar. Stopping criteria for value and strategy iteration on concurrent stochastic reachability games. *CoRR*, abs/1909.08348, 2019. URL `http://arxiv.org/abs/1909.08348`.

[EKVY08] Kousha Etessami, Marta Z. Kwiatkowska, Moshe Y. Vardi, and Mihalis Yannakakis. Multi-objective model checking of Markov decision processes. *Log. Methods Comput. Sci.*, 4(4), 2008. doi:10.2168/LMCS-4(4:8)2008.

[EM03] Tapio Elomaa and Tuomo Malinen. On lookahead heuristics in decision tree learning. In *ISMIS*, volume 2871 of *Lecture Notes in Computer Science*, pages 445–453. Springer, 2003. doi:10.1007/978-3-540-39592-8_63.

[END$^+$18] P. Ezudheen, Daniel Neider, Deepak D'Souza, Pranav Garg, and P. Madhusudan. Horn-ICE learning for synthesizing invariants and contracts. *Proc. ACM Program. Lang.*, 2(OOPSLA):131:1–131:25, 2018. doi:10.1145/3276501.

[EY15] Kousha Etessami and Mihalis Yannakakis. Recursive Markov decision processes and recursive stochastic games. *J. ACM*, 62(2):11:1–11:69, 2015. doi:10.1145/2699431.

[Fan21] Zehua Fang. Robust control of Markov decision processes. In *CONF-CDS*, pages 204:1–204:9. ACM, 2021. doi:10.1145/3448734.3450934.

[FDdB14] Fernando L. Fussuma, Karina Valdivia Delgado, and Leliane Nunes de Barros. Bˆ2RTDP: An efficient solution for bounded-parameter Markov decision process. In *BRACIS*, pages 128–133. IEEE Computer Society, 2014. doi:10.1109/BRACIS.2014.33.

[Fea10] John Fearnley. Exponential lower bounds for policy iteration. In *ICALP (2)*, volume 6199 of *Lecture Notes in Computer Science*, pages 551–562. Springer, 2010. doi:10.1007/978-3-642-14162-1_46.

[FH17] Nicholas Frosst and Geoffrey E. Hinton. Distilling a neural network into a soft decision tree. In *CEx@AI*IA*, volume 2071 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017. URL `http://ceur-ws.org/Vol-2071/CExAIIA_2017_paper_3.pdf`.

[FH19] Felix Freiberger and Holger Hermanns. Concurrent programming from pseuCo to Petri. In *Petri Nets*, volume 11522 of *Lecture Notes in Computer Science*, pages 279–297. Springer, 2019. doi:10.1007/978-3-030-21571-2_16.

[Fis74] Peter C Fishburn. Exceptional paper—lexicographic orders, utilities and decision rules: A survey. *Management science*, 20(11):1442–1471, 1974. doi:10.1287/mnsc.20.11.1442.

[FKL10] Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. In *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2010. doi:10.1007/978-3-642-12002-2_16.

[FKN$^+$11] Vojtech Forejt, Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. Quantitative multi-objective verification for probabilistic systems. In *TACAS*, volume 6605 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2011. doi:10.1007/978-3-642-19835-9_11.

[FKNP11] Vojtech Forejt, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Automated verification techniques for probabilistic systems. In *SFM*, volume 6659 of *Lecture Notes in Computer Science*, pages 53–113. Springer, 2011. doi:10.1007/978-3-642-21455-4_3.

[FKP12] Vojtech Forejt, Marta Z. Kwiatkowska, and David Parker. Pareto curves for probabilistic model checking. In *ATVA*, volume 7561 of *Lecture Notes in Computer Science*, pages 317–332. Springer, 2012. doi:10.1007/978-3-642-33386-6_25.

[FKR95] Jerzy A. Filar, Dmitry Krass, and Keith Ross. Percentile performance criteria for limiting average Markov decision processes. *IEEE Trans. Autom. Control.*, 40(1):2–10, 1995. doi:10.1109/9.362904.

[FMY97] Masahiro Fujita, Patrick C. McGeer, and Jerry Chih-Yuan Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods Syst. Des.*, 10(2/3):149–169, 1997. doi:10.1023/A:1008647823331.

[Fri94] Avner Friedman. Chapter 22 Differential games. In *Handbook of Game Theory with Economic Applications*, volume 2, pages 781–799. Elsevier, 1994. doi:10.1016/S1574-0005(05)80054-2.

[Fri11] Oliver Friedmann. An exponential lower bound for the latest deterministic strategy iteration algorithms. *Log. Methods Comput. Sci.*, 7(3), 2011. doi:10.2168/LMCS-7(3:23)2011.

[FS15] John Fearnley and Rahul Savani. The complexity of the simplex method. In *STOC*, pages 201–208. ACM, 2015. doi:10.1145/2746539.2746558.

[FS18] John Fearnley and Rahul Savani. The complexity of all-switches strategy improvement. *Log. Methods Comput. Sci.*, 14(4), 2018. doi:10.23638/LMCS-14(4:9)2018.

[FT14] Jie Fu and Ufuk Topcu. Probably approximately correct MDP learning and control with temporal logic constraints. In *Robotics: Science and Systems*, 2014. doi:10.15607/RSS.2014.X.039.

[FV97] Jerzy Filar and Koos Vrieze. *Competitive Markov decision processes*. Springer Science & Business Media, 1997. URL `https://link.springer.com/book/10.1007/978-1-4612-4054-9`.

[GH08] Hugo Gimbert and Florian Horn. Simple stochastic games with few random vertices are easy to solve. In *FoSSaCS*, volume 4962 of *Lecture Notes in Computer Science*, pages 5–19. Springer, 2008. doi:10.1007/978-3-540-78499-9_2.

[GH10] Hugo Gimbert and Florian Horn. Solving simple stochastic tail games. In *SODA*, pages 847–862. SIAM, 2010. doi:10.1137/1.9781611973075.69.

[GHS18] Paul Gainer, Ernst Moritz Hahn, and Sven Schewe. Accelerated model checking of parametric Markov chains. In *ATVA*, volume 11138 of *Lecture Notes in Computer Science*, pages 300–316. Springer, 2018. doi:10.1007/978-3-030-01090-4_18.

[Gir13] Antoine Girard. Low-complexity quantized switching controllers using approximate bisimulation. *Nonlinear Analysis: Hybrid Systems*, 10:34–44, 2013. URL `https://www.sciencedirect.com/science/article/abs/pii/S1751570X13000071?casa_token=w864bz-D_W8AAAAA:CqNm5Eu-qoOh8Zo-XjP869nU1k9ggczPsp4VDbUxq_IC3dLXBSDCy4Xc1pJkRW23SDjc2Gvz__I`.

[Gir14] Sergio Giro. Optimal schedulers vs optimal bases: An approach for efficient exact solving of Markov decision processes. *Theor. Comput. Sci.*, 538:70–83, 2014. doi:10.1016/j.tcs.2013.08.020.

[GKG$^+$10] Gilles Geeraerts, Gabriel Kalyon, Tristan Le Gall, Nicolas Maquet, and Jean-François Raskin. Lattice-valued binary decision diagrams. In *ATVA*, volume

6252 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2010. doi:10.1007/978-3-642-15643-4_13.

[GKW19] Julian Gutierrez, Sarit Kraus, and Michael J. Wooldridge. Cooperative concurrent games. In *AAMAS*, pages 1198–1206. International Foundation for Autonomous Agents and Multiagent Systems, 2019. URL `http://dl.acm.org/citation.cfm?id=3331822`.

[GLD00] Robert Givan, Sonia M. Leach, and Thomas L. Dean. Bounded-parameter Markov decision processes. *Artif. Intell.*, 122(1-2):71–109, 2000. doi:10.1016/S0004-3702(00)00047-3.

[GLH$^+$19] Gaetano Geck, Artur Ljulin, Jonas Haldimann, Johannes May, Jonas Schmidt, Marko Schmellenkamp, Daniel Sonnabend, Felix Tschirbs, Fabian Vehlken, and Thomas Zeume. Teaching logic with iltis: an interactive, web-based system. In *ITiCSE*, page 307. ACM, 2019. doi:10.1145/3304221.3325571.

[GLP$^+$18] Gaetano Geck, Artur Ljulin, Sebastian Peter, Jonas Schmidt, Fabian Vehlken, and Thomas Zeume. Introduction to iltis: an interactive, web-based system for teaching logic. In *ITiCSE*, pages 141–146. ACM, 2018. doi:10.1145/3197091.3197095.

[GNMR16] Pranav Garg, Daniel Neider, P. Madhusudan, and Dan Roth. Learning invariants using decision trees and implication counterexamples. In *POPL*, pages 499–512. ACM, 2016. doi:10.1145/2837614.2837664.

[GP$^+$14] Ibrahim Ghafir, Vaclav Prenosil, et al. Advanced persistent threat attack detection: an overview. *Int J Adv Comput Netw Secur*, 4(4):5054, 2014. URL `https://www.researchgate.net/profile/Ibrahim_Ghafir/publication/305956804_Advanced_Persistent_Threat_Attack_Detection_An_Overview/links/57a7568008ae455e854698aa.pdf`.

[GSC$^+$19] David Gunning, Mark Stefik, Jaesik Choi, Timothy Miller, Simone Stumpf, and Guang-Zhong Yang. XAI - explainable artificial intelligence. *Sci. Robotics*, 4(37), 2019. doi:10.1126/scirobotics.aay7120.

[GSC21] Julie Gerlings, Arisa Shollo, and Ioanna D. Constantiou. Reviewing the need for explainable artificial intelligence (xAI). In *HICSS*, pages 1–10. ScholarSpace, 2021. URL `https://hdl.handle.net/10125/70768`.

[HA16] Mikella Hurley and Julius Adebayo. Credit scoring in the era of big data. *Yale JL & Tech.*, 18:148, 2016. URL `https://openyls.law.yale.edu/bitstream/handle/20.500.13051/7808/Hurley_Mikella.pdf?sequence=2&isAllowed=y`.

[Hal07] Nir Halman. Simple stochastic games, parity games, mean payoff games and discounted payoff games are all LP-type problems. *Algorithmica*, 49(1):37–50, 2007. doi:10.1007/s00453-007-0175-3.

[Har22] Arnd Hartmanns. Correct probabilistic model checking with floating-point arithmetic. In *TACAS (2)*, volume 13244 of *Lecture Notes in Computer Science*, pages 41–59. Springer, 2022. doi:10.1007/978-3-030-99527-0_3.

[Has17a] Vahid Hashemi. *Decision algorithms for modelling, optimal control and verification of probabilistic systems.* PhD thesis, Saarland University, Saarbrücken, Germany, 2017. URL `https://publikationen.sulb.uni-saarland.de/handle/20.500.11880/26947`.

[Has17b] Demis Hassabis. Artificial intelligence: chess match of the century. *Nature*, 544(7651):413–414, 2017. URL `https://www.nature.com/articles/544413a.pdf`.

[Has20] Mohammadhosein Hasanbeig. *Safe and certified reinforcement learning with logical constraints.* PhD thesis, University of Oxford, UK, 2020. URL `https://ora.ox.ac.uk/objects/uuid:b3270e19-8ae7-4c0f-b728-c2e4f1fab6f5`.

[HBK17] Lisa Hutschenreiter, Christel Baier, and Joachim Klein. Parametric Markov chains: PCTL complexity and fraction-free Gaussian elimination. In *GandALF*, volume 256 of *EPTCS*, pages 16–30, 2017. doi:10.4204/EPTCS.256.2.

[HBKS21] Aria HasanzadeZonuzy, Archana Bura, Dileep M. Kalathil, and Srinivas Shakkottai. Learning with safety constraints: Sample complexity of reinforcement learning for constrained MDPs. In *AAAI*, pages 7667–7674. AAAI Press, 2021. URL `https://ojs.aaai.org/index.php/AAAI/article/view/16937`.

[HdHA15] Sofie Haesaert, Paul M. J. Van den Hof, and Alessandro Abate. Data-driven property verification of grey-box systems by Bayesian experiment design. In *ACC*, pages 1800–1805. IEEE, 2015. doi:10.1109/ACC.2015.7170994.

[HDM$^+$11] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decis. Support Syst.*, 51(1):141–154, 2011. doi:10.1016/j.dss.2010.12.003.

[HH14] Arnd Hartmanns and Holger Hermanns. The Modest Toolset: An integrated environment for quantitative modelling and verification. In *TACAS*, volume 8413 of *Lecture Notes in Computer Science*, pages 593–598. Springer, 2014. doi:10.1007/978-3-642-54862-8_51.

[HHH$^+$19a] Ernst Moritz Hahn, Arnd Hartmanns, Christian Hensel, Michaela Klauck, Joachim Klein, Jan Kretínský, David Parker, Tim Quatmann, Enno Ruijters, and Marcel Steinmetz. The 2019 comparison of tools for the analysis of quantitative formal models - (QComp 2019 competition report). In *TACAS (3)*, volume 11429 of *Lecture Notes in Computer Science*, pages 69–92. Springer, 2019. doi:10.1007/978-3-030-17502-3_5.

[HHH$^+$19b] Ernst Moritz Hahn, Vahid Hashemi, Holger Hermanns, Morteza Lahijanian, and Andrea Turrini. Interval Markov decision processes with multiple objectives: From robust strategies to Pareto curves. *ACM Trans. Model. Comput. Simul.*, 29(4):27:1–27:31, 2019. doi:10.1145/3309683.

[HJB$^+$10] Ru He, Paul Jennings, Samik Basu, Arka P. Ghosh, and Huaiqing Wu. A bounded statistical approach for model checking of unbounded until properties. In *ASE*, pages 225–234. ACM, 2010. doi:10.1145/1858996.1859043.

[HJKQ20] Arnd Hartmanns, Sebastian Junges, Joost-Pieter Katoen, and Tim Quatmann. Multi-cost bounded tradeoff analysis in MDP. *J. Autom. Reason.*, 64(7):1483–1522, 2020. doi:10.1007/s10817-020-09574-9.

[HK66] A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966. doi:10.1287/mnsc.12.5.359.

[HK20] Arnd Hartmanns and Benjamin Lucien Kaminski. Optimistic value iteration. In *CAV (2)*, volume 12225 of *Lecture Notes in Computer Science*, pages 488–511. Springer, 2020. doi:10.1007/978-3-030-53291-8_26.

[HKL$^+$11] Kristoffer Arnsfelt Hansen, Michal Koucký, Niels Lauritzen, Peter Bro Miltersen, and Elias P. Tsigaridas. Exact algorithms for solving stochastic games: extended abstract. In *STOC*, pages 205–214. ACM, 2011. doi:10.1145/1993636.1993665.

[HKL17] Christoph Haase, Stefan Kiefer, and Markus Lohrey. Computing quantiles in Markov chains with multi-dimensional costs. In *LICS*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005090.

[HKM08] Tingting Han, Joost-Pieter Katoen, and Alexandru Mereacre. Approximate parameter synthesis for probabilistic time-bounded reachability. In *RTSS*, pages 173–182. IEEE Computer Society, 2008. doi:10.1109/RTSS.2008.19.

[HKP$^+$19] Arnd Hartmanns, Michaela Klauck, David Parker, Tim Quatmann, and Enno Ruijters. The quantitative verification benchmark set. In *TACAS (1)*, volume 11427 of *Lecture Notes in Computer Science*, pages 344–350. Springer, 2019. doi:10.1007/978-3-030-17462-0_20.

[HLS21] Norbert Hundeshagen, Martin Lange, and Georg Siebert. DiMo - discrete modelling using propositional logic. In *SAT*, volume 12831 of *Lecture Notes in Computer Science*, pages 242–250. Springer, 2021. doi:10.1007/978-3-030-80223-3_17.

[HM18] Serge Haddad and Benjamin Monmege. Interval iteration algorithm for MDPs and IMDPs. *Theor. Comput. Sci.*, 735:111–131, 2018. doi:10.1016/j.tcs.2016.12.003.

[HMZ$^+$12] David Henriques, João G. Martins, Paolo Zuliani, André Platzer, and Edmund M. Clarke. Statistical model checking for Markov decision processes. In *QEST*, pages 84–93. IEEE Computer Society, 2012. doi:10.1109/QEST.2012.19.

[HMZ13] Thomas Dueholm Hansen, Peter Bro Miltersen, and Uri Zwick. Strategy iteration is strongly polynomial for 2-player turn-based stochastic games with a constant discount factor. *J. ACM*, 60(1):1:1–1:16, 2013. doi:10.1145/2432622.2432623.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963. URL `https://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500830`.

[How60] Ronald A Howard. *Dynamic programming and Markov processes.* John Wiley, 1960. URL `https://www.semanticscholar.org/paper/Dynamic-Programming-and-Markov-Processes.-Ronald-A.-Weiss/89a2f75b8c5a13f897580f8c3f3c886998228708`.

[HPS+19] Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Omega-regular objectives in model-free reinforcement learning. In *TACAS (1)*, volume 11427 of *Lecture Notes in Computer Science*, pages 395–412. Springer, 2019. doi:10.1007/978-3-030-17462-0_27.

[HR76] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete. *Inf. Process. Lett.*, 5(1):15–17, 1976. doi:10.1016/0020-0190(76)90095-8.

[HR00] Thomas A. Henzinger and Jean-François Raskin. Robust undecidability of timed and hybrid systems. In *HSCC*, volume 1790 of *Lecture Notes in Computer Science*, pages 145–159. Springer, 2000. doi:10.1007/3-540-46430-1_15.

[HSHB99] Jesse Hoey, Robert St-Aubin, Alan J. Hu, and Craig Boutilier. SPUDD: stochastic planning using decision diagrams. In *UAI*, pages 279–288. Morgan Kaufmann, 1999. URL `https://arxiv.org/ftp/arxiv/papers/1301/1301.6704.pdf`.

[HZ18] Linan Huang and Quanyan Zhu. Adaptive strategic cyber defense for advanced persistent threats in critical infrastructure networks. *SIGMETRICS Perform. Evaluation Rev.*, 46(2):52–56, 2018. doi:10.1145/3305218.3305239.

[IM12] Rasmus Ibsen-Jensen and Peter Bro Miltersen. Solving simple stochastic games with few coin toss positions. In *ESA*, volume 7501 of *Lecture Notes in Computer Science*, pages 636–647. Springer, 2012. doi:10.1007/978-3-642-33090-2_55.

[Jac20] Mathias Jackermeier. *dtControl: Decision tree learning for explainable controller representation.* Bachelor's thesis, Technical University of Munich, 2020. URL `https://mediatum.ub.tum.de/doc/1547107/file.pdf`.

[JBB$^+$17] Swen Jacobs, Roderick Bloem, Romain Brenguier, Rüdiger Ehlers, Timotheus Hell, Robert Könighofer, Guillermo A. Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup, and Adam Walker. The first reactive synthesis competition (SYNTCOMP 2014). *Int. J. Softw. Tools Technol. Transf.*, 19(3):367–390, 2017. doi:10.1007/s10009-016-0416-3.

[JCL$^+$09] Sumit Kumar Jha, Edmund M. Clarke, Christopher James Langmead, Axel Legay, André Platzer, and Paolo Zuliani. A Bayesian approach to model checking biological systems. In *CMSB*, volume 5688 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2009. doi:10.1007/978-3-642-03845-7_15.

[JDT10] Manuel Mazo Jr., Anna Davitian, and Paulo Tabuada. PESSOA: A tool for embedded controller synthesis. In *CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 566–569. Springer, 2010. doi:10.1007/978-3-642-14295-6_49.

[JJD$^+$16] Sebastian Junges, Nils Jansen, Christian Dehnert, Ufuk Topcu, and Joost-Pieter Katoen. Safety-constrained reinforcement learning for MDPs. In *TACAS*, volume 9636 of *Lecture Notes in Computer Science*, pages 130–146. Springer, 2016. doi:10.1007/978-3-662-49674-9_8.

[JKKK18] Nils Jansen, Joost-Pieter Katoen, Pushmeet Kohli, and Jan Kretínský. Machine learning and model checking join forces (Dagstuhl seminar 18121). *Dagstuhl Reports*, 8(3):74–93, 2018. doi:10.4230/DagRep.8.3.74.

[JKKP13] Kangkook Jee, Vasileios P. Kemerlis, Angelos D. Keromytis, and Georgios Portokalidis. ShadowReplica: efficient parallelization of dynamic data flow tracking. In *CCS*, pages 235–246. ACM, 2013. doi:10.1145/2508859.2516704.

[JL91] Bengt Jonsson and Kim Guldstrand Larsen. Specification and refinement of probabilistic processes. In *LICS*, pages 266–277. IEEE Computer Society, 1991. doi:10.1109/LICS.1991.151651.

[JLD$^+$17] Yang Ji, Sangho Lee, Evan Downing, Weiren Wang, Mattia Fazzini, Taesoo Kim, Alessandro Orso, and Wenke Lee. RAIN: refinable attack investigation with on-demand inter-process information flow tracking. In *CCS*, pages 377–390. ACM, 2017. doi:10.1145/3133956.3134045.

[JLS12] Cyrille Jégourel, Axel Legay, and Sean Sedwards. A platform for high performance statistical model checking - PLASMA. In *TACAS*, volume 7214 of *Lecture Notes in Computer Science*, pages 498–503. Springer, 2012. doi:10.1007/978-3-642-28756-5_37.

[Joh07] David S. Johnson. The NP-completeness column: Finding needles in haystacks. *ACM Trans. Algorithms*, 3(2):24, 2007. doi:10.1145/1240233.1240247.

[JPA$^+$22] Swen Jacobs, Guillermo A. Pérez, Remco Abraham, Véronique Bruyère, Michaël Cadilhac, Maximilien Colange, Charly Delfosse, Tom van Dijk, Alexandre Duret-Lutz, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Michael Luttenberger, Klara J. Meyer, Thibaud Michaud, Adrien Pommellet, Florian Renkin, Philipp Schlehuber-Caissier, Mouhammad Sakr, Salomon Sickert, Gaëtan Staquet, Clement Tamines, Leander Tentrup, and Adam Walker. The reactive synthesis competition (SYNTCOMP): 2018-2021. *CoRR*, abs/2206.00251, 2022. doi:10.48550/arXiv.2206.00251.

[Jun20] Sebastian Junges. *Parameter synthesis in Markov models.* PhD thesis, RWTH Aachen University, Germany, 2020. URL `https://publications.rwth-aachen.de/record/783179`.

[Jün21] Florian Jüngermann. *Learning algebraic predicates for explainable controllers.* Bachelor's thesis, Technical University of Munich, 2021. URL `https://mediatum.ub.tum.de/doc/1612766/1612766.pdf`.

[KCS02] Suresh Kalyanasundaram, Edwin K. P. Chong, and Ness B. Shroff. Markov decision processes with uncertain transition rates: sensitivity and robust control. In *CDC*, pages 3799–3804. IEEE, 2002. doi:10.1109/CDC.2002.1184956.

[KGAK22] Jonis Kiesbye, Kush Grover, Pranav Ashok, and Jan Kretinsky. Planning via model checking with decision-tree controllers. *ICRA (To appear)*, 2022.

[KJL$^+$16] Kenan Kalajdzic, Cyrille Jégourel, Anna Lukina, Ezio Bartocci, Axel Legay, Scott A. Smolka, and Radu Grosu. Feedback control for statistical model checking of cyber-physical systems. In *ISoLA (1)*, volume 9952 of *Lecture Notes in Computer Science*, pages 46–61, 2016. doi:10.1007/978-3-319-47166-2_4.

[KK99] Michael J. Kearns and Daphne Koller. Efficient reinforcement learning in factored MDPs. In *IJCAI*, pages 740–747. Morgan Kaufmann, 1999. URL `http://ijcai.org/Proceedings/99-2/Papers/013.pdf`.

[KKKW18] Edon Kelmendi, Julia Krämer, Jan Kretinsky, and Maximilian Weininger. Value iteration for simple stochastic games: Stopping criterion and learning algorithm. In *CAV (1)*, volume 10981 of *Lecture Notes in Computer Science*, pages 623–642. Springer, 2018. doi:10.1007/978-3-319-96145-3_36.

[KKLW12] Joost-Pieter Katoen, Daniel Klink, Martin Leucker, and Verena Wolf. Three-valued abstraction for probabilistic systems. *J. Log. Algebraic Methods Program.*, 81(4):356–389, 2012. doi:10.1016/j.jlap.2012.03.007.

[KKNP10] Mark Kattenbelt, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. A game-based abstraction-refinement framework for Markov decision processes. *Formal Methods Syst. Des.*, 36(3):246–280, 2010. doi:10.1007/s10703-010-0097-6.

[KLC98] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134, 1998. doi:10.1016/S0004-3702(98)00023-X.

[KLvdPT19] Fabrice Kordon, Michael Leuschel, Jaco van de Pol, and Yann Thierry-Mieg. Software architecture of modern model checkers. In *Computing and Software Science*, volume 10000 of *Lecture Notes in Computer Science*, pages 393–419. Springer, 2019. doi:10.1007/978-3-319-91908-9_20.

[KM17] Jan Kretínský and Tobias Meggendorfer. Efficient strategy iteration for mean payoff in Markov decision processes. In *ATVA*, volume 10482 of *Lecture Notes in Computer Science*, pages 380–399. Springer, 2017. doi:10.1007/978-3-319-68167-2_25.

[KM18] Jan Kretínský and Tobias Meggendorfer. Conditional value-at-risk for reachability and mean payoff in Markov decision processes. In *LICS*, pages 609–618. ACM, 2018. doi:10.1145/3209108.3209176.

[KM20] Jan Kretínský and Tobias Meggendorfer. Of cores: A partial-exploration framework for Markov decision processes. *Log. Methods Comput. Sci.*, 16(4), 2020. URL `https://lmcs.episciences.org/6833`.

[KMMP20] Jan Kretínský, Fabian Michel, Lukas Michel, and Guillermo A. Pérez. Finite-memory near-optimal learning for Markov decision processes with long-run average reward. In *UAI*, volume 124 of *Proceedings of Machine Learning Research*, pages 1149–1158. AUAI Press, 2020. URL `http://proceedings.mlr.press/v124/kretinsky20a.html`.

[KMWW21] Jan Křetínskỳ, Tobias Meggendorfer, Clara Waldmann, and Maximilian Weininger. Index appearance record with preorders. *Acta Informatica*, pages 1–34, 2021. doi:10.1007/s00236-021-00412-y.

[KNP04] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Probabilistic symbolic model checking with PRISM: a hybrid approach. *Int. J. Softw. Tools Technol. Transf.*, 6(2):128–142, 2004. doi:10.1007/s10009-004-0140-2.

[KNP11] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011. doi:10.1007/978-3-642-22110-1_47.

[KNP12] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. The PRISM benchmark suite. In *QEST*, pages 203–204. IEEE Computer Society, 2012. doi:10.1109/QEST.2012.14.

[KNPQ13] Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. Compositional probabilistic verification through multi-objective model checking. *Inf. Comput.*, 232:38–65, 2013. doi:10.1016/j.ic.2013.10.001.

[KNPS20] Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos. PRISM-games 3.0: Stochastic game verification with concurrency, equilibria and time. In *CAV (2)*, volume 12225 of *Lecture Notes in Computer Science*, pages 475–487. Springer, 2020. doi:10.1007/978-3-030-53291-8_25.

[KNPS22] Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos. Correlated equilibria and fairness in concurrent stochastic games. In *TACAS (2)*, volume 13244 of *Lecture Notes in Computer Science*, pages 60–78. Springer, 2022. doi:10.1007/978-3-030-99527-0_4.

[KP99] Daphne Koller and Ronald Parr. Computing factored value functions for policies in structured MDPs. In *IJCAI*, pages 1332–1339. Morgan Kaufmann, 1999. URL `http://ijcai.org/Proceedings/99-2/Papers/094.pdf`.

[KPR18] Jan Kretínský, Guillermo A. Pérez, and Jean-François Raskin. Learning-based mean-payoff optimization in an unknown MDP under omega-regular constraints. In *CONCUR*, volume 118 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.8.

[KPV16] Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016. doi:10.1007/s10472-016-9508-8.

[KPW16] Marta Kwiatkowska, David Parker, and Clemens Wiltsche. PRISM-games 2.0: A tool for multi-objective strategy synthesis for stochastic games. In *TACAS*, volume 9636 of *Lecture Notes in Computer Science*, pages 560–566. Springer, 2016. doi:10.1007/978-3-662-49674-9_35.

[Kre16] Jan Kretínský. Survey of statistical verification of linear unbounded properties: Model checking and distances. In *ISoLA (1)*, volume 9952 of *Lecture Notes in Computer Science*, pages 27–45, 2016. doi:10.1007/978-3-319-47166-2_3.

[Kre17] Jan Kretínský. 30 years of modal transition systems: Survey of extensions and analysis. In *Models, Algorithms, Logics and Tools*, volume 10460 of *Lecture Notes in Computer Science*, pages 36–74. Springer, 2017. doi:10.1007/978-3-319-63121-9_3.

[KRSW20] Jan Kretinsky, Emanuel Ramneantu, Alexander Slivinskiy, and Maximilian Weininger. Comparison of algorithms for simple stochastic games. In *GandALF*, volume 326 of *EPTCS*, pages 131–148, 2020. doi:10.4204/EPTCS.326.9.

[KRSW22] Jan Kretinsky, Emanuel Ramneantu, Alexander Slivinskiy, and Maximilian Weininger. Comparison of algorithms for simple stochastic games. *Inf. Comput.*, 2022. URL `https://doi.org/10.1016/j.ic.2022.104885`.

[KS21] Orna Kupferman and Noam Shenwald. Perspective multi-player games. In *LICS*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470616.

[KTK80] Mikhail K Kozlov, Sergei P Tarasov, and Leonid G Khachiyan. The polynomial solvability of convex quadratic programming. *USSR Computational Mathematics and Mathematical Physics*, 20(5):223–228, 1980. doi:10.1016/0041-5553(80)90098-1.

[KU02] Igor Kozine and Lev V. Utkin. Interval-valued finite Markov chains. *Reliab. Comput.*, 8(2):97–113, 2002. doi:10.1023/A:1014745904458.

[KV19] Orna Kupferman and Gal Vardi. Perspective games. In *LICS*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785662.

[Kwi16] Marta Z. Kwiatkowska. Model checking and strategy synthesis for stochastic games: From theory to practice. In *ICALP*, volume 55 of *LIPIcs*, pages 4:1–4:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.4.

[LAB15] Morteza Lahijanian, Sean B. Andersson, and Calin Belta. Formal verification and synthesis for discrete-time stochastic systems. *IEEE Trans. Autom. Control.*, 60(8):2031–2045, 2015. doi:10.1109/TAC.2015.2398883.

[Lar12] Kim G. Larsen. Statistical model checking, refinement checking, optimization, ... for stochastic hybrid systems. In *FORMATS*, volume 7595 of *Lecture Notes in Computer Science*, pages 7–10. Springer, 2012. doi:10.1007/978-3-642-33365-1_2.

[Lar13] Kim Guldstrand Larsen. Priced timed automata and statistical model checking. In *IFM*, volume 7940 of *Lecture Notes in Computer Science*, pages 154–161. Springer, 2013. doi:10.1007/978-3-642-38613-8_11.

[LaV00] Steven M. LaValle. Robot motion planning: A game-theoretic foundation. *Algorithmica*, 26(3-4):430–465, 2000. doi:10.1007/s004539910020.

[LBB$^+$18] Yamilet R. Serrano Llerena, Marcel Böhme, Marc Brünink, Guoxin Su, and David S. Rosenblum. Verifying the long-run behavior of probabilistic system models in the presence of uncertainty. In *ESEC/SIGSOFT FSE*, pages 587–597. ACM, 2018. doi:10.1145/3236024.3236078.

[LCMF18] Antoine Lemay, Joan Calvet, François Menet, and José M. Fernandez. Survey of publicly available reports on advanced persistent threat actors. *Comput. Secur.*, 72:26–59, 2018. doi:10.1016/j.cose.2017.08.005.

[LDB10] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In *RV*, volume 6418 of *Lecture Notes in Computer Science*, pages 122–135. Springer, 2010. doi:10.1007/978-3-642-16612-9_11.

[Ley02] Michael Ley. The DBLP computer science bibliography: Evolution, research issues, perspectives. In *SPIRE*, volume 2476 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2002. doi:10.1007/3-540-45735-6_1.

[LHJ19] Josje Lodder, Bastiaan Heeren, and Johan Jeuring. A comparison of elaborated and restricted feedback in LogEx, a tool for teaching rewriting logical formulae. *J. Comput. Assist. Learn.*, 35(5):620–632, 2019. doi:10.1111/jcal.12365.

[LHJN21] Josje Lodder, Bastiaan Heeren, Johan Jeuring, and Wendy Neijenhuis. Generation and use of hints and feedback in a Hilbert-style axiomatic proof tutor. *Int. J. Artif. Intell. Educ.*, 31(1):99–133, 2021. doi:10.1007/s40593-020-00222-2.

[Lit94] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *ICML*, pages 157–163. Morgan Kaufmann, 1994. doi:10.1016/b978-1-55860-335-6.50027-1.

[LL08] Jianwei Li and Weiyi Liu. A novel heuristic Q-learning algorithm for solving stochastic games. In *IJCNN*, pages 1135–1144, 2008. URL `https://doi.org/10.1109/IJCNN.2008.4633942`.

[LL14] Kim Guldstrand Larsen and Axel Legay. Statistical model checking past, present, and future - (track introduction). In *ISoLA (2)*, volume 8803 of *Lecture Notes in Computer Science*, pages 135–142. Springer, 2014. doi:10.1007/978-3-662-45231-8_10.

[LL20] Kim G. Larsen and Axel Legay. 30 years of statistical model checking. In *ISoLA (1)*, volume 12476 of *Lecture Notes in Computer Science*, pages 325–330. Springer, 2020. doi:10.1007/978-3-030-61362-4_18.

[LLT$^+$19] Axel Legay, Anna Lukina, Louis-Marie Traonouez, Junxing Yang, Scott A. Smolka, and Radu Grosu. Statistical model checking. In *Computing and Software Science*, volume 10000 of *Lecture Notes in Computer Science*, pages 478–504. Springer, 2019. doi:10.1007/978-3-319-91908-9_23.

[LM13] Kun Lin and Steven I. Marcus. Dynamic programming with non-convex risk-sensitive measures. In *ACC*, pages 6778–6783. IEEE, 2013. doi:10.1109/ACC.2013.6580904.

[LMT15] Kim Guldstrand Larsen, Marius Mikucionis, and Jakob Haahr Taankvist. Safe and optimal adaptive cruise control. In *Correct System Design*, volume 9360 of *Lecture Notes in Computer Science*, pages 260–277. Springer, 2015. doi:10.1007/978-3-319-23506-6_17.

[LN81] S. Lakshmivarahan and Kumpati S. Narendra. Learning algorithms for two-person zero-sum stochastic games with incomplete information. *Math. Oper. Res.*, 6(3):379–386, 1981. doi:10.1287/moor.6.3.379.

[LP08] Richard Lassaigne and Sylvain Peyronnet. Probabilistic verification and approximation. *Ann. Pure Appl. Log.*, 152(1-3):122–131, 2008. doi:10.1016/j.apal.2007.11.006.

[LP12] Richard Lassaigne and Sylvain Peyronnet. Approximate planning and verification for large Markov decision processes. In *SAC*, pages 1314–1319. ACM, 2012. doi:10.1145/2245276.2231984.

[LPLSA17] Rodrigo Lankaites Pinheiro, Dario Landa-Silva, and Jason Atkin. A technique based on trade-off maps to visualise and analyse relationships between objectives in optimisation problems. *Journal of Multi-Criteria Decision Analysis*, 24(1-2):37–56, 2017. doi:10.1002/mcda.1604.

[LPTR10] Shuping Liu, Anand Panangadan, Ashit Talukder, and Cauligi S Raghavendra. Compact representation of coordinated sampling policies for body sensor networks. In *2010 IEEE Globecom Workshops*, pages 2044–2048. IEEE, 2010. URL `https://ieeexplore.ieee.org/abstract/document/5700304`.

[LPY97] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *Int. J. Softw. Tools Technol. Transf.*, 1(1-2):134–152, 1997. doi:10.1007/s100090050010.

[LS17] Edward A. Lee and Sanjit A. Seshia. *Introduction to Embedded Systems, A Cyber-Physical Systems Approach, Second Edition.* MIT press, 2017. URL `https://ptolemy.berkeley.edu/books/leeseshia/`.

[LST14] Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Scalable verification of Markov decision processes. In *SEFM Workshops*, volume 8938 of *Lecture Notes in Computer Science*, pages 350–362. Springer, 2014. doi:10.1007/978-3-319-15201-1_23.

[LST16] Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. Rare events for statistical model checking an overview. In *RP*, volume 9899 of *Lecture Notes in Computer Science*, pages 23–35. Springer, 2016. doi:10.1007/978-3-319-45994-3_2.

[LT88] Kim Guldstrand Larsen and Bent Thomsen. A modal process logic. In *LICS*, pages 203–210. IEEE Computer Society, 1988. doi:10.1109/LICS.1988.5119.

[Lud95] Walter Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Inf. Comput.*, 117(1):151–155, 1995. doi:10.1006/inco.1995.1035.

[LW05] Kong-wei Lye and Jeannette M. Wing. Game strategies in network security. *Int. J. Inf. Sec.*, 4(1-2):71–86, 2005. doi:10.1007/s10207-004-0060-x.

[MA20] Gareth W. Molyneux and Alessandro Abate. ABC(SMC)^2: Simultaneous inference and model checking of chemical reaction networks. In *CMSB*, volume 12314 of *Lecture Notes in Computer Science*, pages 255–279. Springer, 2020. doi:10.1007/978-3-030-60327-4_14.

[Mar90] Donald A. Martin. An extension of borel determinacy. *Ann. Pure Appl. Log.*, 49(3):279–293, 1990. doi:10.1016/0168-0072(90)90029-2.

[MBB07] Abdel-Illah Mouaddib, Matthieu Boussard, and Maroua Bouzid. Towards a formal framework for multi-objective multiagent planning. In *AAMAS*, page 123. IFAAMAS, 2007. doi:10.1145/1329125.1329276.

[MBSG22] Christian Meske, Enrico Bunde, Johannes Schneider, and Martin Gersch. Explainable artificial intelligence: objectives, stakeholders, and future research opportunities. *Information Systems Management*, 39(1):53–63, 2022. URL `https://www.tandfonline.com/doi/full/10.1080/10580530.2020.1849465`.

[MCJ$^+$12] Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D. Nielsen, Kim G. Larsen, and Brian Nielsen. Learning Markov decision processes for model checking. In *QFM*, volume 103 of *EPTCS*, pages 49–63, 2012. doi:10.4204/EPTCS.103.6.

[MHAH22] Tim Miller, Robert R. Hoffman, Ofra Amir, and Andreas Holzinger. Special issue on explainable artificial intelligence (XAI). *Artif. Intell.*, 307:103705, 2022. doi:10.1016/j.artint.2022.103705.

[MHC03] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artif. Intell.*, 147(1-2):5–34, 2003. doi:10.1016/S0004-3702(02)00378-8.

[Mit97] Tom M. Mitchell. *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997. URL `https://www.worldcat.org/oclc/61321007`.

[MKSB93] Sreerama K. Murthy, Simon Kasif, Steven Salzberg, and Richard Beigel. OC1: A randomized induction of oblique decision trees. In *AAAI*, pages 322–327. AAAI Press / The MIT Press, 1993. URL `http://www.aaai.org/Library/AAAI/1993/aaai93-049.php`.

[MLG05] H. Brendan McMahan, Maxim Likhachev, and Geoffrey J. Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML*, volume 119 of *ACM International Conference Proceeding Series*, pages 569–576. ACM, 2005. doi:10.1145/1102351.1102423.

[MMH$^+$19] Shruti Misra, Shana Moothedath, Hossein Hosseini, Joey Allen, Linda Bushnell, Wenke Lee, and Radha Poovendran. Learning equilibria in stochastic information flow tracking games with partial knowledge. In *CDC*, pages 4053–4060. IEEE, 2019. doi:10.1109/CDC40024.2019.9029404.

[MS98] Ashok P. Maitra and William D. Sudderth. Finitely additive stochastic games with Borel measurable payoffs. *Int. J. Game Theory*, 27(2):257–267, 1998. doi:10.1007/s001820050071.

[MSA$^+$20] Shana Moothedath, Dinuka Sahabandu, Joey Allen, Andrew Clark, Linda Bushnell, Wenke Lee, and Radha Poovendran. A game-theoretic approach for dynamic information flow tracking to detect multistage advanced persistent threats. *IEEE Trans. Autom. Control.*, 65(12):5248–5263, 2020. doi:10.1109/TAC.2020.2976040.

[MSC$^+$18] Shana Moothedath, Dinuka Sahabandu, Andrew Clark, Sangho Lee, Wenke Lee, and Radha Poovendran. Multi-stage dynamic information flow tracking game. In *GameSec*, volume 11199 of *Lecture Notes in Computer Science*, pages 80–101. Springer, 2018. doi:10.1007/978-3-030-01554-1_5.

[MSL18] Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit reactive synthesis strikes back! In *CAV (1)*, volume 10981 of *Lecture Notes in Computer Science*, pages 578–586. Springer, 2018. doi:10.1007/978-3-319-96145-3_31.

[MTSS21] Basim Mahbooba, Mohan Timilsina, Radhya Sahal, and Martin Serrano. Explainable artificial intelligence (XAI) to enhance trust management in intrusion detection systems using decision tree model. *Complex.*, 2021:6634811:1–6634811:11, 2021. doi:10.1155/2021/6634811.

[Mun19] Randall Munroe. xkcd 2110. `https://xkcd.com/2110/`, 2019. Accessed: 29.05.2022.

[Mur98] Sreerama K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Min. Knowl. Discov.*, 2(4):345–389, 1998. doi:10.1023/A:1009744630224.

[NG05] Arnab Nilim and Laurent El Ghaoui. Robust control of Markov decision processes with uncertain transition matrices. *Oper. Res.*, 53(5):780–798, 2005. doi:10.1287/opre.1050.0216.

[NJ50] John F Nash Jr. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950. URL `https://www.pnas.org/doi/full/10.1073/pnas.36.1.48`.

[NM19] Daniel Neider and Oliver Markgraf. Learning-based synthesis of safety controllers. In *FMCAD*, pages 120–128. IEEE, 2019. doi:10.23919/FMCAD.2019.8894254.

[Nor98] James R Norris. *Markov chains.* Cambridge university press, 1998. URL `https://books.google.de/books/about/Markov_Chains.html?id=qM65VRmOJZAC&redir_esc=y`.

[NS03] Abraham Neyman and Sylvain Sorin. *Stochastic games and applications*, volume 570. Springer Science & Business Media, 2003. doi:10.1007/978-94-010-0189-2.

[NSM16] Daniel Neider, Shambwaditya Saha, and P. Madhusudan. Synthesizing piecewise functions by learning classifiers. In *TACAS*, volume 9636 of *Lecture Notes in Computer Science*, pages 186–203. Springer, 2016. doi:10.1007/978-3-662-49674-9_11.

[OPW13] Wlodzimierz Ogryczak, Patrice Perny, and Paul Weng. A compromise programming approach to multiobjective Markov decision processes. *Int. J. Inf. Technol. Decis. Mak.*, 12(5):1021–1054, 2013. doi:10.1142/S0219622013400075.

[PGL+13] Sucheendra K. Palaniappan, Benjamin M. Gyori, Bing Liu, David Hsu, and P. S. Thiagarajan. Statistical model checking based calibration and analysis of bio-pathway models. In *CMSB*, volume 8130 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 2013. doi:10.1007/978-3-642-40708-6_10.

[PH98] Larry D Pyeatt and Adele E Howe. Decision tree function approximation in reinforcement learning. *Computer Science Technical Report*, CS-98-112, 1998. URL `https://arxiv.org/ftp/arxiv/papers/1309/1309.6856.pdf`.

[PILM09] Giuseppe Della Penna, Benedetto Intrigila, Nadia Lauri, and Daniele Magazzeni. Fast and compact encoding of numerical controllers using OBDDs. In *Informatics in Control, Automation and Robotics*, pages 75–87. Springer, 2009. URL `https://link.springer.com/chapter/10.1007/978-3-642-00271-7_5`.

[PKSR02] Vili Podgorelec, Peter Kokol, Bruno Stiglic, and Ivan Rozman. Decision trees: an overview and their use in medicine. *Journal of medical systems*, 26(5):445–463, 2002. URL `https://link.springer.com/article/10.1023/A:1016409317640`.

[PLGM16] Rok Piltaver, Mitja Lustrek, Matjaz Gams, and Sanda Martincic-Ipsic. What makes classification trees comprehensible? *Expert Syst. Appl.*, 62:333–346, 2016. doi:10.1016/j.eswa.2016.06.009.

[PLSS13] Alberto Puggelli, Wenchao Li, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Polynomial-time verification of PCTL properties of MDPs with convex uncertainties. In *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 527–542. Springer, 2013. doi:10.1007/978-3-642-39799-8_35.

[PSB22] Jakob Piribauer, Ocan Sankur, and Christel Baier. The variance-penalized stochastic shortest path problem. *CoRR*, abs/2204.12280, 2022. doi:10.48550/arXiv.2204.12280.

[PTHH20] Kittiphon Phalakarn, Toru Takisaka, Thomas Haas, and Ichiro Hasuo. Widest paths and global propagation in bounded value iteration for stochastic games. In *CAV (2)*, volume 12225 of *Lecture Notes in Computer Science*, pages 349–371. Springer, 2020. URL `https://doi.org/10.1007/978-3-030-53291-8_19`.

[Put94] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley Series in Probability and Statistics. Wiley, 1994. doi:10.1002/9780470316887.

[PWGH13] Patrice Perny, Paul Weng, Judy Goldsmith, and Josiah Hanna. Approximation of Lorenz-optimal solutions in multiobjective Markov decision processes. In *AAAI (Late-Breaking Developments)*, volume WS-13-17 of *AAAI Technical Report*. AAAI, 2013. URL `https://arxiv.org/ftp/arxiv/papers/1309/1309.6856.pdf`.

[PY00] Christos H. Papadimitriou and Mihalis Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *FOCS*, pages 86–92. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892068.

[PY01] Christos H. Papadimitriou and Mihalis Yannakakis. Multiobjective query optimization. In *PODS*. ACM, 2001. doi:10.1145/375551.375560.

[QJK17] Tim Quatmann, Sebastian Junges, and Joost-Pieter Katoen. Markov automata with multiple objectives. In *CAV (1)*, volume 10426 of *Lecture Notes in Computer Science*, pages 140–159. Springer, 2017. doi:10.1007/978-3-319-63387-9_7.

[QK18] Tim Quatmann and Joost-Pieter Katoen. Sound value iteration. In *CAV (1)*, volume 10981 of *Lecture Notes in Computer Science*, pages 643–661. Springer, 2018. doi:10.1007/978-3-319-96145-3_37.

[QK21] Tim Quatmann and Joost-Pieter Katoen. Multi-objective optimization of long-run average and total rewards. In *TACAS (1)*, volume 12651 of *Lecture Notes in Computer Science*, pages 230–249. Springer, 2021. doi:10.1007/978-3-030-72016-2_13.

[Ras05] Jean-François Raskin. An introduction to hybrid automata. In *Handbook of Networked and Embedded Control Systems*, pages 491–518. Birkhäuser, 2005. doi:10.1007/0-8176-4404-0_21.

[Rea08] Jesse Read. A pruned problem transformation method for multi-label classification. In *NZCSRS 2008*, volume 143150, page 41, 2008. URL `https://users.ics.aalto.fi/jesse/papers/NZCSRCS08.pdf`.

[RES$^+$10] Sankardas Roy, Charles Ellis, Sajjan G. Shiva, Dipankar Dasgupta, Vivek Shandilya, and Qishi Wu. A survey of game theory as applied to network security. In *HICSS*, pages 1–10. IEEE Computer Society, 2010. doi:10.1109/HICSS.2010.35.

[RGT05] Nicholas Roy, Geoffrey J. Gordon, and Sebastian Thrun. Finding approximate POMDP solutions through belief compression. *J. Artif. Intell. Res.*, 23:1–40, 2005. doi:10.1613/jair.1496.

[RLH21] Marc Rigter, Bruno Lacerda, and Nick Hawes. Minimax regret optimisation for robust planning in uncertain Markov decision processes. In *AAAI*, pages 11930–11938. AAAI Press, 2021. URL `https://ojs.aaai.org/index.php/AAAI/article/view/17417`.

[RP09] Diana El Rabih and Nihal Pekergin. Statistical model checking using perfect simulation. In *ATVA*, volume 5799 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 2009. doi:10.1007/978-3-642-04761-9_11.

[RRS15] Mickael Randour, Jean-François Raskin, and Ocan Sankur. Variations on the stochastic shortest path problem. In *VMCAI*, volume 8931 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2015. doi:10.1007/978-3-662-46081-8_1.

[RRS17] Mickael Randour, Jean-François Raskin, and Ocan Sankur. Percentile queries in multi-dimensional Markov decision processes. *Formal Methods Syst. Des.*, 50(2-3):207–248, 2017. doi:10.1007/s10703-016-0262-7.

[RS14] Jean-François Raskin and Ocan Sankur. Multiple-environment Markov decision processes. In *FSTTCS*, volume 29 of *LIPIcs*, pages 531–543. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.531.

[Rud19] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.*, 1(5):206–215, 2019. doi:10.1038/s42256-019-0048-x.

[RVWD13] Diederik M. Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *J. Artif. Intell. Res.*, 48:67–113, 2013. doi:10.1613/jair.3987.

[RZ16] Matthias Rungger and Majid Zamani. SCOTS: A tool for the synthesis of symbolic controllers. In *HSCC*, pages 99–104. ACM, 2016. doi:10.1145/2883817.2883834.

[Sah74] Sartaj Sahni. Computationally related problems. *SIAM J. Comput.*, 3(4):262–279, 1974. doi:10.1137/0203021.

[SAM$^+$20] Dinuka Sahabandu, Joey Allen, Shana Moothedath, Linda Bushnell, Wenke Lee, and Radha Poovendran. Quickest detection of advanced persistent threats: A semi-Markov game approach. In *ICCPS*, pages 9–19. IEEE, 2020. doi:10.1109/ICCPS48487.2020.00009.

[San20] Gabriel HR Santos. *Automatic verification and strategy synthesis for zero-sum and equilibria properties of concurrent stochastic games.* PhD thesis, University of Oxford, 2020. URL `https://ora.ox.ac.uk/objects/uuid:225cfc43-df30-4dc2-919b-66e8037633a6/download_file?safe_filename=THESIS__GABRIEL_SANTOS.pdf&file_format=pdf&type_of_work=Thesis`.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018. URL `https://mitpress.mit.edu/books/reinforcement-learning-second-edition`.

[SBS20] Farahnaz Sadoughi, Ali Behmanesh, and Nasrin Sayfouri. Internet of things in medicine: A systematic mapping study. *J. Biomed. Informatics*, 103:103383, 2020. doi:10.1016/j.jbi.2020.103383.

[Sch18] François Schwarzentruber. Hintikka's world: Agents with higher-order knowledge. In *IJCAI*, pages 5859–5861. ijcai.org, 2018. doi:10.24963/ijcai.2018/862.

[SFO19] Hicham Moad Safhi, Bouchra Frikh, and Brahim Ouhbi. Assessing reliability of big data knowledge discovery process. *Procedia computer science*, 148:30–36, 2019. doi:10.1016/j.procs.2019.01.005.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(4):623–656, 1948. doi:10.1002/j.1538-7305.1948.tb00917.x.

[SHB00] Robert St-Aubin, Jesse Hoey, and Craig Boutilier. APRICODD: Approximate policy construction using decision diagrams. In *NIPS*, pages 1089–1095. MIT Press, 2000. URL `https://proceedings.neurips.cc/paper/2000/hash/201d7288b4c18a679e48b31c72c30ded-Abstract.html`.

[Sil63] Edward A Silver. Markovian decision processes with uncertain transition probabilities or rewards. Technical report, Massachusetts inst of tech cambridge operations research center, 1963. URL `https://apps.dtic.mil/sti/citations/AD0417150`.

[SJ73] Jay K. Satia and Roy E. Lave Jr. Markovian decision processes with uncertain transition probabilities. *Oper. Res.*, 21(3):728–740, 1973. doi:10.1287/opre.21.3.728.

[SK16] María Svorenová and Marta Kwiatkowska. Quantitative verification and strategy synthesis for stochastic games. *Eur. J. Control*, 30:15–30, 2016. doi:10.1016/j.ejcon.2016.04.009.

[SKD21] Lauren N. Steimle, David L. Kaufman, and Brian T. Denton. Multi-model Markov decision processes. *IISE Trans.*, 53(10):1124–1139, 2021. doi:10.1080/24725854.2021.1895454.

[SKK21] Shalini Sharma, Naresh Kumar, and Kuldeep Singh Kaswan. Big data reliability: A critical review. *J. Intell. Fuzzy Syst.*, 40(3):5501–5516, 2021. doi:10.3233/JIFS-202503.

[SKPK12] Gregor Stiglic, Simon Kocbek, Igor Pernek, and Peter Kokol. Comprehensive decision tree models in bioinformatics. *PloS one*, 7(3):e33812, 2012. URL `https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0033812`.

[SLW$^+$06] Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L. Littman. PAC model-free reinforcement learning. In *ICML*, volume 148 of *ACM International Conference Proceeding Series*, pages 881–888. ACM, 2006. URL `http://doi.acm.org/10.1145/1143844.1143955`.

[SLYD20] Jialin Song, Ravi Lanka, Yisong Yue, and Bistra Dilkina. A general large neighborhood search framework for solving integer linear programs. In *NeurIPS*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/e769e03a9d329b2e864b4bf4ff54ff39-Abstract.html`.

[SLZD04] G. Edward Suh, Jae W. Lee, David Zhang, and Srinivas Devadas. Secure program execution via dynamic information flow tracking. In *ASPLOS*, pages 85–96. ACM, 2004. doi:10.1145/1024393.1024404.

[SMA$^+$19a] Dinuka Sahabandu, Shana Moothedath, Joey Allen, Linda Bushnell, Wenke Lee, and Radha Poovendran. Stochastic dynamic information flow tracking game with reinforcement learning. In *GameSec*, volume 11836 of *Lecture Notes in Computer Science*, pages 417–438. Springer, 2019. doi:10.1007/978-3-030-32430-8_25.

[SMA$^+$19b] Dinuka Sahabandu, Shana Moothedath, Joey Allen, Andrew Clark, Linda Bushnell, Wenke Lee, and Radha Poovendran. Dynamic information flow tracking games for simultaneous detection of multiple attackers. In *CDC*, pages 567–574. IEEE, 2019. doi:10.1109/CDC40024.2019.9029836.

[SMB$^+$19] Dinuka Sahabandu, Shana Moothedath, Linda Bushnell, Radha Poovendran, Joey Allen, Wenke Lee, and Andrew Clark. A game theoretic approach for dynamic information flow tracking with conditional branching. In *ACC*, pages 2289–2296. IEEE, 2019. doi:10.23919/ACC.2019.8814596.

[Som05] Rafal Somla. New algorithms for solving simple stochastic games. *Electron. Notes Theor. Comput. Sci.*, 119(1):51–65, 2005. doi:10.1016/j.entcs.2004.07.008.

[Sor03] Sylvain Sorin. Stochastic games with incomplete information. In *Stochastic Games and applications*, pages 375–395. Springer, 2003. doi:10.1007/978-94-010-0189-2_25.

[Spa12] Matthijs T. J. Spaan. Partially observable markov decision processes. In *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 387–414. Springer, 2012. doi:10.1007/978-3-642-27645-3_12.

[SSJP22] Marnix Suilen, Thiago D. Simão, Nils Jansen, and David Parker. Robust anytime learning of Markov decision processes. *CoRR*, abs/2205.15827, 2022. doi:10.48550/arXiv.2205.15827.

[SV15] Eilon Solan and Nicolas Vieille. Stochastic games. *Proceedings of the National Academy of Sciences*, 112(45):13743–13746, 2015. URL `https://www.pnas.org/doi/epdf/10.1073/pnas.1513508112`.

[SVA04] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Statistical model checking of black-box probabilistic systems. In *CAV*, volume 3114 of *Lecture Notes in Computer Science*, pages 202–215. Springer, 2004. doi:10.1007/978-3-540-27813-9_16.

[SVA05a] Koushik Sen, Mahesh Viswanathan, and Gul Agha. On statistical model checking of stochastic systems. In *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 266–280. Springer, 2005. doi:10.1007/11513988_26.

[SVA05b] Koushik Sen, Mahesh Viswanathan, and Gul A. Agha. VESTA: A statistical model-checker and analyzer for probabilistic systems. In *QEST*, pages 251–252. IEEE Computer Society, 2005. doi:10.1109/QEST.2005.42.

[SVA06] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Model-checking Markov chains in the presence of uncertainties. In *TACAS*, volume 3920 of *Lecture Notes in Computer Science*, pages 394–410. Springer, 2006. doi:10.1007/11691372_26.

[SXC$^+$18] Dinuka Sahabandu, Baicen Xiao, Andrew Clark, Sangho Lee, Wenke Lee, and Radha Poovendran. DIFT games: Dynamic information flow tracking games for advanced persistent threats. In *CDC*, pages 1136–1143. IEEE, 2018. doi:10.1109/CDC.2018.8619416.

[Tar72] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. doi:10.1137/0201010.

[TB07] Ambuj Tewari and Peter L. Bartlett. Bounded parameter Markov decision processes with average reward criterion. In *COLT*, volume 4539 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2007. doi:10.1007/978-3-540-72927-3_20.

[TG21] Erico Tjoa and Cuntai Guan. A survey on explainable artificial intelligence (XAI): Toward medical XAI. *IEEE Trans. Neural Networks Learn. Syst.*, 32(11):4793–4813, 2021. doi:10.1109/TNNLS.2020.3027314.

[TKV08] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Effective and efficient multilabel classification in domains with large number of labels. In *Proc. ECML/PKDD 2008 Workshop on Mining Multidimensional Data (MMD'08)*, volume 21, pages 53–59, 2008. URL `http://www.ecmlpkdd2008.org/files/pdf/workshops/mmd/4.pdf`.

[TT16] Alain Tcheukam and Hamidou Tembine. One swarm per queen: A particle swarm learning for stochastic games. In *SASO*, pages 144–145, 2016. URL `https://doi.org/10.1109/SASO.2016.22`.

[TVK11] Rahul Tripathi, Elena Valkanova, and V. S. Anil Kumar. On strategy improvement algorithms for simple stochastic games. *J. Discrete Algorithms*, 9(3):263–278, 2011. doi:10.1016/j.jda.2011.03.007.

[UB13] Michael Ummels and Christel Baier. Computing quantiles in Markov reward models. In *FoSSaCS*, volume 7794 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2013. doi:10.1007/978-3-642-37075-5_23.

[Ujm15] Mateusz Ujma. *On verification and controller synthesis for probabilistic systems at runtime.* PhD thesis, University of Oxford, UK, 2015. URL `http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.711811`.

[Val84] Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. doi:10.1145/1968.1972.

[VCD$^+$15] Yaron Velner, Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, Alexander Moshe Rabinovich, and Jean-François Raskin. The complexity of multi-mean-payoff and multi-energy games. *Inf. Comput.*, 241:177–196, 2015. doi:10.1016/j.ic.2015.03.001.

[vD18] Tom van Dijk. Attracting tangles to solve parity games. In *CAV (2)*, volume 10982 of *Lecture Notes in Computer Science*, pages 198–215. Springer, 2018. doi:10.1007/978-3-319-96142-2_14.

[VDB$^+$11] Peter Vamplew, Richard Dazeley, Adam Berry, Rustam Issabekov, and Evan Dekker. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Mach. Learn.*, 84(1-2):51–80, 2011. doi:10.1007/s10994-010-5232-5.

[Vel15] Yaron Velner. Robust multidimensional mean-payoff games are undecidable. In *FoSSaCS*, volume 9034 of *Lecture Notes in Computer Science*, pages 312–327. Springer, 2015. doi:10.1007/978-3-662-46678-0_20.

[VFS18] Jørgen Villadsen, Asta Halkjær From, and Anders Schlichtkrull. Natural deduction assistant (NaDeA). In *ThEdu@FLoC*, volume 290 of *EPTCS*, pages 14–29, 2018. doi:10.4204/EPTCS.290.2.

[VG13] Nikos Virvilis and Dimitris Gritzalis. The big four - what we did wrong in advanced persistent threat detection? In *ARES*, pages 248–254. IEEE Computer Society, 2013. doi:10.1109/ARES.2013.32.

[VTRF83] OJ Vrieze, SH Tijs, Tirukkannamangai ES Raghavan, and JA Filar. A finite algorithm for the switching control stochastic game. *Operations-Research-Spektrum*, 5(1):15–24, 1983. URL `https://doi.org/10.1007/BF01720283`.

[Wal96] Peter Walley. Measures of uncertainty in expert systems. *Artif. Intell.*, 83(1):1–58, 1996. doi:10.1016/0004-3702(95)00009-7.

[Wal21] LE Wal. *Using dtControl to process schedulers produced by the Modest Toolset.* Bachelor's thesis, University of Twente, 2021. URL `http://essay.utwente.nl/87059/1/vanderWal_BA_EEMCS.pdf`.

[WE94] Chelsea C. White III and Hany K. Eldeib. Markov decision processes with imprecise transition probabilities. *Oper. Res.*, 42(4):739–749, 1994. doi:10.1287/opre.42.4.739.

[Wei20] Christoph Weinhuber. *Learning domain-specific predicates in decision trees for explainable controller representation.* Bachelor's thesis, Technical University of Munich, 2020. URL `https://mediatum.ub.tum.de/doc/1617115/1617115.pdf`.

[WGMK21] Maximilian Weininger, Kush Grover, Shruti Misra, and Jan Kretínský. Guaranteed trade-offs in dynamic information flow tracking games. In *CDC*, pages 3786–3793. IEEE, 2021. doi:10.1109/CDC45484.2021.9683447.

[WK08] Di Wu and Xenofon D. Koutsoukos. Reachability analysis of uncertain systems using bounded-parameter Markov decision processes. *Artif. Intell.*, 172(8-9):945–954, 2008. doi:10.1016/j.artint.2007.12.002.

[WKR13] Wolfram Wiesemann, Daniel Kuhn, and Berç Rustem. Robust Markov decision processes. *Math. Oper. Res.*, 38(1):153–183, 2013. doi:10.1287/moor.1120.0566.

[WMK19] Maximilian Weininger, Tobias Meggendorfer, and Jan Kretinsky. Satisfiability bounds for ω-regular properties in bounded-parameter Markov decision processes. In *CDC*, pages 2284–2291. IEEE, 2019. doi:10.1109/CDC40024.2019.9029460.

[WSQJ21] Jingyi Wang, Jun Sun, Shengchao Qin, and Cyrille Jégourel. Automatically 'verifying' discrete-time complex systems through learning, abstraction and refinement. *IEEE Trans. Software Eng.*, 47(1):189–203, 2021. doi:10.1109/TSE.2018.2886898.

[WSYP18] Jingyi Wang, Jun Sun, Qixia Yuan, and Jun Pang. Learning probabilistic models for model checking: an evolutionary approach and an empirical study. *Int. J. Softw. Tools Technol. Transf.*, 20(6):689–704, 2018. doi:10.1007/s10009-018-0492-7.

[WT16] Min Wen and Ufuk Topcu. Probably approximately correct learning in stochastic games with temporal logic specifications. In *IJCAI*, pages 3630–3636. IJCAI/AAAI Press, 2016. URL `http://www.ijcai.org/Abstract/16/511`.

[WT21] Min Wen and Ufuk Topcu. Probably approximately correct learning in adversarial environments with temporal logic specifications. *IEEE Transactions on Automatic Control*, pages 1–1, 2021. doi:10.1109/TAC.2021.3115080.

[WTM12] Eric M. Wolff, Ufuk Topcu, and Richard M. Murray. Robust control of uncertain Markov decision processes with temporal logic specifications. In *CDC*, pages 3372–3379. IEEE, 2012. doi:10.1109/CDC.2012.6426174.

[WW21] Tobias Winkler and Maximilian Weininger. Stochastic games with disjunctions of multiple objectives. In *GandALF*, volume 346 of *EPTCS*, pages 83–100, 2021. doi:10.4204/EPTCS.346.6.

[YCZ10] Håkan L. S. Younes, Edmund M. Clarke, and Paolo Zuliani. Statistical verification of probabilistic properties with unbounded until. In *SBMF*, volume 6527 of *Lecture Notes in Computer Science*, pages 144–160. Springer, 2010. doi:10.1007/978-3-642-19829-8_10.

[YKNP06] Håkan L. S. Younes, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Numerical vs. statistical probabilistic model checking. *Int. J. Softw. Tools Technol. Transf.*, 8(3):216–228, 2006. doi:10.1007/s10009-005-0187-8.

[You05] Håkan L. S. Younes. Probabilistic verification for "black-box" systems. In *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 253–265. Springer, 2005. doi:10.1007/11513988_25.

[You06] Håkan L. S. Younes. Error control for probabilistic model checking. In *VMCAI*, volume 3855 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2006. doi:10.1007/11609773_10.

[YS02] Håkan L. S. Younes and Reid G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *CAV*, volume 2404 of *Lecture Notes in Computer Science*, pages 223–235. Springer, 2002. doi:10.1007/3-540-45657-0_17.

[ZPC13] Paolo Zuliani, André Platzer, and Edmund M. Clarke. Bayesian statistical model checking with application to Stateflow/Simulink verification. *Formal Methods Syst. Des.*, 43(2):338–367, 2013. doi:10.1007/s10703-013-0195-3.

[ZSP20] Jialu Zhang, Mark Santolucito, and Ruzica Piskac. Succinct explanations with cascading decision trees. *CoRR*, abs/2010.06631, 2020, 2010.06631. URL `https://arxiv.org/abs/2010.06631`.

[ZVJ18] Ivan S. Zapreev, Cees Verdier, and Manuel Mazo Jr. Optimal symbolic controllers determinization for BDD storage. In *ADHS*, volume 51 (16) of *IFAC-PapersOnLine*, pages 1–6. Elsevier, 2018. doi:10.1016/j.ifacol.2018.08.001.

Appendix

A Value Iteration for Simple Stochastic Games: Stopping Criterion and Learning Algorithm (InC 2022)

This is an exact reprinting of the paper which has been published as a **peer reviewed journal paper**.

Julia Eisentraut, Edon Kelmendi, Jan Kretinsky, and Maximilian Weininger. "Value Iteration for Simple Stochastic Games: Stopping Criterion and Learning Algorithm". In: Information and Computation. Elsevier, 2022. doi: 10.1016/j.ic.2022.104886

Summary

The classical problem of reachability in simple stochastic games is typically solved by value iteration (VI), which produces a sequence of under-approximations of the value of the game, but is only guaranteed to converge in the limit. We provide an additional converging sequence of over-approximations, based on an analysis of the game graph. Together, these two sequences entail the first error bound and hence the first stopping criterion for VI on simple stochastic games, indicating when the algorithm can be stopped for a given precision. Consequently, VI becomes an anytime algorithm returning the approximation of the value and the current error bound. We further use this error bound to provide a learning-based asynchronous VI algorithm; it uses simulations and thus often avoids exploring the whole game graph, but still yields the same guarantees. Finally, we experimentally show that the overhead for computing the additional sequence of over- approximations often is negligible.

For more details on this publication, we refer to Chapter 3 of the main body.

Contribution

Contribution of Maximilian Weininger	
Development and conceptual design of the research project	30%
Discussion and development of ideas	50%
Composition and revision of the manuscript	90%
Implementation	100%
Experimental evaluation	100%

Information and Computation 285 (2022) 104886

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Value iteration for simple stochastic games: Stopping criterion and learning algorithm [☆]

Julia Eisentraut [a], Edon Kelmendi [b], Jan Křetínský [a], Maximilian Weininger [a,*]

[a] *Technical University of Munich, Germany*
[b] *University of Oxford, United Kingdom*

ARTICLE INFO

Article history:
Received 31 August 2020
Received in revised form 23 February 2022
Accepted 27 February 2022
Available online 4 March 2022

Keywords:
Probabilistic verification
Stochastic games
Markov decision processes
Value iteration
Reachability

ABSTRACT

The classical problem of reachability in simple stochastic games is typically solved by value iteration (VI), which produces a sequence of under-approximations of the value of the game, but is only guaranteed to converge in the limit. We provide an additional converging sequence of over-approximations, based on an analysis of the game graph. Together, these two sequences entail the first error bound and hence the first stopping criterion for VI on simple stochastic games, indicating when the algorithm can be stopped for a given precision. Consequently, VI becomes an anytime algorithm returning the approximation of the value and the current error bound. We further use this error bound to provide a learning-based asynchronous VI algorithm; it uses simulations and thus often avoids exploring the whole game graph, but still yields the same guarantees. Finally, we experimentally show that the overhead for computing the additional sequence of over-approximations often is negligible.

1. Introduction

A *simple stochastic game* (SG) [24] is a zero-sum 2.5-player game played on a graph by the two players Maximizer and Minimizer, who choose actions in their respective states. Each action is associated with a probability distribution determining the next state to move to. The objective of Maximizer is to maximize the probability of reaching a given target state; the objective of Minimizer is the opposite.

Stochastic games constitute a fundamental problem for several reasons. From the theoretical point of view, the complexity of this problem[1] is known to be in **UP** ∩ **coUP** [16], but no polynomial-time algorithm is known. Further, several other important problems can be reduced to SGs, for instance parity games, mean-payoff games, discounted-payoff games and their stochastic extensions [16]. The task of solving SGs is also polynomial-time equivalent to solving perfect information Shapley, Everett and Gillette games [1]. Besides, the problem is practically relevant in verification and synthesis. SGs can model reactive systems, with players corresponding to the controller of the system and to its environment, where quantified uncertainty is explicitly modelled. This is useful in many application domains, ranging from smart energy management [20]

[☆] This research was funded in part by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement No. 291763 for TUM – IAS, the Studienstiftung des Deutschen Volkes project "Formal methods for analysis of attack-defence diagrams", TUM IGSSE Grant 10.06 (PARSEC), and the German Research Foundation (DFG) project KR 4890/2-1 "Statistical Unbounded Verification".
[*] Corresponding author at: Technical University of Munich, Department of Informatics, Chair I 7, Boltzmannstr. 3, 85748 Garching, Germany.
E-mail address: maxi.weininger@tum.de (M. Weininger).
[1] Formally, the problem is to decide, for a given rational $p \in [0, 1]$ whether Maximizer has a strategy ensuring probability at least p to reach the target.

https://doi.org/10.1016/j.ic.2022.104886
0890-5401/© 2022 The Authors. Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

over autonomous urban driving [22] and robot motion planning [49] to self-adaptive systems [14]; for various recent case studies, see e.g. [56]. Finally, since Markov decision processes (MDP) [53] are a special case with only one player, SGs can serve as abstractions of large MDPs [38].

Solution techniques. There are several classes of algorithms for solving SGs, namely strategy iteration algorithms, quadratic programming and value iteration algorithms [25]. *Strategy iteration* (also known as policy iteration) computes a sequence of strategies converging to the optimal one. In each iteration, we fix the strategy of one player and compute the best response of the opponent in the resulting one-player game (Markov decision process); this is typically done using linear programming. Then the strategy we originally fixed is improved based on the response of the opponent. Since there are finitely many strategies and they are monotonically improving, an optimal strategy is found after finitely many iterations. Currently, the best known upper bound for the number of iterations is exponential in the size of the SG.

Quadratic programming encodes the SG into a single polynomially sized quadratic program. The constraints of the program mimic the structure of the SG and the objective function ensures that the unique solution is the one where both players behave optimally. Both strategy iteration and quadratic programming produce an exact solution.

In contrast, *value iteration* (VI) is an approximative method that only converges in the limit. It computes a sequence of value-vectors that approximates the true values from below, but may never reach them in finite time. Still, there is a way to obtain the exact values: after $\mathcal{O}(\gamma^2)$ iterations of the algorithm, where γ is a number exponential in the size of the SG, we can round the approximation to the precise result [18]. However, as the required number of iterations is always exponential and the algorithm gives no guarantee before it finishes, this approach is infeasible even for small SGs. It would be useful to be able to stop the VI algorithm early, after a required precision is reached.

Theoretically, the best known upper bound of the runtime of all three classes of algorithms is exponential in the size of the SG. Practically, VI currently is the preferred solution. For instance, the most used probabilistic model checkers Storm [28] and PRISM [44], as well as PRISM's branch PRISM-Games [43] use VI for MDPs and SGs as the default option; in the case of PRISM-games, VI is the only option. As the exponential number of iterations that is necessary to obtain a guarantee is impractical, PRISM-games stops when the difference between the two most recent approximations is low, and thus may return arbitrarily imprecise results [33].

VI was preferred in these model checkers, because evaluating many large linear programs respectively a very large quadratic program was expected to be slow, similar to the special case of MDPs [40]. However, a recent extensive comparison of these algorithms [42] showed that in fact both strategy iteration and value iteration can perform similarly well. Also, there exists cases where either of the two algorithms outperforms the other, depending on the structure of the given SG.

An advantage of strategy iteration and quadratic programming is that they can possibly produce a guaranteed result in less than exponential time, while VI only converges in the limit and always has to run for an exponential number of steps in order to obtain a precise result. We address this by complementing the VI algorithm with error bounds, allowing us to stop the computation early when a required precision is reached. Thus we can leverage VI's ability to typically produce good results fast while also giving guarantees on the precision.

Value iteration with guarantees. In the special case of MDPs, in order to obtain bounds on the imprecision of the result, one can employ a *bounded* variant of VI [51,11] (also called *interval iteration* [33]). Here one computes not only an under-approximation, but also an over-approximation of the actual value. On the one hand, iterative computation of the least fixpoint of Bellman equations yields an under-approximating sequence converging to the value. On the other hand, iterative computation of the greatest fixpoint yields an over-approximation, which, however, does not converge to the value, because the least and greatest fixpoint do not coincide. Moreover, it often results in the trivial bound of 1. A solution suggested for MDPs [11,33] is to modify the underlying graph, namely to collapse end components. In the resulting MDP there is only one fixpoint, thus the least and greatest fixpoint coincide and both approximating sequences converge to the actual value. In contrast, for general SGs no value iteration based procedure is known which approximates the least fixpoint from above. In this paper we provide such an algorithm. Together with standard value iteration from below, this yields a stopping criterion to interrupt execution of VI at a given precision. We show that the pre-processing approach of collapsing is not applicable in general and provide a solution on the original graph. We also characterize SGs where the fixpoints coincide and naive VI converges from above. The main technical challenge is that states in an end component in SGs can have different values, in contrast to the case of MDPs.

Practical efficiency using guarantees. We further utilize the obtained guarantees to practically improve our algorithm. Similar to the MDP case [11], the quantification of the error allows for ignoring parts of the state space, and thus a speed up without jeopardizing the correctness of the result. Indeed, we provide a technique where some states are not explored and processed at all, but their potential effect is still taken into account. It relies on learning algorithms with simulations; these are guided by the quantification of the error, using the information to decide which state to explore and analyse next. While for MDPs this idea has already demonstrated speed ups in orders of magnitude [11,3], this paper provides the first technique of this kind for SGs.

Our contribution. We can summarize our contribution as follows.

- We introduce a VI algorithm for under- and over-approximation sequences, both of which converge to the value of the game. Thus we present the first practical stopping criterion for VI on SGs, yielding an anytime algorithm[2] with guaranteed precision. We also characterize when a simpler solution is sufficient.
- We provide a learning-based algorithm, which preserves the guarantees, and which additionally is more efficient in some cases since it avoids exploring the whole state space.
- We evaluate the running times of the algorithms experimentally, concluding that obtaining guarantees requires an overhead that is either negligible or mitigated by the learning-based approach.

Related work. The works closest to ours are the following. As mentioned above, [11,33] describe the solution to the special case of MDPs. While [11] also provides a learning-based algorithm, [33] discusses the convergence rate and the exact solution. The basic algorithm of [33] is implemented in PRISM [8] and the learning approach of [11] in Storm [28]. The extension for SGs where the interleaving of players is severely limited (every end component belongs to one player only) is discussed in [58].

Further, in the area of probabilistic planning, bounded real-time dynamic programming [51] is related to our learning-based approach. However, it is limited to the setting of stopping MDPs where the target sink or the non-target sink is reached almost surely under any strategy.

For stopping SGs, the error of value iteration can be bounded without computing an additional over-approximation. [25, Section 3] describes how to transform an arbitrary SG to a polynomially larger stopping SG such that the original value can be inferred. In essence, the construction adds a transition to every action that reaches a sink state with very low probability γ; this is the same as computing the discounted reachability with discount factor γ. However, this approach is impractical, because the convergence rate of value iteration depends on the lowest occurring probability in the SG. Further, in order to be able to infer the original value, γ has to be chosen smaller than standard machine precision already for an SG with 27 states [42, Section 4.1.4]. Our algorithm works for general SGs, not only for stopping ones, without increasing the size of the game or introducing very low probabilities.

The idea of strategy iteration was suggested in [35], for the class of irreducible concurrent SGs. In [60], the authors provide a strategy iteration algorithm for turn-based SGs with average reward objectives. The algorithm relies on the linear program formulation from [36] that solves the one-player game (Markov decision process) after fixing one strategy. The algorithm is also described in the textbook [32, Chapter 6.3]. Note that reachability objectives can be encoded as average reward objectives. The formulation of strategy iteration explicitly for simple stochastic games was given in [25], together with a randomized version. For a description of further developments and a practical comparison of all three algorithm classes (value iteration, strategy iteration and quadratic programming), see the recent work [42].

The tools implementing the standard SI and/or VI algorithms for SGs are PRISM-games [43], GAVS+ [23] and GIST [19]. GAVS+ is however not maintained anymore, and more for educational purposes. GIST considers ω-regular objectives, but performs only qualitative analysis. For MDPs (games with a single player), the recent friendly competition QComp [34] gives an overview of the existing tools.

Apart from fundamental algorithms to solve SGs, there are various practically efficient heuristics that, however, provide none or weak guarantees, often based on some form of learning [10,50,61,57,2,12]. For MDPs, the first probably approximately correct (PAC [59]) algorithm was given in [55]. This result has been further extended [11] to MDP settings more relevant for verification: firstly, [11] considers non-discounted unbounded horizon, such as the reachability objective. Secondly, it treats not only the case when the transition probabilities are unknown, but also when they are given.

In terms of complexity, [9] recently proved that deciding whether an action in an MDP is optimal for the n-step bounded reachability is EXPTIME-complete. This means that the worst case running time for VI on SGs is exponential, since VI computes the n-step bounded reachability for all states and since SGs are an extension of MDPs. The runtime is not larger than exponential, as [18] proved that VI can always compute the precise value in an exponential number of steps.

Finally, the concept of simple end component that is introduced in this paper is similar to *tangles* in parity games [29]. Both concepts describe a strongly connected sub-graph in which one of the players can surely win, which forces the other player to leave this sub-graph.

This article is the extended version of the paper presented at CAV 2018 [39]. We added more examples, an in-depth description of the learning-based algorithm as well as more on the implementation of both our algorithms and more experimental results. Also, we provide the full technical proofs, including many instructive details in the proofs of Theorem 1 and 2, as well as the new Theorem 3 about the learning-based algorithm. In particular, we extracted Lemma 2, a statement relating the existence of exiting actions to the existence of end components. Further, we clarified the case distinction in the proof of Theorem 1 and show in Fig. 4 the many ways in which a simple end component can affect the values of other states. Furthermore, in Lemma 6, we provide a detailed lattice-theoretic argument about the convergence of the upper bound to a fixpoint; this includes a surprising complication which is elaborated on in Remark 2. Moreover, we fixed an error in the statement and proof of Lemma 1, cf. Footnote 9. Finally, we give—to the best of our knowledge for the first time—a formal description of how to obtain optimal strategies from value vectors, see Appendix A.

[2] An algorithm which can be stopped at any time in the computation, returning an estimate and its precision.

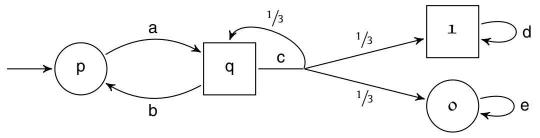


Fig. 1. An example of an SG with $S = \{\mathsf{p}, \mathsf{q}, \mathsf{1}, \mathsf{0}\}$, $S_\square = \{\mathsf{q}, \mathsf{1}\}$, $S_\bigcirc = \{\mathsf{p}, \mathsf{0}\}$, the initial state p and the set of actions $\mathsf{A} = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}\}$; $\mathsf{Av}(\mathsf{p}) = \{\mathsf{a}\}$ with $\delta(\mathsf{p}, \mathsf{a})(\mathsf{q}) = 1$; $\mathsf{Av}(\mathsf{q}) = \{\mathsf{b}, \mathsf{c}\}$ with $\delta(\mathsf{q}, \mathsf{b})(\mathsf{p}) = 1$ and $\delta(\mathsf{q}, \mathsf{c})(\mathsf{q}) = \delta(\mathsf{q}, \mathsf{c})(\mathsf{1}) = \delta(\mathsf{q}, \mathsf{c})(\mathsf{0}) = \frac{1}{3}$. For actions with only one successor, we do not depict the transition probability 1.

Several works have extended the original paper [39] already: the ideas have been lifted to a setting with limited information [6], with multiple objectives [4] and to concurrent stochastic games [30]. There also is an improved version of the algorithm based on *global* propagation of the over-approximation [52].

Organization of the paper. Section 2 introduces the basic notions and revises value iteration. Section 3 explains the idea of our approach on an example. Section 4 first characterizes the sub-graphs of the game which cause the non-convergence. Then it introduces the concept of *simple end components* and finally shows how to have a converging algorithm using a special treatment of these components. Section 5 describes the learning-based algorithm, Section 6 discusses experimental results and Section 7 concludes.

2. Preliminaries

2.1. Stochastic games

A probability distribution on a finite set X is a mapping $\delta\colon X \to [0,1]$, such that $\sum_{x\in X} \delta(x) = 1$. The set of all probability distributions on X is denoted by $\mathcal{D}(X)$. We now define stochastic games, which are often referred to simple stochastic games or turn-based stochastic two-player games with a reachability objective. As opposed to the notation of e.g. [24], we do not have special stochastic nodes, but rather a probabilistic transition function.

Definition 1 *(SG).* A *stochastic game (SG)* is a tuple $(S, S_\square, S_\bigcirc, \mathsf{s}_0, \mathsf{A}, \mathsf{Av}, \delta)$ where S is a finite set of *states* partitioned[3] into the sets $S_\square$ and $S_\bigcirc$ of states of the player *Maximizer* and *Minimizer*[4] respectively, $\mathsf{s}_0 \in S$ is the *initial* state, A is a finite set of *actions*, $\mathsf{Av}\colon S \to 2^{\mathsf{A}}$ assigns to every state a set of *available* actions, and $\delta\colon S \times \mathsf{A} \to \mathcal{D}(S)$ is a *transition function* that given a state s and an action $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$ yields a probability distribution over *successor* states.

A *Markov decision process (MDP)* is a special case of an SG where $S_\bigcirc = \emptyset$, and a Markov chain (MC) is a special case of an MDP, where for all $\mathsf{s} \in S : \left|\mathsf{Av}(\mathsf{s})\right| = 1$.

We assume that SGs are non-blocking, so for all states s we have $\mathsf{Av}(\mathsf{s}) \neq \emptyset$. For a state s and an available action $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$, we denote the set of successors by $\mathsf{Post}(\mathsf{s}, \mathsf{a}) := \{\mathsf{s}' \mid \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') > 0\}$. Finally, for any set of states $T \subseteq S$, we use $T_\square$ and $T_\bigcirc$ to denote the states in T that belong to Maximizer and Minimizer, whose states are drawn in the figures as $\square$ and $\bigcirc$, respectively.

Since we consider the reachability objective, in addition to an SG we require a set of target states $F \subseteq S$. As it is irrelevant what happens after reaching a target state, we assume that all of them are absorbing, i.e. they only have one action that is a self-loop with probability 1.

An example of an SG is given in Fig. 1.

2.2. Semantics: paths and strategies

The semantics of SGs is given in the usual way, see e.g. [25], by means of strategies, the induced Markov chain and the respective probability space, as follows. An *infinite path* ρ is an infinite sequence $\rho = \mathsf{s}_0\mathsf{a}_0\mathsf{s}_1\mathsf{a}_1 \dots \in (S \times \mathsf{A})^\omega$, such that for every $i \in \mathbb{N}$, $\mathsf{a}_i \in \mathsf{Av}(\mathsf{s}_i)$ and $\mathsf{s}_{i+1} \in \mathsf{Post}(\mathsf{s}_i, \mathsf{a}_i)$. *Finite paths* are defined analogously as elements of $(S \times \mathsf{A})^* \times S$.

As this paper deals with the reachability objective, we can restrict our attention to memoryless strategies, which are optimal for this objective [24]. A *strategy* of Maximizer, respectively Minimizer, is a function $\sigma\colon S_\square \to \mathcal{D}(\mathsf{A})$, respectively $S_\bigcirc \to \mathcal{D}(\mathsf{A})$, such that $\sigma(\mathsf{s}) \in \mathcal{D}(\mathsf{Av}(\mathsf{s}))$ for all s. We call a strategy *deterministic* if it maps to Dirac distributions only, i.e. if it plays a single action surely. Although deterministic strategies suffice, we still allow randomizing strategies, because they are needed for the learning-based algorithm later on. A pair (σ, τ) of strategies of Maximizer and Minimizer induces a Markov chain $\mathsf{G}^{\sigma,\tau}$ where the transition probabilities are defined as $\delta(\mathsf{s}, \mathsf{s}') = \sum_{\mathsf{a}\in\mathsf{Av}(\mathsf{s})} \sigma(\mathsf{s}, \mathsf{a}) \cdot \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}')$ for states of Maximizer and analogously for states of Minimizer, with σ replaced by τ. The Markov chain induces a unique probability distribution $\mathbb{P}^{\sigma,\tau}_{\mathsf{s},\mathsf{G}}$ over measurable sets of infinite paths [7, Ch. 10].

[3] I.e., $S_\square \subseteq S$, $S_\bigcirc \subseteq S$, $S_\square \cup S_\bigcirc = S$, and $S_\square \cap S_\bigcirc = \emptyset$.

[4] The names are chosen, because Maximizer maximizes the probability of reaching a given target state, and Minimizer minimizes it.

We write $\Diamond F = \{\rho \mid \rho = \mathsf{s}_0\mathsf{a}_0\mathsf{s}_1\mathsf{a}_1 \cdots \in (S \times \mathsf{A})^\omega \wedge \exists i \in \mathbb{N}.\ \mathsf{s}_i \in F\}$ to denote the (measurable) set of all paths which eventually reach the target states F.

For each $\mathsf{s} \in S$, we define the *value* in s as

$$\mathsf{V}_\mathsf{G}(\mathsf{s}) := \sup_\sigma \inf_\tau \mathbb{P}^{\sigma,\tau}_{\mathsf{s},\mathsf{G}}(\Diamond F) = \inf_\tau \sup_\sigma \mathbb{P}^{\sigma,\tau}_{\mathsf{s},\mathsf{G}}(\Diamond F),$$

where the equality follows from [24]. We omit the G in the subscript when it is clear from context.

To compute the value, we need take special care of *end components* (ECs). Intuitively, an EC is a subset of states of the SG, where the game can remain forever; i.e. given certain strategies of both players, there is no positive probability to exit the EC to some other state. An EC corresponds to a bottom strongly connected component in a Markov chain induced by some pair of strategies.

To define ECs formally, we first introduce the following notation. Given a set of states $T \subseteq S$ and a state $\mathsf{s} \in T$, we say that (s, a) exits T if we have $\mathsf{s} \in T, \mathsf{a} \in \mathsf{Av}(\mathsf{s})$ and $\mathsf{Post}(\mathsf{s}, \mathsf{a}) \nsubseteq T$.

Definition 2 *(EC).* A non-empty set $T \subseteq S$ of states is an *end component (EC)* if there is a non-empty set $B \subseteq \bigcup_{\mathsf{s} \in T} \mathsf{Av}(s)$ of actions such that

1. for each $\mathsf{s} \in T, \mathsf{a} \in B \cap \mathsf{Av}(\mathsf{s})$ we do *not* have (s, a) exits T,
2. for each $\mathsf{s}, \mathsf{s}' \in T$ there is a finite path $\mathsf{w} = \mathsf{s}\mathsf{a}_0 \ldots \mathsf{a}_n\mathsf{s}' \in (T \times B)^* \times T$, i.e. the path stays inside T and only uses actions in B.

An end component T is a *maximal end component (MEC)* if there is no other end component T' such that $T \subseteq T'$. Note that the MECs of an SG are equivalent to the MECs in an MDP with both players unified,[5] as the definition only depends on the existence of the actions. Thus the set of MECs of an SG G, denoted by MEC(G), can be computed in polynomial time using the standard algorithm for MDPs, e.g. [17].

2.3. (Bounded) value iteration

Let $\mathcal{Z}$ be the set of states that cannot reach any state in F (called $\mathcal{Z}$, because they have value **z**ero). It can be computed, for example, by a reverse breadth-first-search from all states in F.

The value function V satisfies the following system of equations, referred to as the *Bellman equations*:

$$\mathsf{V}(\mathsf{s}) := \begin{cases} 1 & \text{if } \mathsf{s} \in F \\ 0 & \text{if } \mathsf{s} \in \mathcal{Z} \\ \max_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{V}(\mathsf{s}, \mathsf{a}) & \text{if } \mathsf{s} \in S_\square \\ \min_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{V}(\mathsf{s}, \mathsf{a}) & \text{if } \mathsf{s} \in S_\bigcirc \end{cases} \tag{1}$$

where[6]

$$\mathsf{V}(\mathsf{s}, \mathsf{a}) := \sum_{s' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{V}(\mathsf{s}') \tag{2}$$

Moreover, V is the *least* solution to the Bellman equations, see e.g. [18]. To compute the value of V for all states in an SG, one can thus utilize the iterative approximation method *value iteration (VI)* as follows: we start with a lower bound function $\mathsf{L}_0 \colon S \to [0, 1]$ such that $\mathsf{L}_0(\mathsf{s}) = 1$ for target states $\mathsf{s} \in F$ and for all other states $\mathsf{s} \in S \setminus F$, we have $\mathsf{L}_0(\mathsf{s}) = 0$. Then we repetitively apply Bellman updates according to Equations (3) and (4)

$$\mathsf{L}_n(\mathsf{s}, \mathsf{a}) := \sum_{s' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{L}_{n-1}(\mathsf{s}') \tag{3}$$

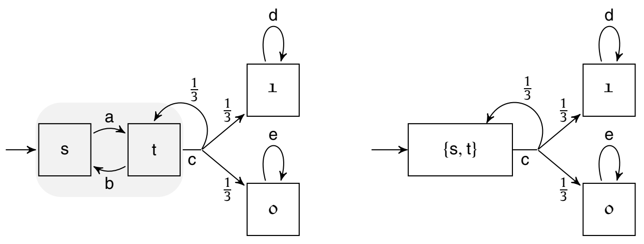
$$\mathsf{L}_n(\mathsf{s}) := \begin{cases} \max_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{L}_n(\mathsf{s}, \mathsf{a}) & \text{if } \mathsf{s} \in S_\square \\ \min_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{L}_n(\mathsf{s}, \mathsf{a}) & \text{if } \mathsf{s} \in S_\bigcirc \end{cases} \tag{4}$$

until convergence. Note that convergence may happen only in the limit even for such a simple game (even an MDP) as in Fig. 2 on the left. The sequence is continuous and monotonic, because all involved operations – summation and multiplication – are continuous and monotonic. Moreover, at all times it is a *lower* bound on V, i.e. $\mathsf{L}_i(\mathsf{s}) \leq \mathsf{V}(\mathsf{s})$ for all $\mathsf{s} \in S$, and the least fixpoint satisfies $\mathsf{L}^* := \lim_{n\to\infty} \mathsf{L}_n = \mathsf{V}$, see e.g. [18].

[5] Intuitively, this is also why it does not suffice to consider MECs in an SG, and we need to introduce the stronger notion of simple end component.
[6] Throughout the paper, for any function $f : S \to [0, 1]$ we overload the notation and also write $f(\mathsf{s}, \mathsf{a})$ meaning $\sum_{s' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot f(\mathsf{s}')$.

Algorithm 1 Bounded value iteration algorithm.

```
1: procedure BVI(precision $\varepsilon > 0$)
2:     for $s \in S$ do # Initialization
3:         $L()\ (s) = 0$   # Lower bound
4:         $U(s) = 1$   # Upper bound
5:     for $s \in F$ do $L(s) = 1$ # Value of targets is determined a priori
6:     for $s \in \mathfrak{Z}$ do $U(s) = 0$ # Value of sinks is determined a priori
7:     repeat
8:         UPDATE(L, U)        # Bellman updates or their modification
9:     until $U(s_0) - L(s_0) < \epsilon$ # Guaranteed error bound
```



i	$L_i(\{s,t\})$	$G_i(\{s,t\})$
0	0	1
1	$\frac{1}{3}$	$\frac{2}{3}$
2	$\frac{4}{9}$	$\frac{5}{9}$
3	$\frac{13}{27}$	$\frac{14}{27}$

Fig. 2. Left: an MDP (as special case of SGs) where BVI does not converge due to the grayed EC. Middle: the same MDP where the EC is collapsed, making BVI converge. Right: the approximations illustrating the convergence of the MDP in the middle.

We achieve our goal when $L(s_0)$ is ε-close to the value $V(s_0)$. Then we have an ε-approximation of the value of the SG $V(s_0)$ and can extract a corresponding pair of ε-optimal strategies for both players, i.e. strategies σ, τ such that $\mathbb{P}^{\sigma,\tau}_{s_0}(\Diamond F)$ is ε-close to $V(s_0)$. To the best of our knowledge, no explicit description of how to obtain strategies from a vector of (an ε-approximation of the) values exists, thus we provide it in Appendix A.

Unfortunately, there is no known stopping criterion which estimates how close the current under-approximation is to the value. The current tools stop when the difference between two successive approximations is smaller than a certain threshold, which can lead to arbitrarily wrong results [33].

For the special case of MDPs, it has been suggested to also compute the greatest fixpoint [51] and thus an *upper* bound as follows. The function $G: S \to [0,1]$ is initialized for all states $s \in S$ as $G_0(s) := 1$ except for states $s \in \mathfrak{Z}$ with no path to a target where $G_0(s) := 0$. Then we repetitively apply updates (3) and (4), where L is replaced by G. The resulting sequence G_n again is monotonic and continuous, and moreover provides an upper bound on V. The greatest fixpoint $G^* := \lim_n G_n$ is the greatest solution to the Bellman equations on $[0,1]^S$. This approach is called *bounded value iteration (BVI)* (or *bounded real-time dynamic programming (BRTDP)* [51,11] or *interval iteration* [33]).

If $L^* = G^*$ then they are both equal to V and we say that *BVI converges*. BVI is guaranteed to converge in MDPs if the only ECs are trivial, i.e. the self-loops of target states in F or ECs with no path to a target in $\mathfrak{Z}$. Otherwise, if there are non-trivial ECs, BVI does not converge. For MDPs, the authors of [11,33] independently introduced the solution to "collapse" all ECs, i.e. merge all states of an EC into one and preserve only the actions exiting the EC. Computing the greatest fixpoint on the modified MDP results in another sequence U_i of upper bounds on V, converging to $U^* := \lim_n U_n$, and it holds that $U^* = V$ and BVI converges. The next section illustrates the difficulty and the solution through collapsing on an example.

Collapsing can be extended to SGs, but it does not solve the problem for arbitrary SGs. This is also illustrated in an example in the next section. In certain cases, it can still be used as a heuristic to speed up the algorithm, see Section 6.2.

In summary, all versions of BVI discussed so far and later on in the paper follow the pattern of Algorithm 1. In the naive version, UPDATE just performs the Bellman update on L and U according to Equations (3) and (4). For a general MDP, U does not converge to V, but to G^*, and thus the termination criterion may never be met if $G^*(s_0) - V(s_0) > 0$. If the ECs are collapsed in pre-processing then U converges to V. For the general case of SGs, the collapsing approach fails and this paper provides another version of BVI where U converges to V, based on a more detailed structural analysis of the game.

3. Example: why over-approximations do not converge

In this section, we illustrate why naive BVI does not converge on SGs and sketch our solution on a few examples. We face two problems: **(i)** states in ECs promise overly high values, even in MDPs. **(ii)** ECs in SGs do not belong to a single player and thus states in them can have different values.

We always denote by G the sequence converging to the greatest solution of the Bellman equations (monotonically from above), while we use U for any sequence over-approximating V that one or another BVI algorithm suggests.

(i) We illustrate the issue that arises already for the special case of MDPs. Consider the MDP of Fig. 2 on the left. Although $V(s) = V(t) = 0.5$, we have $G_i(s) = G_i(t) = 1$ for all i. Indeed, the upper bound for t is always updated as the maximum of $G_i(t, c)$ and $G_i(t, b)$. Although $G_i(t, c)$ decreases over time, $G_i(t, b)$ remains the same, namely equal to $G_i(s)$, which in turn

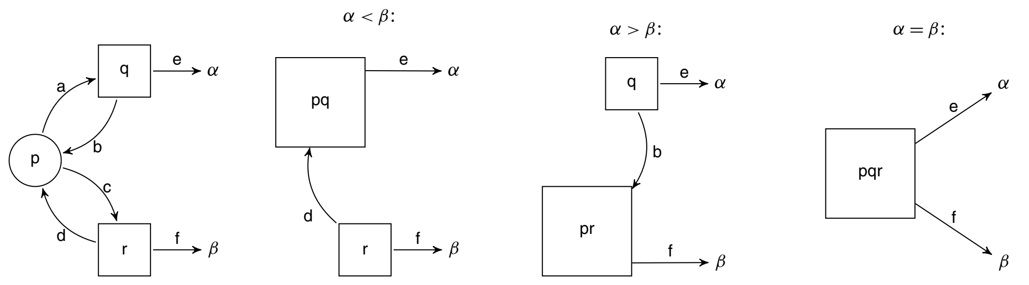


Fig. 3. Collapsing ECs in SGs may lead to incorrect results. The Greek letters on the leaving arrows denote the values of the exiting actions. Left figure: an example of an EC that can be collapsed in different ways depending on the relationship of α and β. Right three figures: correct collapsing in the different cases. In contrast to MDPs, some actions of the EC exiting the collapsed part have to be removed.

remains equal to $\mathsf{G}_i(\mathsf{s},\mathsf{a}) = \mathsf{G}_i(\mathsf{t})$. This cyclic dependency lets both s and t remain in an illusion that the value of the other one is 1.

The solution for MDPs is to remove this cyclic dependency by collapsing all MECs into singletons and removing the resulting purely self-looping actions. The middle of Fig. 2 shows the MDP after collapsing the EC $\{\mathsf{s},\mathsf{t}\}$. This turns the MDP into a stopping one, where the states in F or $\mathcal{Z}$ are reached with probability 1 under any strategy. In such MDPs, there is a unique solution to the Bellman equations. Therefore, the greatest fixpoint is equal to the least one and thus to V.

(ii) We illustrate the issues that additionally arise for general SGs. It turns out that the collapsing approach can be extended only to games where all states of each EC belong to one player only [58]. In this case, both Maximizer's and Minimizer's ECs are collapsed the same way as in MDPs.

However, when both players are present in an EC, then collapsing may not solve the issue. Consider the SG of Fig. 3. Here α and β represent the values of the respective actions.[7] There are three cases.

- First, let $\alpha < \beta$. If the bounds converge to these values we eventually observe $\mathsf{G}_i(q,e) < \mathsf{L}_i(r,f)$ and learn the induced inequality. Since p is a Minimizer's state it will never pick the action leading to the greater value of β. Therefore, we can safely merge p and q, and remove the action leading to r, as shown in the second sub-figure
- Second, if $\alpha > \beta$, p and r can be merged in an analogous way, as shown in the third sub-figure
- Third, if $\alpha = \beta$, both previous solutions as well as collapsing all three states as in the fourth sub-figure is possible. However, since the approximations may only converge to α and β in the limit, we may not know in finite time which of these cases applies and thus cannot decide for any of the collapses.

Sketching the solution. Consequently, the approach of collapsing is not applicable in general. In order to ensure BVI convergence, we suggest a different method, which we call *deflating*. It does not involve changing the state space, but rather decreasing the upper bound U_i to the least value that is currently provable (and thus still correct). To this end, we analyse the *exiting actions*, i.e. actions with successors outside of the EC, for the following reason: if the play stays in the EC forever, the target is never reached and Minimizer wins. Therefore, Maximizer has to pick some exiting action to avoid staying in the EC.

For the EC with the states s and t in Fig. 2, the only exiting action is c. In this example, since c is the only exiting action, $\mathsf{U}_i(\mathsf{t},\mathsf{c})$ is the highest possible upper bound that the EC can achieve. Thus, by decreasing the upper bound of all states in the EC to that number, we still have a safe upper bound. Moreover, with this modification BVI converges in this example, intuitively because now the upper bound of t depends on action c as it should. We choose the name "deflating" to evoke decreasing the overly high "pressure" in the EC until it equalizes with the actual "pressure" outside.

However, this does not lead to convergence of BVI yet. Consider for instance the SG in Fig. 3. It is correct to decrease the upper bound to the maximal exiting one, i.e. $\max\{\hat{\alpha},\hat{\beta}\}$, where $\hat{\alpha} := \mathsf{U}_i(\mathsf{a}), \hat{\beta} := \mathsf{U}_i(\mathsf{b})$ are the current approximations of α and of β, but this itself does not ensure BVI convergence. Indeed, if for instance $\hat{\alpha} < \hat{\beta}$ then deflating all states to $\hat{\beta}$ is not tight enough, as values of p and q can even be bounded by $\hat{\alpha}$. In fact, we have to find a certain sub-EC that corresponds to $\hat{\alpha}$, in this case $\{\mathsf{p},\mathsf{q}\}$ and set all its upper bounds to $\hat{\alpha}$. We define and compute these sub-ECs in the next section.

In summary, the general structure of our convergent BVI algorithm is to produce a sequence U by applying Bellman updates and occasionally finding the relevant sub-ECs and deflating them. The main technical challenge is that states in an EC in SGs can have different values, in contrast to the case of MDPs.

4. Convergent over-approximation

In Section 4.1, we characterize SGs where Bellman equations have more than one fixpoint. Based on this analysis, subsequent sections show how to alter the procedure computing the sequence G_i over-approximating V so that the resulting

[7] Precisely, they represent a probabilistic branching with probability α (or β) to 1 and with the remaining probability to 0. For clarity, we omit this branching and depict only the value.

tighter sequence U_i still over-approximates V, but also converges to V, which ensures convergence of BVI. Section 5 presents the learning-based variant of our BVI.

4.1. Bloated end components cause non-convergence

As we have seen in the example of Fig. 3, BVI generally does not converge due to ECs with a particular structure of the exiting actions. The analysis of ECs relies on the extremal values that can be achieved by exiting actions (in the example, α and β). Given the value function V or just its current over-approximation U_i, we define the most profitable exiting action for Maximizer (denoted by $\square$) and Minimizer (denoted by $\bigcirc$) as follows.

Definition 3 *(*$\mathsf{bestExit}$*).* Given a set of states $T \subseteq S$ and a function $f : S \to [0,1]$ (and its equivalent on state-action pairs, see Footnote 6), the best exit according to f from T of Maximizer and Minimizer, respectively, is defined as

$$\mathsf{bestExit}_f^{\square}(T) := \max_{\substack{\mathsf{s}\in T_\square \\ (\mathsf{s},\mathsf{a})\text{ exits } T}} f(\mathsf{s},\mathsf{a})$$

$$\mathsf{bestExit}_f^{\bigcirc}(T) := \min_{\substack{\mathsf{s}\in T_\bigcirc \\ (\mathsf{s},\mathsf{a})\text{ exits } T}} f(\mathsf{s},\mathsf{a})$$

with the convention that $\max_\emptyset = 0$ and $\min_\emptyset = 1$.

Example 1 *(Non-convergence).* In the example of Fig. 3 on the left with $T = \{\mathsf{p},\mathsf{q},\mathsf{r}\}$ and $\alpha < \beta$, we have $\mathsf{bestExit}_\mathsf{V}^{\square}(T) = \beta$ and $\mathsf{bestExit}_\mathsf{V}^{\bigcirc}(T) = 1$. It is due to $\beta < 1$ that BVI does not converge here. We generalize this in the following lemma. $\triangle$

Lemma 1 *(Multiple solutions). Let $T \subseteq S \setminus F$ be an EC. For every m satisfying $\mathsf{bestExit}_\mathsf{V}^{\square}(T) \leq m \leq \mathsf{bestExit}_\mathsf{V}^{\bigcirc}(T)$, there is a solution $\mathsf{U}: S \to [0,1]$ to the Bellman equations, which for all states in T is greater than m, and for at least one state is equal to m.*

Proof. Intuitively, this lemma states the following: given an over-approximation that is greater than what T can actually achieve, but smaller than what Minimizer can certainly restrict to, Bellman updates (value iteration) will be stuck at the over-approximation. However, constructing the exact over-approximation where the iteration is stuck is technically involved.

Let m satisfy $\mathsf{bestExit}_{\mathsf{V}_\mathsf{G}}^{\square}(T) \leq m \leq \mathsf{bestExit}_{\mathsf{V}_\mathsf{G}}^{\bigcirc}(T)$. Note that we explicitly added G in the index, as we will later construct a modified SG G' and thus have to make clear to which SG we are referring to. We first construct a function from states to real numbers, then prove that it is greater than m for all states in T and afterwards that it is a solution to the Bellman equations, i.e. a fixpoint of applying Equations (3) and (4). Lastly, we show that it assigns exactly m to at least one state in T.

We obtain a modified SG G' from the given SG G as follows: for every Maximizer state in T, add an action that leads to a target state with probability m and to a sink state with probability $(1-m)$. Intuitively, this allows all Maximizer states to actually achieve the over-approximation m in every state. Further, we have to treat the corner case of ECs that only consist of Minimizer states. If there is an EC $T' \subseteq T_\bigcirc$, then all states in T' also get an action leading to a target with m and to a sink with the $1-m$; moreover, all other actions of these states are removed. Let $\mathsf{U} := \mathsf{V}_{\mathsf{G}'}$ be the value function of G'.

We now prove that for all states $\mathsf{s} \in T$, it holds that $\mathsf{U}(\mathsf{s}) = \mathsf{V}_{\mathsf{G}'}(\mathsf{s}) \geq m$. If s is a Maximizer state, then it has the newly added action with value m. Thus, the value in the modified SG[8] of every $\mathsf{s} \in T_\square$ is at least m (but possibly higher[9]).

If s is a Minimizer state, we have to treat the corner case of an EC consisting only of Minimizer states. If s is in an EC $T' \subseteq T_\bigcirc$, then by construction it only has an action with value m, and thus its value is (at least) m. Finally, we consider the case of a Minimizer state which is not part of an EC $T' \subseteq T_\bigcirc$. We know that for all leaving actions (actions $\mathsf{a}_\ell \in \mathsf{Av}(\mathsf{s})$ with $(\mathsf{s},\mathsf{a}_\ell)$ exits T) it holds that

$$\mathsf{V}_{\mathsf{G}'}(\mathsf{s},\mathsf{a}_\ell) \geq \mathsf{V}_\mathsf{G}(\mathsf{s},\mathsf{a}_\ell) \geq \mathsf{bestExit}_{\mathsf{V}_\mathsf{G}}^{\bigcirc} \geq m$$

The first inequality holds, because in G' we only added more possibility to reach the target, potentially increasing the value; note in particular that ECs with only Minimizer states actually have a value of 0. The second inequality follows from the definition of $\mathsf{bestExit}$ and the third from the choice of m.

[8] For the remainder of this sub-proof of $\mathsf{V}_{\mathsf{G}'}(\mathsf{s}) \geq m$, we omit the phrase "in the modified SG" for ease of notation.

[9] In the original paper [39], this lemma was wrong, since it claimed that the solution has to be constant on T. We provide a counter example. Consider the leftmost EC from Fig. 3. Let action e lead to a target with probability $\frac{1}{2}$ and to r with probability $\frac{1}{2}$; let $\beta = \frac{1}{2}$. Then we have $\mathsf{V}(r) = \mathsf{V}(r, f) = \frac{1}{2}$ and $\mathsf{V}(q) = \mathsf{V}(q, e) = \frac{1}{2} + \frac{1}{2}\cdot\frac{1}{2} = \frac{3}{4}$. Note that the Minimizer state p prefers to go down using action c, as $\mathsf{V}(r) < \mathsf{V}(q)$. The best exit is $\mathsf{bestExit}_\mathsf{V}^{\square}(T) = \frac{3}{4}$. However, picking $m = \frac{3}{4}$ and setting $\mathsf{U}(q) = \mathsf{U}(p) = \mathsf{U}(r) = m$ is not a fixpoint of Bellman updates, because applying an update on q we have $\mathsf{U}(q, e) = \frac{1}{2} + \frac{1}{2}\cdot m = \frac{7}{8} > m$. This is why we adjusted the statement of the lemma to only require equality with m in at least one state in T, not all of them. This weaker claim still suffices to prove Corollary 1.

Now we consider staying actions of s. Let X be the set of states that are reached by following Minimizer's optimal staying actions starting from s. Formally, X is initialized to $\{\mathsf{s}\}$ and then for all $\mathsf{s}' \in X_{\bigcirc}$ we recursively add $\underset{\substack{\mathsf{a} \in \mathsf{Av}(\mathsf{s}) \\ \neg(\mathsf{s},\mathsf{a})\,\text{exits}\,T}}{\arg\min} \; \mathsf{Post}(\mathsf{s},\mathsf{a})$. As T is finite and only states from T can be added (since we consider staying actions), X will converge. We analyze all cycles in X. If there is an EC $X' \subseteq X_{\bigcirc}$ consisting only of Minimizer states, we already proved that its value is m. All other cycles have at least some probability to go to a Maximizer state, and we already proved that all Maximizer states in T have value at least m. So s has no other choice but to almost surely reach a Maximizer state or an EC consisting only of Minimizer states, both of which have value of at least m. Thus, for all actions of s we have proven that their value in the modified SG is at least m.

Now we show that U is a fixpoint applying Bellman updates in the original SG G. If in G' a state depended on an action that is also available in G, applying one Bellman update will not change its value; this is because U was a fixpoint of applying Bellman updates in G'. Thus we only have to prove that all states that depended on an action added in the modified SG do not change their value upon applying a Bellman update. Firstly consider a Maximizer state $\mathsf{s} \in T_{\square}$. If $\mathsf{U}(\mathsf{s}) > m$, it would not depend on the newly added action, and hence we know that $\mathsf{U}(\mathsf{s}) = m$. As T is an EC, s has a staying action a_s with $\mathsf{U}(\mathsf{s}, \mathsf{a}_s) \geq m$, because all states in T have U of at least m. We also know that $\mathsf{U}(\mathsf{s}, \mathsf{a}_s) = m$, because $\mathsf{U}(\mathsf{s}) = m$, and Maximizer would have used an action yielding more than m if it was available. So for every Maximizer state that previously used one of the newly added actions, it is optimal to choose a staying action and the estimate after one Bellman update is still m. Secondly consider a Minimizer state $\mathsf{s} \in T_{\bigcirc}$. We only added an action for those if they were part of some EC $T' \subseteq T_{\bigcirc}$. For all states $\mathsf{s}' \in T'$ we have $\mathsf{U}(\mathsf{s}') = m$. Hence, by playing actions that stay in T', Minimizer can achieve a U of m. It is the optimal choice for Minimizer to do this, because by assumption all exiting actions are worse (higher number) than m. In conclusion, applying a Bellman update on U in G will not change U, and thus it is a fixpoint.

Finally we prove that at least one state $\mathsf{s} \in T$ has $\mathsf{U}(\mathsf{s}) = m$. We already know $\mathsf{U}(\mathsf{s}) \geq m$ for all $\mathsf{s} \in T$. Assume for contradiction that $\mathsf{U}(\mathsf{s}) > m$ for all $\mathsf{s} \in T$. Then no state would depend on one of the newly added actions. Thus, the values in G and G' would be equal, and we would also have $\mathsf{V}_{\mathsf{G}}(\mathsf{s}) > m$ for all $\mathsf{s} \in T$. This is a contradiction, because the value of states in an EC can be at most $\mathsf{bestExit}^{\square}_{\mathsf{V}}(T) \leq m$. So we conclude that there exists a $\mathsf{s} \in T$ with $\mathsf{U}(\mathsf{s}) = m$. □

Corollary 1. *If* $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T) > \mathsf{bestExit}^{\square}_{\mathsf{V}}(T)$ *for some EC* T*, then* $\mathsf{G}^* \neq \mathsf{V}$.

Proof. There are m_1, m_2 such that $\mathsf{bestExit}^{\square}_{\mathsf{V}}(T) < m_1 < m_2 < \mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T)$. By Lemma 1, for both there exists a solution to the Bellman equations U_1 and U_2. The solutions are distinct, as there always exists at least one state where $\mathsf{U}_1(s) = m_1 < m_2 \leq \mathsf{U}_2(s)$.

Thus, there exist multiple fixpoints of Bellman updating. In particular, $\mathsf{G}^* > \mathsf{L}^* = \mathsf{V}$, and BVI does not converge. □

In accordance with our intuition that ECs satisfying the above inequality should be deflated, we call them bloated.

Definition 4 *(BEC).* An EC T is called a *bloated end component (BEC)*, if $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T) > \mathsf{bestExit}^{\square}_{\mathsf{V}}(T)$.

Example 2 *(BEC).* In the example of Fig. 3 on the left with $\alpha < \beta$, the ECs $\{\mathsf{p}, \mathsf{q}\}$ and $\{\mathsf{p}, \mathsf{q}, \mathsf{r}\}$ are BECs. △

Example 3 *(Corner cases of* $\mathsf{bestExit}$*).* If an EC T has no exiting actions of Minimizer (or no Minimizer's states at all, as in an MDP), then $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T) = 1$ (the case with $\min_{\emptyset}$). Hence all numbers between $\mathsf{bestExit}^{\square}_{\mathsf{V}}(T)$ and 1 are a solution to the Bellman equations and $\mathsf{G}^*(s) = 1$ for all states $s \in T$.

Analogously, if Maximizer does not have any exiting action in T, then it holds that $\mathsf{bestExit}^{\square}_{\mathsf{V}}(T) = 0$ (the case with $\max_{\emptyset}$), T is a BEC and all numbers between 0 and $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T)$ are a solution to the Bellman equations. △

Note that in MDPs all ECs belong to one player, namely Maximizer. Consequently, all ECs are BECs except for ECs where Maximizer has an exiting action with value 1; all other ECs thus have to be collapsed (or deflated) to ensure BVI convergence in MDPs. Interestingly, all non-trivial ECs in MDPs are a problem, while in SGs through the presence of the other player some ECs can converge, namely if both players want to exit, which we illustrate in the subsequent example.

Example 4 *(Unproblematic ECs).* If in the example of Fig. 3 on the left p had an exiting action with value $\gamma < \min(\alpha, \beta)$, then Minimizer's best strategy is to pick the leaving action and ensure the smaller value. Consequently, Maximizer realizes that going to p is suboptimal, as both α and β are larger than γ. Thus all states depend on exiting actions, we have no cyclic dependencies and BVI converges. △

Our first theorem states that BECs are indeed the only obstacle for BVI convergence. To prove it, we need the following lemma.

Lemma 2. *If a set $T \subseteq S$ does not contain an EC, then there exists a state $\mathsf{s} \in T$, such that for all $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$ it holds that (s, a) exits T.*

Proof. Let $T \subseteq S$ be a set of states that does not contain an EC. Assume for contradiction that for all states $\mathsf{s} \in T$ there exists an action a, such that $\neg(\mathsf{s}, \mathsf{a})$ exits T. Then we can construct an EC as follows: let $T' = \{\mathsf{s}\}$ for some state $\mathsf{s} \in T$. Then, for all states in T' add all successors of the actions that do not exit T to T'. Repeat this until a fixpoint is reached. This must happen, as T is finite and the actions are not exiting, i.e. only have successors in T. Now compute the strongly connected components of T' with only the staying actions. There has to be a non-trivial strongly connected component, because T' is connected and finite, and every state has at least one outgoing edge to another state in T', so there exists a cycle. The strongly connected component forms an EC, which is a contradiction. Thus we know that for some state, all actions are leaving T. □

We will also sometimes use the contrapositive of this lemma, i.e. that if every state in a set T has at least one staying action, then T contains an EC.

Theorem 1 *(Convergence without BECs). If an SG contains no non-trivial BECs, i.e. no BECs in $S \setminus (F \cup \mathcal{Z})$, then $\mathsf{G}^* = \mathsf{V}$, i.e. value iteration from above converges to the value in the limit.*

Proof (*Structure*). We first give the general idea of the proof.

We assume towards a contradiction, that there is some state s with a positive difference $\mathsf{G}^*(\mathsf{s}) - \mathsf{V}(\mathsf{s}) > 0$. Then we will look at the set of states X that have the maximal difference and prove the following.

No state in X "depends on the outside". (5)

This means that for all states in X, the best action must not have successors outside of X. Best action means that it minimizes V for Minimizer states and maximizes G^* for Maximizer states.

Then, using a case distinction over the possible graph structures of X, we show that the only way to satisfy Statement (5) is that we have an EC $Z \subseteq X$ where the best action for every state is not exiting Z. However, this implies that Z is a BEC (and in fact even a *simple end component*, which will be introduced in the next section). This is a contradiction to the assumption that the game is BEC-free, and thus proves our goal.

Proof (*Complete*). We denote the difference in a state by $\Delta(\mathsf{s}) := \mathsf{G}^*(\mathsf{s}) - \mathsf{V}(\mathsf{s})$. We also use $\Delta(\mathsf{s}, \mathsf{a})$, see Footnote 6. Assume for contradiction there exists a state $\mathsf{s} \in S$ with positive difference $\Delta(\mathsf{s}) > 0$.

Let $X := \{\mathsf{s} \in S \mid \Delta(\mathsf{s}) = \max_{\mathsf{s} \in S} \Delta(\mathsf{s})\}$ denote the set of all states with maximal difference. Note that by assumption of there being a state with positive difference, X is non-empty and for all $\mathsf{s} \in X$ we have $\Delta(\mathsf{s}) > 0$. By definition, we require $\mathsf{V}(\imath) = \mathsf{G}^*(\imath) = 1$ for every $\imath \in \mathsf{F}$. Hence, $\Delta(\imath) = 0$ and thus, $\mathsf{F} \cap X = \emptyset$. Analogously, it holds that $\mathcal{Z} \cap X = \emptyset$. Hence we know that $X \subseteq S \setminus (F \cup \mathcal{Z})$.

We will now prove that no state in X "depends on the outside", i.e.

$$\forall (\mathsf{s}, \mathsf{a}) \, \text{exits} \, X : \begin{cases} \mathsf{G}^*(\mathsf{s}, \mathsf{a}) < \mathsf{G}^*(\mathsf{s}) & \text{if } \mathsf{s} \in X_\Box \\ \mathsf{V}(\mathsf{s}, \mathsf{a}) > \mathsf{V}(\mathsf{s}) & \text{if } \mathsf{s} \in X_\bigcirc \end{cases} \tag{5}$$

To prove Statement (5), first observe the following:

$$\forall (\mathsf{s}, \mathsf{a}) \, \text{exits} \, X : \Delta(\mathsf{s}, \mathsf{a}) < \Delta(\mathsf{s}). \tag{6}$$

This holds, because if (s, a) exits X, then $\exists t \in \mathsf{Post}(\mathsf{s}, \mathsf{a}) \setminus X$. Since $t \notin X$, $\Delta(t) < \Delta(\mathsf{s})$. Then the following chain of equations proves Statement (6):

$$\begin{aligned}
\Delta(\mathsf{s}, \mathsf{a}) &:= \mathsf{G}^*(\mathsf{s}, \mathsf{a}) - \mathsf{V}(\mathsf{s}, \mathsf{a}) && \text{(Definition of } \Delta) \\
&= \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot (\mathsf{G}^*(\mathsf{s}') - \mathsf{V}(\mathsf{s}')) && \text{(Definition of V and G, pulling together the sums)} \\
&= \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \Delta(\mathsf{s}') && \text{(Definition of } \Delta) \\
&< \Delta(\mathsf{s})
\end{aligned}$$

The last inequality holds, since $\delta(\mathsf{s}, \mathsf{a}, t) > 0$ and $\Delta(t) < \Delta(\mathsf{s})$, so there is one summand that is smaller than $\Delta(\mathsf{s})$; also, $\Delta(\mathsf{s})$ is the maximum difference, so there can be no larger summand to make up for the loss.

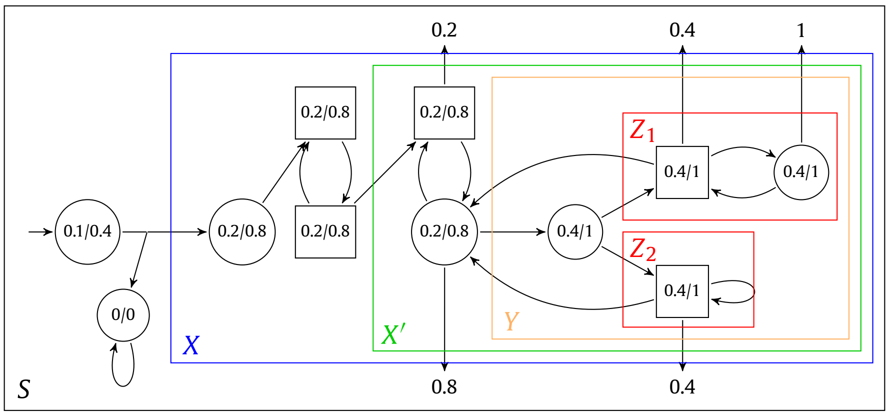


Fig. 4. An example of an SG where all the sets in the proof of Theorem 1 are different. Numbers on exiting arrows denote values, see Footnote 7. Every state is inscribed with V/G.

Now we can prove Statement (5) with the following chain of equations:

$$\begin{aligned} \mathsf{G}^*(\mathsf{s},\mathsf{a}) - \mathsf{V}(\mathsf{s},\mathsf{a}) &= \Delta(\mathsf{s},\mathsf{a}) && \text{(Definition of } \Delta\text{)} \\ &< \Delta(\mathsf{s}) && \text{(Using Statement (6))} \\ &= \mathsf{G}^*(\mathsf{s}) - \mathsf{V}(\mathsf{s}) && \text{(Definition of } \Delta\text{)} \\ &\leq \begin{cases} \mathsf{G}^*(\mathsf{s}) - \mathsf{V}(\mathsf{s},\mathsf{a}) & \text{if } \mathsf{s} \in S_\Box \\ \mathsf{G}^*(\mathsf{s},\mathsf{a}) - \mathsf{V}(\mathsf{s}) & \text{if } \mathsf{s} \in S_\bigcirc \end{cases} \end{aligned}$$

The last inequality holds since for every available action a and for $\mathsf{s} \in S_\Box : \mathsf{V}(\mathsf{s}) \geq \mathsf{V}(\mathsf{s},\mathsf{a})$, as Maximizer uses the maximum action, and dually for $\mathsf{s} \in S_\bigcirc : \mathsf{G}^*(\mathsf{s}) \leq \mathsf{G}^*(\mathsf{s},\mathsf{a})$. Simplifying the inequation yields Statement (5).

Now we make a case distinction over the possible graph structures of X and use Statement (5) and Lemma 2 to derive a contradiction.

- If X does not contain an EC,
  then by Lemma 2 there exists a state $\mathsf{s} \in X$, such that for all $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$ we have (s,a) exits X. Thus, this state "depends on the outside", which by Statement (5) yields a contradiction. Formally, if $\mathsf{s} \in X_\Box$, we have that $\forall \mathsf{a} \in \mathsf{Av}(\mathsf{s}) : \mathsf{G}^*(\mathsf{s},\mathsf{a}) < \mathsf{G}^*(\mathsf{s})$, because all actions exit X and Statement (5). However, by definition $\mathsf{G}^*(\mathsf{s}) = \max_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{G}^*(\mathsf{s},\mathsf{a})$, so there has to be an equality for some action. The argument works analogously for a Minimizer state $\mathsf{s} \in X_\bigcirc$.
- If X contains an EC,
  we need to make further steps. States can have a large difference just by depending on a BEC. Our further steps will allow us to find a BEC inside X. See Fig. 4 for an example of a game where all the sets we introduce in the following are different.
  We now need the notion of bottom MEC. When talking about the whole game, bottom MECs are defined as MECs without leaving actions. However, we need the notion of bottom MEC *in* X. Formally, this requires that every successor of an action leaving the bottom MEC reaches a state outside of X with positive probability under all pairs of strategies. Intuitively, a bottom MEC in X is a MEC, such that after leaving it, there is no other MEC in X. They are computed by computing the MEC decomposition of X, ordering the MECs topologically and picking one at the end of a chain.
  Let $X' \subseteq X$ be a bottom MEC in X. X' exists, because by assumption X contains an EC, so there also is an EC that is bottom in X. If X' was a MEC at the end of the topological ordering of MECs in X, but not bottom, that would mean that there is a successor state s' of some action exiting X', such that there is a pair of strategies under which the play from s' surely remains in X. This would imply that s' can reach an EC inside X, which is a contradiction to the assumption that X' is at the end of the topological ordering of MECs in X.
  Let $m = \max_{\mathsf{s} \in X'} \mathsf{G}^*(\mathsf{s})$ be the maximum upper bound in X', and let $Y := \{\mathsf{s} \mid \mathsf{s} \in X' \wedge \mathsf{G}^*(\mathsf{s}) = m\}$ be the states with that maximum upper bound.
  $\forall \mathsf{s} \in Y, \exists \mathsf{a} \in \mathsf{Av}(\mathsf{s}) : \neg(\mathsf{s},\mathsf{a})$ exits Y, i.e. all states in Y have actions that stay in Y. We prove this by a case distinction over where the actions of Y can exit to. Let $\mathsf{s} \in Y$ be an arbitrary state. There has to be some action a_m with $\mathsf{G}^*(\mathsf{s},\mathsf{a}_m) = \mathsf{G}^*(\mathsf{s}) = m$. We say that action a_m exits X towards a set of states T, if $(\mathsf{s},\mathsf{a}_m)$ exits X and some successor of the action is in T.
  - Action a_m cannot exit towards $S \setminus X$. Otherwise, s would "depend on the outside" and we get a contradiction as in the case of X not containing an EC.

- No action can exit towards $X \setminus X'$, because X' is a bottom MEC in X. If an action left towards some state $t \in X \setminus X'$, then from t there would be no reachable EC in X. Hence, by Lemma 2, the states that t can reach must contain some state that only has actions exiting X, which yields a contradiction as in the case of X not containing an EC.
- a_m cannot exit towards $X' \setminus Y$, as by definition of Y all states $\mathsf{s}' \in X' \setminus Y$ have $\mathsf{G}^*(\mathsf{s}') < m$.
- The only remaining possibility is that a_m stays in Y.

Let $Z \subseteq Y$ be a bottom MEC in Y. Z exists, since by the previous step all states in Y have actions staying in Y, and by the contrapositive of Lemma 2 this implies the existence of a MEC in Y.

Note that for all states $\mathsf{s} \in Z$ we have $\mathsf{V}(\mathsf{s}) = m - c$, where $c = \max_{\mathsf{s} \in S} \Delta(\mathsf{s})$, as $Z \subseteq Y$ and $Z \subseteq X$. So in fact, Z is a *simple EC*, which will be introduced in the next subsection. This fact is relevant for the proof of Theorem 2.

We now prove that Z is a BEC by a case distinction on the bestExit of Maximizer.

- If $\mathsf{bestExit}^{\Box}_{\mathsf{G}^*}(Z) > m$,
  then the Maximizer state s that has this exit would use it to get $\mathsf{G}^*(\mathsf{s}) > m$. However, we know that $\mathsf{G}^*(\mathsf{s}) = m$, so this is a contradiction.
- If $\mathsf{bestExit}^{\Box}_{\mathsf{G}^*}(Z) = m$,
  then let $(\mathsf{s}, \mathsf{a}_\ell)$ be a state-action pair, such that $\mathsf{s} \in Z_\Box$, $(\mathsf{s}, \mathsf{a}_\ell)$ exits Z and $\mathsf{G}^*(\mathsf{s}, \mathsf{a}_\ell) = \mathsf{G}^*(\mathsf{s}) = m$. This pair exists by the assumption that there is an exit of Maximizer from Z that achieves m. However, we showed earlier, that such an action a_ℓ can only stay in Y (note that $Z \subseteq Y$, thus the earlier case distinction fully applies). Since $(\mathsf{s}, \mathsf{a}_\ell)$ exits Z, we know that there is a successor of that state-action pair in $Y \setminus Z$. This is a contradiction, because Z is a bottom MEC in Y. The argumentation is analogous to that of an action exiting Y towards $X \setminus X'$.
- If $\mathsf{bestExit}^{\Box}_{\mathsf{G}^*}(Z) < m$,
  first note that we are not done yet, because the case distinction talks about $\mathsf{bestExit}_{\mathsf{G}^*}$, while the definition of BEC uses $\mathsf{bestExit}_{\mathsf{V}}$. We still need to show that the true value of Minimizer's best exit is higher than the true value of Maximizer's one. There are two possibilities, depicted in Fig. 4 as Z_1 and Z_2.
  * If there are $\bigcirc$-exits,
    then let $(\mathsf{s}, \mathsf{a}_\ell)$ be the minimal $\bigcirc$-exit, i.e. the state-action pair with $\mathsf{s} \in Z_\bigcirc$ and $(\mathsf{s}, \mathsf{a}_\ell)$ exits Z with $\mathsf{V}(\mathsf{s}, \mathsf{a}_\ell) = \mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(Z)$. We know that $(\mathsf{s}, \mathsf{a}_\ell)$ cannot exit towards $X \setminus X'$ or $Y \setminus Z$, by the same bottom MEC argumentation as before. It cannot exit towards $X' \setminus Y$, because then $\mathsf{G}^*(\mathsf{s}, \mathsf{a}_\ell) < m = \mathsf{G}^*(\mathsf{s})$, but Minimizer uses the minimal available action. Thus, it exits towards $S \setminus X$, i.e. $(\mathsf{s}, \mathsf{a}_\ell)$ exits X. Then, by Statement (5) we know that $\mathsf{V}(\mathsf{s}, \mathsf{a}_\ell) > \mathsf{V}(\mathsf{s})$. Recall that all states in Z have the same value. Additionally, for all $\mathsf{s}' \in Z_\Box, \mathsf{a}' \in \mathsf{Av}(\mathsf{s}')$ we have $\mathsf{V}(\mathsf{s}') \geq \mathsf{V}(\mathsf{s}', \mathsf{a}')$ since Maximizer picks the best action. Let $(\mathsf{s}', \mathsf{a}')$ be the state-action pair used for $\mathsf{bestExit}^{\Box}_{\mathsf{V}}(Z)$. Then we know that Z is a BEC by the following inequation chain:
    $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(Z) = \mathsf{V}(\mathsf{s}, \mathsf{a}_\ell) > \mathsf{V}(\mathsf{s}) = \mathsf{V}(\mathsf{s}') \geq \mathsf{V}(\mathsf{s}', \mathsf{a}') = \mathsf{bestExit}^{\Box}_{\mathsf{V}}(Z)$.
  * If there are no $\bigcirc$-exits,
    then $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}} = 1$, since $\min_\emptyset = 1$. Hence, by Lemma 1, $m = 1$. Recall that in the case distinction we assumed $\mathsf{bestExit}^{\Box}_{\mathsf{G}^*}(Z) < m$. Thus Z is a BEC, since

$$\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(Z) = 1 = m > \mathsf{bestExit}^{\Box}_{\mathsf{G}^*}(Z) \geq \mathsf{bestExit}^{\Box}_{\mathsf{V}}(Z).$$

The fact that Z is a BEC is a contradiction to the assumption of the theorem that the SG contains no non-trivial BECs, as $Z \subseteq S \setminus (F \cup \mathfrak{Z})$. Thus we arrive at a contradiction in all cases, and Theorem 1 is proven. □

In Section 4.2, we show how to eliminate BECs by collapsing their "core" parts, called below MSECs (maximal simple end components). Since MSECs can only be identified with enough information about V, Section 4.3 shows how to avoid direct *a priori* collapsing and instead dynamically deflate candidates for MSECs in a conservative way.

4.2. The core of the problem: simple end components

Now we turn our attention to SG with BECs. In a BEC all Minimizer's exiting actions have a higher value than what Maximizer can achieve; hence, Minimizer will not use exiting actions, but prefers staying in the EC and steering Maximizer towards his worse exiting actions. Consequently, only Maximizer wants to take an exiting action.

Intuitively, solving a BEC amounts to computing the attractors of Maximizer's exits, i.e. those states that can reach an exit if Minimizer plays optimally. All states in such an attractor have the same value, namely that of the best exit they can reach. Thus, we define the following sub-component.

Definition 5 *(Simple EC).* An EC T is called *simple end component (SEC)*, if for all $\mathsf{s} \in T$ we have $\mathsf{V}(\mathsf{s}) = \mathsf{bestExit}^{\Box}_{\mathsf{V}}(T)$.
A SEC T is *maximal (MSEC)* if there is no SEC T' such that $T \subsetneq T'$.

To give another intuition: an EC is simple, if Minimizer cannot keep Maximizer away from his bestExit. Independently of Minimizer's decisions, Maximizer can reach the bestExit almost surely, unless Minimizer decides to leave, in which case Maximizer would achieve an even higher value.

Algorithm 2 FIND_MSEC.

```
1: function FIND_MSEC($f : S \to [0,1]$)
2:     $\mathsf{Av}' \leftarrow \mathsf{Av}$
3:     for $\mathsf{s} \in S_{\bigcirc}$ do                    # Keep only optimal actions for Minimizer
4:         $\mathsf{Av}'(\mathsf{s}) \leftarrow \{\mathsf{a} \in \mathsf{Av}(\mathsf{s}) \mid f(\mathsf{s},\mathsf{a}) = \min_{b \in \mathsf{Av}(\mathsf{s})} f(\mathsf{s}, b)\}$
5:     return $\mathsf{MEC}(\mathsf{G}_{[\mathsf{Av}/\mathsf{Av}']})$   # $\mathsf{MEC}(\mathsf{G}_{[\mathsf{Av}/\mathsf{Av}']})$ are MSECs of the original $\mathsf{G}$
```

In the MDP case, every EC is simple, as Maximizer can decide to use any exit. As a result, in MDPs all states of an EC have the *same value* and can all be collapsed into one state. In the SG case, Maximizer may be restricted by Minimizer's behaviour or even not given any chance to exit the EC at all. As a result, a BEC may contain several parts, each with different value, intuitively corresponding to different exits. Thus instead of MECs, we have to decompose into finer MSECs and only collapse these.

Example 5 *(SECs).* Assume $\alpha < \beta$ in the example of Fig. 3. Then $\{\mathsf{p},\mathsf{q}\}$ is a SEC and an MSEC. Further observe that action c is sub-optimal for Minimizer and removing it does not affect the value of any state, but simplifies the graph structure. Namely, it destructs the whole EC into several (here only one) SECs and some non-EC states (here r). △

Lemma 3 *(SECs are the core of the problem). For a set $T \subseteq S$, the following two statements hold.*

1. *If T is a BEC, then there exists a $T' \subseteq T$ such that T' is a SEC.*
2. *If T is a SEC, then either T also is a BEC or* $\mathsf{bestExit}_{\mathsf{V}}^{\bigcirc}(T) = \mathsf{bestExit}_{\mathsf{V}}^{\square}(T)$.

Proof. 1. Let τ be an optimal strategy of Minimizer. If $T \subseteq S$ is a BEC, then $\mathsf{bestExit}_{\mathsf{V}}^{\bigcirc}(T) > \mathsf{bestExit}_{\mathsf{V}}^{\square}(T)$, so the optimal strategy of Minimizer does not contain any exiting actions, or formally for all (s,a) exits T with $\mathsf{s} \in T_{\bigcirc}$, we have $\tau(\mathsf{s})(\mathsf{a}) = 0$. Hence, every state in the MDP G^{τ} induced by fixing an optimal Minimizer strategy has an available staying action, and thus it still contains an EC $T' \subseteq T$, by the contrapositive of Lemma 2.
Note that T' can be a subset of T, if Minimizer restricts the game to a part of the EC, as in Example 5.
2. Let $T \subseteq S$ be a SEC. Note that our goal is equivalent to showing

$$\mathsf{bestExit}_{\mathsf{V}}^{\bigcirc}(T) \geq \mathsf{bestExit}_{\mathsf{V}}^{\square}(T),$$

as that means either the two numbers are equal, or the best Minimizer exit is greater and thus T is a BEC.
If there is no Minimizer exit, we have $\mathsf{bestExit}_{\mathsf{V}}^{\bigcirc}(T) = 1 \geq \mathsf{bestExit}_{\mathsf{V}}^{\square}(T)$, using the convention that $\min_{\emptyset} = 1$ and the fact that no probability can be greater than 1.
If on the other hand there is a Minimizer exit, then let (s,a) be the best Minimizer exit. Then the following chain of equations proves our goal:

$$\begin{aligned}
\mathsf{bestExit}_{\mathsf{V}}^{\bigcirc}(T) &= \mathsf{V}(\mathsf{s},\mathsf{a}) && \text{(By } (\mathsf{s},\mathsf{a}) \text{ being the best Minimizer exit)}\\
&\geq \mathsf{V}(\mathsf{s}) && \text{(By definition of } \mathsf{V}(\mathsf{s}) \text{ (Equation (1)))}\\
&= \mathsf{bestExit}_{\mathsf{V}}^{\square}(T) && \text{(As } T \text{ is a SEC)}
\end{aligned}$$

□

So SECs are the core of the problem, as by Theorem 1 BECs are the reason for non-convergence, and by Lemma 3 every BEC contains a SEC. As all states in a SEC have the same value, they are easy to treat; we can even collapse them as in MDPs without changing the value.

Thus, we need to find SECs. Algorithm 2, called FIND_MSEC, shows how to compute the inclusion maximal MSECs. It returns the set of all MSECs if called with parameter V. However, as we want to compute the value and hence cannot assume to know it, later we also call this function with other parameters $f : S \to [0,1]$.

The idea of the algorithm is the following: Minimizer actions with a non-minimum value cannot be part of any SEC and thus should be ignored when identifying SECs. Hence, we construct the set of available actions Av' which does not contain non-minimum Minimizer actions. (The previous example illustrates that ignoring these actions is indeed safe as it does not change the value of the game.) We denote the game G where the available actions Av are changed to the new available actions Av' (ignoring the Minimizer's sub-optimal ones) as $\mathsf{G}_{[\mathsf{Av}/\mathsf{Av}']}$. Once removed, Minimizer has no choices to affect the outcome of the game and thus every remaining EC is simple. Another intuition for this is that we fix Minimizer's strategy to be optimal according to f and then compute the ECs in the resulting MDP.

Lemma 4 *(Correctness of Algorithm 2). $T \in$* $\mathsf{FIND\_MSEC}(\mathsf{V})$ *if and only if T is an MSEC.*

Proof. Direction $\Rightarrow$ We want to show: $T \in \mathsf{FIND\_MSEC}(\mathsf{V}) \implies T$ is an MSEC. Let $T \in \mathsf{FIND\_MSEC}(\mathsf{V})$. So T is a MEC of the game where the mapping of available actions is Av'. Hence for all $\mathsf{s} \in T_{\bigcirc}, \mathsf{a} \in \mathsf{Av}'(\mathsf{s})$ we have $\mathsf{V}(\mathsf{s},\mathsf{a}) = \min_{b \in \mathsf{Av}(\mathsf{s})} \mathsf{V}(\mathsf{s}, b) = \mathsf{V}(\mathsf{s})$. We now show that for all $\mathsf{s} \in T$ we have $\mathsf{V}(\mathsf{s}) = \mathsf{bestExit}_{\mathsf{V}}^{\Box}(T)$.

- If there is no exit for Maximizer, then $\mathsf{bestExit}_{\mathsf{V}}^{\Box}(T) = 0$, and all states in T have the value 0, since Minimizer can force the game to stay inside the EC forever.
- If there is an exiting state e for Maximizer, then also from each state in T there is a path using only states in T to e. For each state $\mathsf{s} \in T$ we can compute $\mathsf{V}(\mathsf{s})$ by recursively applying the Bellman equations. In each application, we choose an action that leads us closer to e, i.e. for t being the next state on the path to e, choose a such that $\mathsf{V}(\mathsf{s},\mathsf{a}) = \delta(\mathsf{s},\mathsf{a},t) \cdot \mathsf{V}(t) + \sum_{\mathsf{s}' \in \mathsf{Post}(\mathsf{s},\mathsf{a}) \setminus \{t\}} \delta(\mathsf{s},\mathsf{a},\mathsf{s}') \cdot \mathsf{V}(\mathsf{s}')$. Since for each state in T we have a path to e, we can replace every $\mathsf{V}(\mathsf{s}')$ in this way. By repeating this and thereby multiplying all the probabilities $\delta(\mathsf{s},\mathsf{a},t)$, the factor in front of the terms that do not contain $\mathsf{V}(e)$ approaches 0, and thus we get $\mathsf{V}(\mathsf{s},\mathsf{a}) = \mathsf{V}(e)$ for an arbitrary $\mathsf{s} \in T$ and a staying action a. Since we showed before that for Minimizer all actions have the same value, all Minimizer states have as value $\mathsf{V}(e)$. Maximizer can certainly achieve $\mathsf{V}(e)$ by picking the action with the maximal value. Maximizer cannot achieve more than $\mathsf{V}(e)$, because this is defined to be the best exit from the EC, and thus the best value that can be achieved from any state in the EC.

Since T is a MEC of the game with some available actions removed, it certainly is an EC in the original game. Thus, from this and the previous argument, we know that T is a SEC.

T is inclusion maximal, because if there was some $T' \supsetneq T$, such that T' is a SEC, then there exists some state $\mathsf{s} \in T' \setminus T$ with all staying actions of this state having the value $\mathsf{bestExit}_{\mathsf{V}}^{\Box}(T)$. If it is a Minimizer's state, it cannot have a lower exit available, because otherwise T' would not be a SEC. Thus no staying action of s is removed by Line 4, and it should also be part of the MEC in the modified game. This contradicts the assumption that $\mathsf{s} \notin T$, and thus T is an MSEC.

Direction $\Leftarrow$ Let T be an MSEC. We need to show that T is a MEC of the game where the mapping of available actions is Av'.

We first show that T is an EC in the modified game. Since T is an EC in the original game, there exists a B such that for each $\mathsf{s} \in T, \mathsf{a} \in B \cap \mathsf{Av}(\mathsf{s})$ we do not have (s,a) exits T by the first condition of Definition 2. Moreover, $B \cap \mathsf{Av}(\mathsf{s})$ is non-empty, as otherwise there could not be a path starting in s and using only actions from B, see the second condition of Definition 2. Hence for all $\mathsf{s} \in T, \mathsf{a} \in B \cap \mathsf{Av}(\mathsf{s})$ the following holds.

$$
\begin{aligned}
\mathsf{V}(\mathsf{s},\mathsf{a}) &:= \sum_{s' \in S} \delta(\mathsf{s},\mathsf{a},\mathsf{s}') \cdot \mathsf{V}(\mathsf{s}') && \text{(by definition of } V(\mathsf{s},\mathsf{a}) \text{ (Equation (2)))} \\
&= \sum_{s' \in S} \delta(\mathsf{s},\mathsf{a},\mathsf{s}') \cdot \mathsf{bestExit}_{\mathsf{V}}^{\Box}(T) && \text{(since } \mathsf{s}' \in T \text{ and } T \text{ is simple)} \\
&= \mathsf{bestExit}_{\mathsf{V}}^{\Box}(T)
\end{aligned}
$$

So every action that stays in T (i.e. does not exit T), in particular every action in B, is not removed from the game, because for all $\mathsf{s} \in T, \mathsf{a} \in \mathsf{Av}(\mathsf{s}), \neg(\mathsf{s},\mathsf{a})\,\mathsf{exits}\,T$ we have $\mathsf{V}(\mathsf{s}) = \mathsf{bestExit}_{\mathsf{V}}^{\Box}(T) = \mathsf{V}(\mathsf{s},\mathsf{a})$. The first equality comes from T being an MSEC, the second is what we have just shown.

Since no action in B is removed from the game and T is still an EC after the removal, T is inclusion maximal by the same argumentation as at the end of the $\Rightarrow$ case, and thus $T \in \mathsf{FIND\_MSEC}(\mathsf{V})$. $\Box$

Remark 1 *(Hypothetical algorithm with an oracle).* In Section 3, we have seen that collapsing MECs does not ensure BVI convergence on SGs. Collapsing does not preserve the values, since in BECs states with different values would be collapsed. Hence we only want to collapse MSECs, where the values are the same. If we collapse all MSECs of a game G, the resulting game G′ does not contain any BECs (as by Lemma 3 every BEC contains a SEC), and hence by Theorem 1 BVI converges. Using a similar argument as the correctness of collapsing in MDPs [11], we could show that the values in G′ are equivalent to those in G.

However, the difficulty with this algorithm is that it requires an oracle to compare values, for instance a sufficiently precise approximation of V. Consequently, we cannot pre-compute the MSECs, but have to find them while running BVI.

But since the approximations converge only in the limit we may never be able to conclude on simplicity of some ECs. For instance, if $\alpha = \beta$ in Fig. 3, and if the approximations converge at different speeds, then Algorithm 2 always outputs only a part of the EC, although the whole EC on $\{\mathsf{p},\mathsf{q},\mathsf{r}\}$ is simple.

4.3. Deflating simple end components

Since MSECs cannot be identified from approximations of V for sure, we refrain from collapsing[10] and instead only decrease the over-approximation in a conservative way. We call the method *deflating*, by which we mean decreasing the

[10] Our subsequent method can be combined with local collapsing whenever the lower and upper bounds on V are conclusive.

Algorithm 3 DEFLATE.

```
1: function DEFLATE(State set $T$, $f: S \to [0,1]$)
2:     for $s \in T$ do
3:         $f(s) \leftarrow \min(f(s), \mathsf{bestExit}^{\Box}_{f}(T))$ # Decrease the upper bound
4:     return $f$
```

Algorithm 4 UPDATE procedure for bounded value iteration on SG.

```
1: procedure UPDATE($\mathsf{L}: S \to [0,1]$, $\mathsf{U}: S \to [0,1]$)
2:     $\mathsf{L}, \mathsf{U}$ get updated according to Eq. (3) and (4) # Bellman updates
3:     for $T \in$ FIND_MSEC($\mathsf{L}$) do          # Use lower bound to find ECs
4:         $\mathsf{U} \leftarrow$ DEFLATE($T$, $\mathsf{U}$)          # and deflate the upper bound there
```

upper bound of all states in an EC to its $\mathsf{bestExit}^{\Box}_{\mathsf{U}}$, see Algorithm 3. The procedure DEFLATE (called on the current upper bound U_i) decreases this upper bound to the minimum possible value according to the current approximation and thus prevents states from only depending on each other, as in SECs.

Overall, we want to gradually approximate SECs and perform the corresponding adjustments, but we do not commit to any of the approximations by collapsing and thus definitively changing the game graph.

Lemma 5 *(DEFLATE is sound). For any $f: S \to [0,1]$ such that $f \geq \mathsf{V}$ (where $\geq$ is point-wise comparison) and any state-set $T \subseteq S$, it holds that* $\mathsf{DEFLATE}(T, f) \geq \mathsf{V}$.

Proof. Let $T \subseteq S$ and $f: S \to [0,1]$ be such that $f \geq \mathsf{V}$. We reformulate the goal to saying that for all states $s \in T$ it holds that $\min(f(s), \mathsf{bestExit}^{\Box}_{f}(T)) \geq \mathsf{V}(s)$.

This is equivalent to our goal, because the change in Line 3 is the only change Algorithm 3 applies and because the comparison of the functions $\mathsf{DEFLATE}(T, f)$ and V is point-wise.

If in Line 3 of DEFLATE the expression $\min(f(s), \mathsf{bestExit}^{\Box}_{f}(T))$ gets evaluated to $f(s)$ or if $f(s) = \mathsf{bestExit}^{\Box}_{f}(T)$, by assumption of $f \geq \mathsf{V}$ the goal trivially holds.

If $f(s) > \mathsf{bestExit}^{\Box}_{f}(T)$, the following chain of equations proves our goal:

$$\begin{aligned} \mathsf{DEFLATE}(T, f)(s) &= \mathsf{bestExit}^{\Box}_{f}(T) \\ &\geq \mathsf{bestExit}^{\Box}_{\mathsf{V}}(T) && \text{(Since } f \geq \mathsf{V}\text{)} \\ &\geq \mathsf{V}(s) && \text{(Since no state can achieve a greater value than the best exit)} \end{aligned}$$

□

Using DEFLATE, we can define an improved UPDATE procedure which makes the BVI algorithm (Algorithm 1) converge. The difference to the standard update are the additional lines 3 and 4. After the usual Bellman updates in Line 2, we compute the MSEC candidates according to the current under-approximation in Line 3. Then for every MSEC candidate, the upper bound is deflated in Line 4. Since DEFLATE is sound, the upper bound always remains a correct over-approximation. Intuitively, the over-approximation converges, because L converges to the value in the limit, and thus we eventually find the correct MSECs and deflate them.

The main loop of Algorithm 1 using Algorithm 4 as UPDATE describes an operator $\mathcal{B}: ([0,1]^S, [0,1]^S) \to ([0,1]^S, [0,1]^S)$, i.e. given two functions from states to $[0,1]$, it computes two updated versions of these functions. In order to prove the correctness and termination of Algorithm 1 using Algorithm 4 as UPDATE, we first prove that the sequence of upper bounds converges to a fixpoint.

Lemma 6 *(Upper bound converges to a fixpoint). The limit of repeatedly applying the operator $\mathcal{B}$ to the initial lower and upper bound exists and is a fixpoint of the operator $\mathcal{B}$, i.e.* $\lim_{i\to\infty} \mathcal{B}^i(\mathsf{L}_0, \mathsf{U}_0) = \mathcal{B}\left(\lim_{i\to\infty} \mathcal{B}^i(\mathsf{L}_0, \mathsf{U}_0)\right)$.

Proof. On a high level, the proof amounts to phrasing the problem so that the Kleene fixpoint theorem is applicable. Concretely, we follow the proof of [27, 8.15 CPO Fixpoint Theorem I.], adjusting some details to fit our setting.

We consider the domain $[0,1]^{|S|} \times [0,1]^{|S|}$, i.e. every element consists of two functions from states to real numbers, the under- and over-approximation. We say $(\mathsf{L}_1, \mathsf{U}_1) \preceq (\mathsf{L}_2, \mathsf{U}_2)$ if and only if both $\mathsf{L}_1 \leq \mathsf{L}_2$ and $\mathsf{U}_1 \geq \mathsf{U}_2$ with component-wise comparison. Intuitively, an element on the right side of $\preceq$ is a more precise approximation. The bottom element of the domain is $(\vec{0}, \vec{1})$, where $\vec{a}$ denotes the function that assigns a to all states. We further restrict the domain to exclude elements of the domain that are trivially irrelevant for the computation. Concretely, we exclude all tuples (L, U) where $\mathsf{L}(s) < 1$ for a target state $s \in F$ or $\mathsf{U}(s) > 0$ for a state with no path to the target $s \in \mathcal{Z}$. Then the bottom element is $\bot = (\mathsf{L}_0, \mathsf{U}_0)$, where $\mathsf{L}_0(s)$ is 1 for target states and 0 everywhere else, and $\mathsf{U}_0(s)$ is 0 for states in $\mathcal{Z}$ and 1 everywhere else. Note that these are the vectors that we have before the first iteration of the main loop of Algorithm 4. The comparator $\preceq$ induces a complete partial order over the domain, as we have a bottom element, and as every directed subset has a supremum; the latter claim holds, because $\preceq$ reduces to component-wise comparisons between real numbers from $[0,1]$, where suprema exist. For more details on the definition of directed set and complete partial orders, we refer to [27, Definition 7.7] respectively [27, Definition 8.1].

We prove several facts about $\mathcal{B}$ that we need for the final argument. $\mathcal{B}$ first applies Bellman updates and then additionally deflates the over-approximation. Bellman updates are monotonic (see Section 2.3) and deflating is monotonic, as in Line 3 of Algorithm 3 we take the minimum of the current value and the best exit. Thus for all i and every pair of approximations (L,U) we have $\mathcal{B}^i(\mathsf{L},\mathsf{U}) \preceq \mathcal{B}^{i+1}(\mathsf{L},\mathsf{U})$. Intuitively, applying $\mathcal{B}$ can only make the approximations more precise. It follows that

$$\bot \preceq \mathcal{B}(\bot) \preceq \cdots \preceq \mathcal{B}^i(\bot) \preceq \mathcal{B}^{i+1}(\bot) \preceq \ldots$$

form an ascending chain. Since the domain with $\preceq$ is a complete partial order, we know that

$$\lim_{i\to\infty} \mathcal{B}^i(\bot) = \sup_{i\geq 0} \mathcal{B}^i(\bot), \tag{7}$$

and that the supremum exists.

We now argue that $\mathcal{B}$ eventually is continuous. Bellman updates are continuous (see Section 2.3). Applying Bellman updates to the lower bound converges to the true value V, see e.g. [18]. Thus, after a finite number of steps n the optimal actions of Minimizer according to L do not change any more, because the lower approximation has converged enough such that L_n of every suboptimal action is larger than L_n of every optimal action. In case of multiple optimal actions, one of them might converge at slower speed, and only some optimal action is selected, not all of them. Still, the selected actions stay constant for every $n' \geq n$. This implies that we always find and deflate the same ECs. Thus deflating an EC T simplifies to applying a Bellman update on the whole T, updating every over-approximation to the best exit. Thus, for every $n' \geq n$, deflating is also continuous. Since we argue about behaviour in the limit, it suffices that $\mathcal{B}$ is eventually continuous.

Then we can conclude:

$$\begin{aligned}
\mathcal{B}(\lim_{i\to\infty} \mathcal{B}^i(\mathsf{L}_0,\mathsf{U}_0)) &= \mathcal{B}(\sup_{i\geq 0}\mathcal{B}^i(\bot)) && \text{(Definition of } \bot \text{ and Equation (10))}\\
&= \sup_{i\geq 0}\mathcal{B}(\mathcal{B}^i(\bot)) && \text{(The supremum is certainly larger than } \mathcal{B}^n\text{, thus } \mathcal{B} \text{ is continuous.)}\\
&= \sup_{i\geq 1}\mathcal{B}^i(\bot) && \\
&= \sup_{i\geq 0}\mathcal{B}^i(\bot) && \text{(since } \bot \preceq \mathcal{B}^n(\bot) \text{ for all } n\text{)}\\
&= \lim_{i\to\infty} \mathcal{B}^i(\mathsf{L}_0,\mathsf{U}_0) && \text{(Definition of } \bot \text{ and Equation (10))}
\end{aligned}$$

□

Remark 2. Lemma 6 was assumed without justification in the conference version of the paper, as it relies on standard methods from lattice theory and seemed obvious. However, there is one surprising complication: the operator $\mathcal{B}$ is not continuous in general. This is because the algorithm deflates SEC-candidates, and these candidates vary depending on the current under-approximation. A state might be part of a SEC-candidate for a less precise under-approximation, but not be part of a SEC-candidate for a more precise under-approximation. In the first case it is deflated and its upper bound decreased. In the second case, it is not deflated and its upper bound stays. This violates continuity, since the upper bound in the first case is more precise, even though the element of the domain was more precise in the second case. This is why we argue about eventual continuity in the proof of Lemma 6.

Theorem 2 *(Soundness and completeness). Algorithm 1, using Algorithm 4 as* UPDATE*, produces monotonic sequences* L *under- and* U *over-approximating* V*, and terminates for every* $\varepsilon > 0$*.*

Proof. We denote by L_i and U_i the lower/upper bound function after the i-th call of UPDATE. L_i and U_i are monotonic under- respectively over-approximations of V because they are updated via Bellman updates, which preserve monotonicity and the under-/over-approximating property (this can be shown by a simple induction), and from Lemma 5.

Note that UPDATE, FIND_MSEC and DEFLATE take finite time, as all the for-loops in the algorithms iterate over finite sets. So it remains to prove that the main loop of Algorithm 1 terminates, i.e. that for all $\varepsilon > 0$ there exists an n such that $\mathsf{U}_n(\mathsf{s}_0) - \mathsf{L}_n(\mathsf{s}_0) < \varepsilon$. It suffices to show that $\lim_{n\to\infty}\mathsf{U}_n - \mathsf{V} = 0$, because $\lim_{n\to\infty}\mathsf{L}_n = \mathsf{V}$ (from e.g. [18]).

In the following, let $\mathsf{U}^* := \lim_{n\to\infty}\mathsf{U}_n$ and $\Delta(\mathsf{s}) := \mathsf{U}^*(\mathsf{s}) - \mathsf{V}(\mathsf{s})$; we also use $\Delta(\mathsf{s},\mathsf{a})$, see Footnote 6. Assume for contradiction that the algorithm does not converge, i.e. there exists a state with $\Delta(\mathsf{s}) > 0$.

The rest of the proof is structured as follows.

Step 1 From $\Delta(\mathsf{s}) > 0$ we derive that there has to be a SEC Z by using an analogue of the proof of Theorem 1.
Step 2 Then we show that by applying one more iteration of the loop of Algorithm 1, a SEC $Z' \subseteq Z$ is found and deflated, thereby decreasing the upper bound.
Step 3 This is a contradiction, because by Lemma 6, U^* is a fixpoint. Thus we get that for all $\mathsf{s} \in S$ we have $\Delta(\mathsf{s}) = 0$ and hence convergence from above.

The full technical proof follows.

Step 1 We reuse parts of the proof of Theorem 1 to prove that there exists a SEC Z. Note that as before when proving Theorem 1, we have the assumption that there exists a state with $\Delta(\mathsf{s}) > 0$.
The difference is, that instead of the greatest fixpoint G^* we use limit of the upper bound computed by our algorithm U^*. We prove the analogue of Statement (5).

$$\forall(\mathsf{s},\mathsf{a})\,\text{exits}\,X: \begin{cases} \mathsf{U}^*(\mathsf{s},\mathsf{a}) < \mathsf{U}^*(\mathsf{s}) & \text{if}\,\mathsf{s} \in X_\Box \\ \mathsf{V}(\mathsf{s},\mathsf{a}) > \mathsf{V}(\mathsf{s}) & \text{if}\,\mathsf{s} \in X_\bigcirc \end{cases}$$

The only argument that we need to adjust is that for the fact: $\forall \mathsf{s} \in S_\Box, \mathsf{a} \in \mathsf{Av}(\mathsf{s}) : \mathsf{U}^*(\mathsf{s}) \geq \mathsf{U}^*(\mathsf{s},\mathsf{a})$. For a Bellman update this property holds, as the upper bound of a state is always the maximum upper bound of all its actions.
The additional deflating does not violate the property. Deflating is only executed if $\mathsf{s} \in Z$ for some EC Z that was found by the latest call of FIND_MSEC. It sets $\mathsf{U}(\mathsf{s}) = \mathsf{bestExit}^\Box_\mathsf{U}(Z)$. For actions staying inside Z, all successors have the same upper bound, namely $\mathsf{bestExit}^\Box_\mathsf{U}(Z)$. Thus all staying actions a satisfy $\mathsf{bestExit}^\Box_\mathsf{U}(Z) = \mathsf{U}(\mathsf{s}) = \mathsf{U}(\mathsf{s},\mathsf{a})$. No exiting action a can have $\mathsf{U}(\mathsf{s},\mathsf{a}) > \mathsf{bestExit}^\Box_\mathsf{U}(Z)$, because then by definition of bestExit we would have $\mathsf{U}(\mathsf{s},\mathsf{a}) = \mathsf{bestExit}^\Box_\mathsf{U}(Z)$. So all exiting actions satisfy $\mathsf{U}(\mathsf{s}) = \mathsf{bestExit}^\Box_\mathsf{U}(Z) \geq \mathsf{U}(\mathsf{s},\mathsf{a})$. As every application of deflating preserves the property, it also transfers to U^*.
Then, using the analogue of Statement (5) and Lemma 2, we can construct the sets X, X', Y and Z as before. All states in Z have the same value, as noted already in the proof of Theorem 1. This implies that Z is a SEC, as the only possible value for the states is $\mathsf{bestExit}^\Box_\mathsf{V}(Z)$. If there is no Maximizer exit, the value of all states as well as $\mathsf{bestExit}^\Box_\mathsf{V}(Z)$ is 0. If there is a Maximizer exit, let (s,a) be the best exit. Then $\mathsf{V}(\mathsf{s}) \geq \mathsf{V}(\mathsf{s},\mathsf{a}) = \mathsf{bestExit}^\Box_\mathsf{V}(Z)$, and an EC cannot have a higher value than that of its best Maximizer exit.
Step 2 Applying one more iteration of the loop of Algorithm 1, i.e. calling UPDATE once more, finds an EC $Z' \subseteq Z$. This is the case, because after some finite number of steps L_i is close enough to V such that the following holds: $\forall \mathsf{s} \in Z_\bigcirc, (\mathsf{s},\mathsf{a})$ exits $Z : \mathsf{L}_i(\mathsf{s},\mathsf{a}) > \mathsf{L}_i(\mathsf{s})$. This will happen, because all $\bigcirc$-exits from Z lead towards $S \setminus X$, and hence by Statement (5), we have $\mathsf{V}(\mathsf{s},\mathsf{a}) > \mathsf{V}(\mathsf{s})$. So from some point onward, for all $i' \geq i$, we have $\mathsf{L}_{i'}(\mathsf{s},\mathsf{a}) > \mathsf{V}(\mathsf{s}) \geq \mathsf{L}_{i'}(\mathsf{s})$. Then, when computing FIND_MSEC($\mathsf{L}_{i'}$), all actions exiting Z are removed and the MECs of the modified game are computed. This will result in finding an EC $Z' \subseteq Z$, because every state has to have at least one action, and all actions have to stay inside Z.
Z' can be a strict subset of Z, for example if the BEC looks as depicted in Fig. 3 and $\alpha = \beta$, but $\mathsf{L}_{i'}(q,e) \neq \mathsf{L}_{i'}(r,f)$ for all i', because they converge at different speeds. However, this poses no problem for the convergence of our algorithm; intuitively, after Z' is deflated, the states from $Z \setminus Z'$ adjust their upper bounds according to their best exit, as going to Z' is suboptimal.
When calling UPDATE once more on U^*, we can instantiate L with some L_i that is such that we find $Z' \subseteq Z$ (by previous arguments). Since Z' is returned by FIND_MSEC(L_i), the algorithm executes DEFLATE(Z', U^*). Then for all $\mathsf{s} \in Z'$ we set $\mathsf{U}^*_{\text{new}}(\mathsf{s}) = \mathsf{bestExit}^\Box_{\mathsf{U}^*}(Z')$, thereby decreasing the upper bound and yielding the contradiction $\mathsf{bestExit}^\Box_{\mathsf{U}^*}(Z') < \mathsf{U}^*(\mathsf{s})$, because of the following argument.
There have to be $\Box$-exits. If there were no $\Box$-exits in Z', then for all i we have $\mathsf{bestExit}^\Box_{\mathsf{U}_i}(Z') = 0$ (the case of $\max_\emptyset$), and hence $\Delta(\mathsf{s})$ would be 0; however, since $Z' \subseteq X$, it has to hold that $\Delta(\mathsf{s}) > 0$. Let $(\mathsf{s}_e, \mathsf{a}_e)$ be one of the best $\Box$-exits, i.e. $\mathsf{U}^*(\mathsf{s}_e,\mathsf{a}_e) = \mathsf{bestExit}^\Box_{\mathsf{U}^*}(Z')$. Using Statement (5), we know that $\mathsf{U}^*(\mathsf{s}_e,\mathsf{a}_e) < \mathsf{U}^*(\mathsf{s}_e)$.
Step 3 Finally, we get $\mathsf{U}^*_{\text{new}}(\mathsf{s}_e) = \mathsf{bestExit}^\Box_{\mathsf{U}^*}(Z') = \mathsf{U}^*(\mathsf{s}_e,\mathsf{a}_e) < \mathsf{U}^*(\mathsf{s}_e)$. This is a contradiction, because by Lemma 6 U^* is a fixpoint, but applying one more update yields a smaller $\mathsf{U}^*_{\text{new}}$.
Hence our assumption that there was some state with a positive difference is wrong, and thus BVI also converges from above. Thus, the algorithm will terminate for every $\varepsilon > 0$. $\Box$

The story so far:

1. Using only Bellman updates on an over-approximation need not converge to the value (see Corollary 1).
2. Since maximal ECs can have non-constant values, instead we have to focus on maximal *simple* ECs. Collapsing those allows BVI to converge (see Remark 1).
3. As we cannot find maximal simple ECs without knowing a close enough approximation of the value, we identify candidates for simple ECs on the fly and gradually *deflate* them (see Algorithm 4).
4. This is correct, because deflating is sound, and it converges, because we eventually find the correct maximal simple ECs and deflate them (see Theorem 2).

Algorithm 5 Update procedure for the learning/BRTDP version of BVI on SG.

```
1:  procedure UPDATE($\mathsf{L}\colon S \to [0,1]$, $\mathsf{U}\colon S \to [0,1]$)
      # Simulating
2:      $\rho \leftarrow \mathsf{s}_0$
3:      repeat
4:          $\mathsf{s}' \leftarrow$ sampled as successor of the last state in $\rho$
5:          $\rho \leftarrow \rho\ \mathsf{s}'$
6:      until $\mathsf{s}' \in F$ or SIMULATION_STUCK
      # Deflating
7:      if SIMULATION_STUCK then
8:          for $T \in$ FIND_MSEC$_\rho$(L) do
9:              DEFLATE($T$, U)
      # Bellman updates
10:     $\mathsf{L}, \mathsf{U}$ get updated by Eq. (3) and (4) on all states $\mathsf{s} \in \rho$
```

5. Learning-based algorithm

State spaces of systems in model checking often are very large, so that standard algorithms take too long; the state space might be so large that it doesn't even fit onto memory. To tackle this problem, we harvest our convergence results and BVI algorithm for SGs to extend the asynchronous learning-based approach of BRTDP (bounded real time dynamic programming) to SGs.

Asynchronous value iteration selects in each round a subset $T \subseteq S$ of states and performs the Bellman update in that round only on T. Consequently, it may speed up computation if "important" states are selected, i.e. states that have a high probability to be visited and where the current error still is large. However, using standard VI one does not know the current error and thus cannot estimate the importance of the states; moreover, if certain states are not selected infinitely often the lower bound may not even converge.

In the setting of bounded value iteration, the current error bound is known for each state and thus convergence can easily be enforced. This gave rise to algorithms such as BRTDP in the setting of stopping MDPs [51], where the states are selected as those that appear on a simulation run. Very similar is the adaptation for general MDP [11, Section 4.1].

These algorithms in principle still have access to every parameter of the model, e.g. the exact transition function for every state-action pair. The advantage is that they are based on *partial exploration*, i.e. they try to avoid working on the whole state space by only considering those states that were encountered in simulations. This allows to work on the "core" of the model, which typically is much smaller than the whole state space [41]. Still, in the worst case the whole state space might be explored. Note that there are variants of the algorithms which do not know the transition function, but only the minimum transition probability for MDPs [11, Section 4.2] and for SGs [6].

Algorithm 5 shows the pseudocode for the BRTDP version of UPDATE. After describing the algorithm, we illustrate it as well as its advantages on several examples. Then we explain how some parts of the algorithm can be improved. Afterwards, we talk about the differences between Algorithm 5 and its predecessor from [11], as well as advantages and disadvantages when compared to the deterministic Algorithm 4 and finally prove its correctness.

Description of the algorithm. First a run of the SG is simulated (Lines 2-6). It starts with the initial state and then repeatedly samples a successor of the last state in the current run. When sampling, the non-deterministic choice is resolved by picking the "most promising action". For a Minimizer state, this is the action with the lowest lower bound, for a Maximizer state dually it is the highest upper bound; in case of ties, the choice is resolved uniformly at random. The probabilistic choice can be resolved by sampling a successor according to the transition distribution; a different idea is described below. The simulation is stopped either when a target state is reached or when it is stuck (SIMULATION_STUCK) inside an EC that it cannot leave, see Example 6. Several ways to detect if a simulation is stuck are discussed after the examples. If the simulation was stuck in an EC, we need to deflate this EC to get new information (Line 7-9), see Example 7. Note that in the pseudocode, we wrote FIND_MSEC$_\rho$(L) to indicate that we do not search for MSECs in the whole game, but only on the path we just constructed. The procedure uses the knowledge of the transition function to ensure that a cycle that occurred along the path is indeed an EC. Afterwards Bellman updates are executed on all states that were just simulated (Line 10). These updates happen asynchronously and backwards, which allows them to propagate information faster, see Example 8.

Example 6 shows why we need to check whether a simulation is stuck.

Example 6 *(Simulation).* Consider the SG in Fig. 2. The starting state s only has a single available action with a single successor. The next state t can choose between action b and c. Both currently have an upper bound of 1, so the action is picked randomly. Assume action c was selected; then the next state is sampled according to the distribution $\delta(\mathsf{t},\mathsf{c})$. Let o be the next state in the simulation. If we just had the breaking condition $\mathsf{s}' \in F$, we would continue sampling infinitely. The additional condition to detect that the simulation is stuck in the EC $\{\mathsf{o}\}$ allows us to stop. Several ways to detect whether a simulation is stuck are discussed after the examples. △

Example 7 demonstrates when and how DEFLATE is executed.

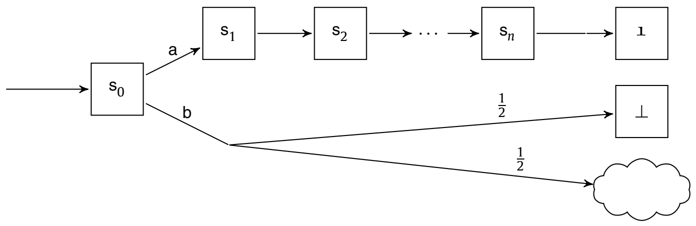


Fig. 5. An example of an SG to illustrate the advantages of BRTDP. The cloud symbolizes an arbitrarily large state space that need not be explored.

Example 7 *(Deflating).* Consider the SG in Fig. 2. When a simulation is stuck in the EC $\{\mathsf{o}\}$, DEFLATE is executed on this EC. Since there is no exit, the upper bound is correctly set to 0.

If afterwards the simulation is stuck in the EC $\{\mathsf{s}, \mathsf{t}\}$, because $\mathsf{U}(\mathsf{t}, \mathsf{b}) = 1 > \frac{2}{3} = \mathsf{U}(\mathsf{t}, \mathsf{c})$, then DEFLATE is executed on $\{\mathsf{s}, \mathsf{t}\}$, setting the upper bound of both states to the best current approximation $\mathsf{bestExit}_{\mathsf{U}}^{\Box}(\{\mathsf{s}, \mathsf{t}\}) = \mathsf{U}(\mathsf{t}, \mathsf{c}) = \frac{2}{3}$.

In the next iteration, the simulation is stuck again as $\mathsf{U}(\mathsf{t}, \mathsf{b}) = \frac{2}{3} > \frac{5}{9} = \mathsf{U}(\mathsf{t}, \mathsf{c})$, and thus DEFLATE is executed again. This is repeated until the required precision is reached. △

Example 8 shows that information can be propagated faster using asynchronous VI.

Example 8 *(Propagation of information).* Consider the top path in the SG in Fig. 5. To propagate the information that the target can be reached surely from s_n to s_0, normal VI needs $n + 1$ steps. After simulating the top path, BRTDP executes the Bellman updates in a backward manner, directly using the updated information: first $\mathsf{L}(\mathsf{s}_n)$ is set to 1; then $\mathsf{L}(\mathsf{s}_{n-1})$ is updated and can immediately use the information that $\mathsf{L}(\mathsf{s}_n) = 1$, and hence also is set to 1. Thus after one simulation of the top path and one update, BRTDP knows the result for this game, while normal VI needs n steps. This idea of immediately using the information available is part of asynchronous value iteration, and not directly linked to simulating paths; however, simulating the paths gives an easy and reasonable way to decide which states to update first. △

Example 9 shows how BRTDP can avoid exploring the whole state space.

Example 9 *(Avoid state space exploration).* Consider the bottom path in the SG in Fig. 5. The cloud symbolizes an arbitrarily large state space with arbitrary value. After sampling $\perp$ as successor once, $\mathsf{U}(\mathsf{s}_0, \mathsf{b}) \leq 0.5$. Then b is never used in simulations from then on, as the upper bound of the other action is higher, and hence the other action is "more promising". So the cloud will most likely not be explored often, potentially (if $\perp$ is picked as successor the first time) never. This way, BRTDP can ignore billions of states, if they are not relevant for convergence. The value of s_0 certainly depends on action a, so the exact value of action b does not need to be computed. In contrast, normal VI and BVI have to remember the bounds for all the states symbolized by the cloud. △

Detecting SIMULATION_STUCK. We now discuss ways to detect that a simulation is stuck. A safe approach is to keep a partial model of all states of the current simulation and their successors, and check this partial model for ECs. This is similar to the idea of candidates in [26], but in contrast to that work it can utilize the full information about the model. However, as it is acceptable to have false positives (report stuck if we are not stuck), we can use heuristics such as stopping after the simulation has a certain length, which is a lot faster to check and easier than keeping track of a partial model. Several heuristics about when to stop are discussed and compared in [11,58]; in our own experiments, however, none of their suggestions performed as well as the simple idea of stopping the simulation when the length of the run becomes longer than twice the size of the currently explored state space. Even if deflation is not necessary in an EC and the simulation has a positive probability to exit, it might still loop several times. Hence the bound to stop the simulation should be greater than the explored state space, to allow for some looping. However, the approaches of [11,58] allowed to cycle for a very long time, thus sometimes wasting time before generating new information. For a more in-depth discussion of stopping heuristics, we refer the interested reader to [5].
Improvements for probabilistic choice. Another part of the algorithm that can be heuristically improved is how to resolve the probabilistic choice. Instead of picking the successor according to the transition distribution, we can weight the probabilities with the difference $\mathsf{U} - \mathsf{L}$. It was shown in [11,58] that this converges faster than just using the probabilities. This is the case, because states where we still know little (high difference between U and L) get more weight than states where we know a lot. In particular, states that have already converged ($\mathsf{U} - \mathsf{L} = 0$) are not assigned any weight. Note that it is important to not only consider the difference between the bounds, but also the transition distribution, as states that are reached with a high probability are more important than states that are only reached with a very small probability.
Comparison. Algorithm 5 is built upon the algorithm in [11]. The key differences are that (i) in SG there also is a minimizing player, who minimizes the lower bound instead of maximizing the upper bound and that (ii) in SG the approach of collapsing does not work in general, and hence we use deflating.

In contrast to using Algorithm 4 as UPDATE, BRTDP uses simulation and asynchronous partial updates and deflations. The (dis-)advantages of BRTDP on SG compared to BVI are the same as for the special case of MDP [11]: the backward updates propagate information faster and the exploration of parts of the state space can be avoided, as illustrated in Examples 8 and 9. This can save both time and memory. However, if large parts of the state space are relevant or if there are many ECs, simulations tend to perform worse than straightforward updates on the whole state space. Still, for models that are too large to load into memory, BRTDP is an option to get at least some information on the value.

Theorem 3 *(BRTDP soundness and completeness). Algorithm 1, using Algorithm 5 as* UPDATE*, produces monotonic sequences* L *under- and* U *over-approximating* V*, and terminates almost surely for every* $\varepsilon > 0$.

Proof. Soundness (L and U being monotonic sequences under- respectively over-approximating the value) follows from the same argument as in the proof for the deterministic algorithm. The fact that we only update parts of the state space does not affect the correctness of the Bellman update or deflation.

For completeness, the proof is similar to the one for MDPs [11, Theorem 3]. The difference is (i) that we also have to argue that Minimizer picks correct actions in the simulations by using the one with minimal lower bound and (ii) that instead of collapsing ECs we deflate them.

As before, we denote by L_i and U_i the under-/over-approximations after the i-th iteration of the main loop and show that $\mathsf{U}_\infty(\mathsf{s}_0) - \mathsf{L}_\infty(\mathsf{s}_0) = 0$, which implies that for every $\varepsilon > 0$ the algorithm terminates. Note that in contrast to the previous theorems, we only have to show that the approximations converge in the initial state, not in all states. We instead prove the claim that the approximations converge in all states that are sampled infinitely often by the simulations.

Let S_∞ be exactly those states that, when running the algorithm forever, are sampled infinitely often almost surely. Since every simulation starts in s_0, we have $\mathsf{s}_0 \in S_\infty$.

We call actions that are optimal according to U respectively L approx-optimal. Concretely, in iteration i an action in a Maximizer state is approx-optimal if it has the highest upper bound U_i; and dually in a Minimizer state if it has the lowest lower bound L_i. Every approx-optimal action has a positive probability to be chosen, and every successor of such an action has a positive probability to be sampled. Thus, when considering a state $\mathsf{s} \in S_\infty$ and an action a that is approx-optimal infinitely often, this action is picked infinitely often almost surely; further, all successors are sampled infinitely often almost surely, thus we have $\mathsf{Post}(\mathsf{s}, \mathsf{a}) \subseteq S_\infty$. Note that actions that are not picked infinitely often are detected as not being approx-optimal in some iteration, cf. Example 9.

We know the approximations for all states in S_∞ are updated infinitely often almost surely. Thus we can use the same argument as in the proofs of Theorems 1 and 2: assume there was a state $\mathsf{s} \in S_\infty$ with a positive difference $\mathsf{U}_\infty(\mathsf{s}) - \mathsf{L}_\infty(\mathsf{s}) > 0$. Then we can consider the set of states $X \subseteq S_\infty$ with maximal difference, where again we know that no state may "depend on the outside": since the sampling picks the optimal actions (according to U_∞ respectively L_∞ for Maximizer/Minimizer states) and every state in S_∞ is updated infinitely often, we know that no optimal action can lead outside of X. Thus, by the same case distinction as in Theorem 1, we have that X has to contain a bottom MEC Z. However, since $X \subseteq S_\infty$, Z is almost surely sampled, detected and deflated. If Z has no exit, its upper bound becomes 0, decreasing the difference to 0; if Z has an exit, it has to exit towards a state outside of X, thereby again "depending on the outside" and decreasing the upper bound. Thus we get the contradiction to the assumption that there was a state $\mathsf{s} \in S_\infty$ with a positive difference and can conclude that the approximations converge for all states that are sampled infinitely often, in particular the initial state. □

6. Experimental results

In the following, we present our experimental results on how these theoretical improvements perform practically on MDPs and on SGs. After describing the used models, tools and the technical setup, we report on some optimizations which we considered. Next, we present our results on MDPs and finally on SGs. Note that we include tests on MDPs since there are not so many SG models available for benchmarking, and since it is interesting to see how the more general approach of deflating compares to the more specialized collapsing.

6.1. Models and tools

We implemented both our algorithms as an extension of PRISM-games [43], a branch of PRISM [44] that allows for modelling SGs, utilizing previous work of [11,58] for MDP and SG with single-player ECs, respectively.

We tested the implementation on the SGs from the PRISM-games case studies[11] that have reachability properties and one additional model from [13] (thus using all models with reachability properties that were used in [58]). We compared the results with both the explicit and the hybrid engine of PRISM-games, but since the models are small both of them performed similar and we only display the results of the faster hybrid engine in Table 4.

Furthermore we ran experiments on MDPs from the PRISM benchmark suite [45] and on the adversarial example from [33]. We compared our results there to the hybrid and explicit engine of PRISM 4.5, the interval iteration of [33]

[11] prismmodelchecker.org/games/casestudies.php.

Table 1
Verification times (in seconds) for both our algorithms, BVI and BRTDP, applied to the model cloud with N=6, using various values of i for Optimization (III). An X in the table indicates a timeout.

	1	10	100	1000
BVI	X	123	123	117
BRTDP	11	12	25	255

as implemented in PRISM, the BRTDP implementation of [11] and the hybrid engine of STORM 1.3.0 [28], which uses topological value iteration. All the models, MDPs as well as SGs, are described in Appendix C.
Technical setup. All of the experiments were conducted on a server with 256 GB RAM and 2 Intel(R) Xeon(R) E5-2630 v4 2.20 GHz processors. We limited the computation to one core to avoid results being incomparable due to different times spent parallelizing. All model checkers worked at a precision of $\varepsilon = 10^{-6}$. Each experiment had a timeout of 15 minutes, which is represented by an X in the tables. We set the available Java memory to 16 GB. Still, the largest versions of csma and mer could not be loaded with the explicit engine of PRISM. Since the learning-based approaches are randomized, we took the average of 20 repetitions of the experiments.

6.2. Optimizations

We considered the following optimizations:

(I) collapsing of SECs when we are certain that it is safe;
(II) preferring exiting actions during the simulation in BRTDP;
(III) executing FIND_MSEC and DEFLATE only every i steps for various i;
(IV) caching the MECs of the SG and restricting the SEC computation to them.

When used with the learning-based algorithm, optimizations (I) and (II) reduce the length of simulations. As an example, consider the SG in Fig. 1: after DEFLATE was executed on the EC $\{\mathsf{p}, \mathsf{q}\}$, both action b and c have the same upper bound. So in state q, a simulation randomizes uniformly between picking b and c, which means that there is a positive probability to cycle through the EC again. Since we are only interested in runs that leave the EC, we want to minimize the time that is spent cycling through an EC again.

Optimization (I) does this by collapsing ECs where we are certain that collapsing is safe, i.e. that the EC is a SEC. Collapsing a SEC intuitively means replacing it with a single state that has only the exiting actions available, thereby removing the possibility to loop inside the EC. Thus we always use the best leaving action immediately. Extending the collapsing from MDPs to SGs requires some technical changes, which are explained in Appendix B.

If it is not clear how to collapse an EC, e.g. if it looks like in Fig. 3, we can still avoid cycling by using Optimization (II), i.e. preferring exiting actions if possible. For example, if in Fig. 3 the actions d and f for state r have the same value, we prefer f.

Our implementation of optimization (I) and (II) did not result in a significant speed up, because both checking whether an EC can really be collapsed as well as checking in each step whether some action is exiting was about as costly as the gain it brought in our implementation. However, with a better implementation these optimizations are promising, as they ensure that there no longer is a probability for simulations to cycle in an EC.

Optimization (I) can also be used for the deterministic algorithm, but our experiments did not show a significant speed up there as well.

Since finding all MSEC candidates is a costly operation, Optimization (III) decreases the number of times that FIND_MSEC and DEFLATE are executed, by only calling the procedures every i steps for some i. Optimization (III) had the largest impact on the model cloud (with scaling parameter $N = 6$), as depicted in Table 1. For $i = 1$, BVI was not able to finish within 15 minutes, while for the other i it was done within approximately 2 minutes. Looking at BRTDP on this model, we see that increasing the i increased the verification time from 11 seconds for $i = 1$ to 255 for $i = 1000$. Note that depending on the combination of algorithm and model, both a small and a large i can be advantageous. Probably the number of MSECs as well as their position in the model play a role. On the one hand, if i is too small, time is wasted to repeatedly compute the current MSEC decomposition. On the other hand, if i is too large, then we might need to wait for i iterations of the main loop before we can deflate some MSEC. As in our experiments both very small and very large i (1 or 1000) brought the possibility of a huge increase in verification time, for all following experiments we chose $i = 10$. This number always yielded a verification time that was very close to the optimum we could achieve.

Finally, Optimization (IV) is a technical improvement that speeds up the computation of SECs. In Algorithm 2, we compute the MECs of the modified SG with only optimal Minimizer actions. However, note that any MEC in the modified SG is an EC in the original SG, and hence a subset of a MEC in the original SG. Thus, we do not have to look for MECs in the whole modified SG, but it suffices to perform the computation on the MECs of the original SG. By caching these original

Table 2
CPU time for each MDP experiment in seconds. For each model, there first is a row giving the name of the model and the parameter(s), and then several rows for the different parameter values we tried. We compared our deterministic and learning based approaches (BVI and BRTDP) to the deterministic and learning based approaches based on collapsing ([33] and [11]). We also compared to the explicit engine of PRISM (PRISM_e), and the hybrid engines of PRISM (PRISM_h) and the topological value iteration of Storm, all three of which are without guarantees. X indicates a timeout, W that the returned result was wrong.

Model (scaling-parameters)	With guarantees				Without guarantees		
	Deterministic		Learning-based				
	BVI	[33]	BRTDP	[11]	PRISM_e	PRISM_h	Storm
firewire (dl)							
220	418	389	5	4	389	202	125
240	634	512	5	4	491	346	163
260	894	621	5	4	617	695	186
wlan (k, COL)							
4, 2	20	17	5	4	18	12	3
4, 6	36	34	5	3	34	16	7
6, 2	715	691	5	4	760	128	53
6, 6	709	701	5	5	773	134	53
zeroconf (K, N)							
2, 20	14	9	3	7	9	7	1
2, 1000	14	10	8	11	9	7	1
10, 20	254	233	7	8	163	133	26
10, 1000	282	170	8	10	155	146	27
csma (N, K)							
2, 2	1	1	7	2	1	<1	<1
2, 6	5	3	93	39	3	1	<1
3, 2	4	3	22	12	3	1	<1
3, 6	X	X	X	X	X	47	X
leader (N)							
3	3	2	4	4	1	1	<1
4	5	3	10	9	1	1	<1
5	8	6	17	28	2	2	<1
6	20	14	50	X	8	8	3
mer (x, n)							
10^{-4}, 1500	X	X	73	15	629	83	145
10^{-4}, 3000	X	X	70	13	X	172	516
0.1, 1500	X	X	X	X	571	115	146
0.1, 3000	X	X	X	X	X	235	513
hm (N, p)							
20, 0.5	70	33	X	604	W	W	W
20, 0.9	67	33	X	585	W	W	W

MECs, we can speed up the computation drastically, especially since typically MECs are small and many states are not part of any MEC.

6.3. MDP results

We compare seven different approaches to compute the reachability probability as well as optimal strategies on MDPs. We group these algorithms as follows.

Value iteration with guarantees. There are four guaranteed value iteration approaches – two of them based on collapsing ([11] and [33]) and two based on the new idea of deflating (BRTDP and BVI). Comparing columns 2 and 3 as well as 4 and 5 of Table 2, we see that both the deterministic (BVI and [33]) and both the learning-based (BRTDP and [11]) approaches perform similarly. We conclude that collapsing and deflating are both useful for practical purposes. The collapsing based approaches are usually slightly faster. This might be the case, because after an EC is collapsed, it will not have to be considered again, while for our new approach it is deflated every time. However, the difference might also depend on other implementation details, for example the different implementations of SIMULATION_STUCK between the two learning-based approaches.

Value iteration without guarantees. We compare BVI to the usual (unguaranteed) value iteration of PRISM's explicit engine (column 6 of Table 2) and see that the guaranteed approach did not take significantly more time in most cases. In all models but zeroconf for K=10 and mer for n=1500, PRISM_e and BVI produce times in the same order of magnitude. This implies that the overhead for the computation of the guarantees often is negligible.

Table 3
The number of states for each model and the number of states that the two simulation based approaches of [11] and of this paper (BRTDP) explored. Models and their scaling parameters are denoted on the left as in Table 2. An X indicates a timeout.

Model (scaling-parameters)	#States	[11]	BRTDP
firewire (dl)			
220	10,490,495	792	751
240	13,366,666	779	702
260	15,255,584	791	596
wlan (k, COL)			
4, 2	345,118	767	199
4, 6	728,990	764	333
6, 2	5,007,666	858	116
6, 6	5,007,670	691	133
zeroconf (K, N)			
2, 20	89,586	393	125
2, 1000	89,586	1,625	898
10, 20	3,001,911	1,161	599
10, 1000	3,001,911	5,358	782
csma (N, K)			
2, 2	1,038	964	965
2, 6	66,718	64,341	33,413
3, 2	36,850	22,883	26,650
3, 6	84,856,004	X	X
leader (N)			
3	364	335	313
4	3,172	2,789	2,599
5	27,299	21,550	8,281
6	237,656	128,593	22,368
mer (x, n)			
10^{-4}, 1500	8,862,064	2,603	2,032
10^{-4}, 3000	17,722,564	2,632	2,028
0.1, 1500	8,862,064	X	X
0.1, 3000	17,722,564	X	X
hm (N, p)			
20, 0.5	41	41	X
20, 0.9	41	41	X

Note that for hm the approaches without guarantees return the wrong result (denoted by "W" in the table), e.g. PRISM_e reports 0.35 when the true result is 0.9. The model checker does not print a warning that this might have happened.

Hybrid approaches. Compared to the hybrid engine of Storm and PRISM (columns 7 and 8 in Table 2), BVI is vastly outperformed on larger models; however, this difference is because BVI explicitly constructs the whole model, while the hybrid engines can avoid this using symbolic representations. An implementation of BVI using the hybrid engine would most probably also get the speed up of this approach, and hence again be comparable. Looking at the differences between PRISM's explicit and hybrid engine we see that the gain of the different engine is much larger than the overhead for computing the guarantees.

Simulation-based approaches. The simulation based approaches BRTDP and [11] perform well on firewire, wlan and zeroconf, even outperforming Storm in some cases. For firewire, they are two orders of magnitude faster. So in certain cases, simulation based approaches can produce a huge speed-up while still giving guarantees, as already noted in [11]. However for csma, leader and mer they are not well suited, as they need to explore thousands of states to achieve convergence. For the first two rows of mer they are still faster, since the explored part of the state space is very small in comparison to the whole model and not too large in general, but as the model is scaled, the number of relevant states grows too large for the simulation based approaches to work well. In hm, the adversarial handcrafted model, every value iteration algorithm is slow due to the slow convergence of the values. Moreover, the simulation based approaches have an additional problem that increases their runtime: every time a simulation starts in the initial state, it has a high probability, around 99.99% for our choice of N, to lead back to the initial state and not reach a target or a sink. Thus, simulations are often stopped by SIMULATION_STUCK without making any progress. Intuitively, the algorithm assumes that it does not reach a target because of an EC, while in fact it is only due to the low probability to reach the goal.

The results in Table 3 show that [11] explores a larger portion of the state space for almost all experiments. This is due to the different choice of the heuristic SIMULATION_STUCK, i.e. the number of steps before a simulation is stopped. The implementation in [11] allows for simulating longer, and hence more of the state space is explored. Depending on the model

Table 4
Experimental results for the experiments on SGs. Models and their scaling parameters are denoted on the left as in Table 2. Columns 2, 3 and 4 display the verification time in seconds for each of the solvers, namely PRISM-games (referred as PRISM), our deterministic algorithm (BVI) and our learning-based algorithm (BRTDP). The next two columns compare the number of states that BRTDP explored (#States_B) to the total number of states in the model. The rightmost column shows the number of MSECs in the model. An X indicates a timeout.

Model	Model checking times			Model properties		
(scaling-parameters)	PRISM	BVI	BRTDP	#States_B	#States	#MSECs
mdsm (prop)						
1	10	14	20	895	62,245	1
2	6	9	28	471	62,245	1
cdmsn						
	3	4	9	1,213	1,240	1
teamform (N)						
3	5	7	102	7,337	12,476	0
4	10	14	X	X	96,666	0
cloud (N)						
5	6	15	19	1,311	8,842	4,421
6	9	123	12	768	34,954	17,477

structure, this can be advantageous or detrimental for the verification time. In hm, the longer simulations are necessary to have a good chance of reaching a target or sink and making progress, so here [11] outperforms BRTDP. In contrast, in leader with N=6 [11] explores 128,593 states without producing a result, while BRTDP can terminate after seeing only 22,368 states. Here, stopping a simulation early prevents the algorithm from wasting time exploring irrelevant parts of the state space or cycling in ECs for a long time.

In general one can see, that the approach of deflating works well also on MDPs and that giving guarantees often is possible without significant overhead.

6.4. SG results

Since we gave the first guaranteed value iteration approach, we performed experiments to empirically estimate the overhead for the upper bound computation. We compared both our algorithms, the deterministic BVI and the learning-based BRTDP, to the hybrid engine of PRISM-games; note that the latter does not give any guarantees on the value. The results in Table 4 show that on all our benchmarks at least one of our approaches was close to the time that the hybrid engine of PRISM-games needed. The amount of times deflate has to be executed grows when there are many MSECs, as for example in cloud. This explains the longer runtime that BVI has, especially for cloud with N=6. Luckily, for this model apparently only a small part of the state space is relevant for convergence, and thus BRTDP performs well. On the other hand, for cdmsn and teamform a large part of the state space is relevant, so that BRTDP is slow in comparison and does not even finish in time for teamform with N=4.

In conclusion, we see that deflating can be very costly, but the cost often is negligible or can be mitigated by using the learning-based approach.

7. Conclusions

We have provided the first stopping criterion for value iteration on simple stochastic games and an anytime algorithm with bounds on the current error and thus guarantees on the precision of the result. The main technical challenge was that states in end components in SGs can have different values, in contrast to the case of MDPs. We have shown that collapsing is in general not possible, but we utilized the analysis of the game graph to obtain the procedure of *deflating*, a solution on the original graph. Besides, whenever a simple end component is identified for sure it can be collapsed and the two techniques of collapsing and deflating can thus be combined.

The experiments indicate that the price to pay for the overhead to compute the error bound is often negligible. For each of the experimental models, at least one of our two implementations has performed similar to or better than the standard approach that yields no guarantees. Further, the obtained guarantees facilitate (e.g. learning-based) heuristics which treat only a part of the state space and can thus potentially lead to huge improvements. Surprisingly, already our straightforward adaptation of a learning-based algorithm from MDP to SG yields good results, sometimes palliating the overhead of our non-learning method, despite the most naive implementation of deflating. Future work could reveal whether other heuristics or more efficient implementation can lead to huge savings as in the case of MDP [11].

The concept of simple end components has already been generalized to settings with limited information [6], with multiple objectives [4] and to concurrent stochastic games [30]. It is probable that it can also be applied in settings with

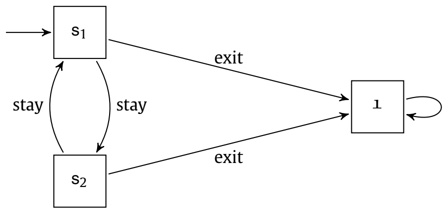


Fig. A.6. An example of an SG (also MDP) where following (ε-)optimal actions is not necessarily an (ε-)optimal strategy.

other objectives, e.g. expected reward or parity. To this end, the relationship between simple end components and tangles in parity games [29] should also be investigated.

Declaration of competing interest

This research was funded in part by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement No. 291763 for TUM – IAS, the Studienstiftung des Deutschen Volkes project "Formal methods for analysis of attack-defence diagrams", TUM IGSSE Grant 10.06 (PARSEC), and the German Research Foundation (DFG) project KR 4890/2-1 "Statistical Unbounded Verification".

Appendix A. Obtaining ε-optimal strategies

In this appendix, we show how to obtain ε-optimal strategies, given an SG $\mathsf{G} = (S, S_\square, S_\bigcirc, \mathsf{s}_0, \mathsf{A}, \mathsf{Av}, \delta)$ and vectors of under- and over approximants of the value, L respectively U. Intuitively, the strategy just plays an action which is optimal according to the approximants. However, this might not be sufficient, as the following counter-example shows.

Consider the SG (that also is an MDP) from Fig. A.6. After one update we have $\mathsf{L}(\mathsf{s}_1) = \mathsf{L}(\mathsf{s}_2) = 1$. Thus, $\arg\max_{\mathsf{a}\in\mathsf{Av}(\mathsf{s}_i)} \mathsf{L}(\mathsf{s}_i, \mathsf{a}) = \{\mathit{stay}, \mathit{exit}\}$ and L is ε-close to the true value (in fact it is equal). Both actions are indeed optimal, as after playing *stay*, the other state still can reach the target with probability 1. However, using the strategy that in both states picks the *stay*-action results in probability 0 to reach the target. Intuitively, the problem is that picking ε-optimal actions only guarantees that they are one-step ε-optimal, but does not ensure that the target is actually reached in the end. Thus, in order to obtain ε-optimal strategies, we additionally have to ensure that the strategies "make progress" towards the target.

A way to achieve this would be to randomize over all ε-optimal actions; however, this is suboptimal, as theoretically it needs the additional power of randomization and practically the strategy has a chance of cycling unnecessarily by using the actions that stay instead of making progress.

In [7, Remark 10.104], the authors observe that L_i corresponds to the i-step-bounded reachability probability (for MDPs, but the argument extends to SGs). They suggest that a strategy obtaining this value can be generated by choosing an optimal action according to L_i in the first state, then according to L_{i-1} in the next step and so on.[12] However, note that this strategy requires memory as well as knowledge of all intermediate results L_i.

We now describe a way to obtain a memoryless deterministic ε-optimal strategy for Maximizer and Minimizer, using only the final approximants L respectively U. We also prove the correctness of our strategies. Concretely, we want to find a σ such that for all τ: $\mathbb{P}^{\sigma,\tau}_{\mathsf{s}_0}(\Diamond F) > \mathsf{V}(\mathsf{s}_0) - \varepsilon$. Dually we also want a τ such that for all σ: $\mathbb{P}^{\sigma,\tau}_{\mathsf{s}_0}(\Diamond F) < \mathsf{V}(\mathsf{s}_0) + \varepsilon$. Intuitively, for the Minimizer it suffices to just play actions which are optimal according U. For the Maximizer, we play optimally according to L, but additionally have to ensure that the actions make progress; we achieve this by performing a backwards search from the target states and picking only those actions that decrease the distance to the targets.

A.1. Assumptions on the approximants

We first state several assumptions which the approximants have to satisfy and for each assumption prove that it is satisfied for approximants coming from our algorithm. For ease of notation, we use the operator arg opt to find the set of optimal actions for a state s:

$$\operatorname*{arg\,opt}_{\mathsf{a}\in\mathsf{Av}(\mathsf{s})} := \begin{cases} \arg\max_{\mathsf{a}\in\mathsf{Av}(\mathsf{s})} & \text{if } \mathsf{s} \in S_\square \\ \arg\min_{\mathsf{a}\in\mathsf{Av}(\mathsf{s})} & \text{if } \mathsf{s} \in S_\bigcirc \end{cases}$$

- The approximants are correct, i.e. for all states $\mathsf{L}(\mathsf{s}) \le \mathsf{V}(\mathsf{s}) \le \mathsf{U}(\mathsf{s})$. This assumption is satisfied for our algorithm by Theorem 2.

[12] In the previous example, for all $i > 1$, both *stay* and *exit* are optimal, while for $i = 1$ only *exit* is optimal. Thus, this strategy ensures that the target is reached.

- The approximants are ε-close in the initial state, i.e. $\mathsf{V}(\mathsf{s}_0) - \mathsf{L}(\mathsf{s}_0) < \varepsilon$ and $\mathsf{U}(\mathsf{s}_0) - \mathsf{V}(\mathsf{s}_0) < \varepsilon$. This is satisfied for our algorithm, as the bounds are correct and the stopping criterion ensures $\mathsf{U}(\mathsf{s}_0) - \mathsf{L}(\mathsf{s}_0) < \varepsilon$.
- For all target states the lower approximant is 1. This assumption is satisfied by the initialization of the algorithm.
- Unfolding the Bellman equation for the approximants can only make them more precise, i.e. for all states s:

$$\mathsf{L}(\mathsf{s}) \leq \arg\operatorname{opt}_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{L}(\mathsf{s}')$$

$$\mathsf{U}(\mathsf{s}) \geq \arg\operatorname{opt}_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{U}(\mathsf{s}')$$

We first prove this assumption for the lower approximants. Let i be the iteration in which the algorithm computing the approximants finished. We denote by L_j the approximants in the j-th iteration for all $j \leq i$. Then we have $\mathsf{L} = \mathsf{L}_i$ by definition. The lower approximant is computed according to the Bellman equations (3) and (4). Thus for all states s we have

$$\begin{aligned}\mathsf{L}_i(\mathsf{s}) &:= \arg\operatorname{opt}_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{L}_{i-1}(\mathsf{s}') \\ &\leq \arg\operatorname{opt}_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{L}_i(\mathsf{s}'),\end{aligned}$$

where the second inequality follows the fact that the bounds are monotonically increasing, i.e. $\mathsf{L}_{i-1}(\mathsf{s}) \leq \mathsf{L}_i(\mathsf{s})$.
For the upper approximant, the proof is mostly analogous, replacing $\leq$ with $\geq$. However, we also have to address the additional deflating that can modify the upper approximants. If a state s is deflated down to x, then it is part of an EC T and all states in the T were deflated to x as well. As T is an EC, every $\mathsf{s} \in T$ has a staying action a_s. Since all successors of this action are in the EC and have an upper estimate of x, we have $\mathsf{U}(\mathsf{s}, \mathsf{a}_s) = x$. Hence, if s is a Minimizer state, we have $\mathsf{U}(\mathsf{s}) = x$ and $\arg\min_{\mathsf{a}' \in \mathsf{Av}(\mathsf{s})} \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{U}(\mathsf{s}') \leq \mathsf{U}(\mathsf{s}, \mathsf{a}_s) = x$ and the assumption is satisfied. If s is a Maximizer state, we also use the fact that $x = \mathsf{bestExit}_{\mathsf{U}}^{\Box}(T)$, and thus for all leaving actions a_ℓ we have $\mathsf{U}(\mathsf{s}, \mathsf{a}_\ell) \leq x$. Then, as all staying actions have an estimate of x and all leaving actions have at most x, we can conclude that $\arg\max_{\mathsf{a}' \in \mathsf{Av}(\mathsf{s})} \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{U}(\mathsf{s}') = x$.
Note that given the true values V, unfolding the Bellman equations for one step results in exactly the same values, as the value vector is a fixpoint. Thus, in that case we have an equality.

A.2. Strategies following only the approximants

Let σ^{L} and τ^{U} be the strategies that pick an action that is optimal according to the approximants L respectively U. Formally, for all states s, let $\sigma^{\mathsf{L}}(\mathsf{s})$ be an element of $\arg\max_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{L}(\mathsf{s}')$. Dually, let $\tau^{\mathsf{U}}(\mathsf{s})$ be an element of $\arg\min_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{U}(\mathsf{s}')$.

We now prove that following these strategies for a finite number of steps does not decrease the probability to reach the target. For this, let $\Diamond^i X := \{\rho \in (S \times \mathsf{A})^\omega \mid \rho = \mathsf{s}_0 \mathsf{a}_0 \mathsf{s}_1 \mathsf{a}_1 \cdots \wedge\ \mathsf{s}_i \in X\}$ denote the measurable set of all paths which reach a state from the set $X \subseteq S$ after exactly i steps. We mention two technical details: firstly, we only require that X is visited after exactly i steps. We do *not* require that this is the first visit to X. Secondly, this definition only restricts a finite prefix of the path; however, as we have defined the probability distribution only over infinite paths, this set contains infinite paths.

Lemma 7 *(Following the approximants for a finite time). For all states* $\mathsf{s} \in S$ *and all* $i \in \mathbb{N}_0$, *the following two statements hold. For all Minimizer strategies* τ:

$$\sum_{\mathsf{s}' \in S} \mathbb{P}_{\mathsf{s}}^{\sigma^{\mathsf{L}}, \tau}[\Diamond^i \{\mathsf{s}'\}] \cdot \mathsf{L}(\mathsf{s}') \geq \mathsf{L}(\mathsf{s}).$$

Dually, for all Maximizer strategies σ:

$$\sum_{\mathsf{s}' \in S} \mathbb{P}_{\mathsf{s}}^{\sigma, \tau^{\mathsf{U}}}[\Diamond^i \{\mathsf{s}'\}] \cdot \mathsf{U}(\mathsf{s}') \leq \mathsf{U}(\mathsf{s}).$$

Proof. We show only the first statement, as the second proof is completely analogous. We proceed by induction on i, the number of steps for which we follow σ^{L}. For $i = 0$,

$$\mathbb{P}_{\mathsf{s}}^{\sigma^{\mathsf{L}}, \tau}[\Diamond^0 \{\mathsf{s}'\}] = \begin{cases} 1 & \text{if } \mathsf{s} = \mathsf{s}' \\ 0 & \text{otherwise} \end{cases}.$$

Thus, the left side of the inequality evaluates to $\mathsf{L}(\mathsf{s})$ and the base case is completed.

For the induction step, the following chain of equations proves our goal. We justify every transformation below the equations.

$$\begin{aligned}
&\sum_{\mathsf{s}' \in S} \mathbb{P}_{\mathsf{s}}^{\sigma^{\mathsf{L}},\tau}[\Diamond^{i+1}\{\mathsf{s}'\}] \cdot \mathsf{L}(\mathsf{s}') \\
&= \sum_{t \in S} \mathbb{P}_{\mathsf{s}}^{\sigma^{\mathsf{L}},\tau}[\Diamond^{i}\{t\}] \cdot \left(\sum_{\mathsf{s}' \in S} \delta(t, \pi(t), \mathsf{s}') \cdot \mathsf{L}(\mathsf{s}') \right) && \text{(Step 1)} \\
&\geq \sum_{t \in S} \mathbb{P}_{\mathsf{s}}^{\sigma^{\mathsf{L}},\tau}[\Diamond^{i}\{t\}] \cdot \mathsf{L}(t) && \text{(Step 2)} \\
&\geq \mathsf{L}(\mathsf{s}) && \text{(Induction hypothesis)}
\end{aligned}$$

For Step 1, instead of considering paths that follow the strategies for $i+1$ steps, we consider paths that follow it for i steps and explicitly unfold the final step. Here, $\pi(\mathsf{s}') = \begin{cases} \sigma^{\mathsf{L}}(\mathsf{s}') & \text{if } \mathsf{s} \in S_{\Box} \\ \tau(\mathsf{s}') & \text{if } \mathsf{s} \in S_{\bigcirc} \end{cases}$.

For Step 2, we use the fourth assumption on the approximants, namely that unfolding the Bellman equation can only make them more precise. If t is a Maximizer state, by definition we have

$$\sigma^{\mathsf{L}}(t) = \arg\max_{\mathsf{a} \in \mathsf{Av}(\mathsf{t})} \sum_{\mathsf{s}' \in S} \delta(t, \mathsf{a}, \mathsf{s}') \cdot \mathsf{L}(\mathsf{s}')$$

and thus can immediately apply the assumption. If t is a Minimizer state, we know that the assumption holds for arg min of all available actions, so it also holds for whichever action τ uses. In the final step, we apply the induction hypothesis. □

Lemma 8. *τ^{U} is an ε-optimal strategy of Minimizer.*

Proof. Using Lemma 7, we have that for all states s, all Maximizer strategies σ and all $i \in \mathbb{N}_0$:

$$\begin{aligned}
&\sum_{\mathsf{s}' \in S} \mathbb{P}_{\mathsf{s}}^{\sigma,\tau^{\mathsf{U}}}[\Diamond^{i}\{\mathsf{s}'\}] \cdot \mathsf{U}(\mathsf{s}') \\
&= \left(\sum_{\mathsf{s}' \in F} \mathbb{P}_{\mathsf{s}}^{\sigma,\tau^{\mathsf{U}}}[\Diamond^{i}\{\mathsf{s}'\}] \cdot \mathsf{U}(\mathsf{s}') \right) + \left(\sum_{\mathsf{s}' \in S \setminus F} \mathbb{P}_{\mathsf{s}}^{\sigma,\tau^{\mathsf{U}}}[\Diamond^{i}\{\mathsf{s}'\}] \cdot \mathsf{U}(\mathsf{s}') \right) && \text{(Splitting the sum)} \\
&\leq \mathsf{U}(\mathsf{s}) && \text{(Lemma 7)}
\end{aligned}$$

The second sum over all non-target states certainly evaluates to 0 or more. Also, $\mathsf{U}(\mathsf{s}') = 1$ for all $\mathsf{s}' \in F$. Hence, it holds that

$$\sum_{\mathsf{s}' \in F} \mathbb{P}_{\mathsf{s}}^{\sigma,\tau^{\mathsf{U}}}[\Diamond^{i}\{\mathsf{s}'\}] \leq \mathsf{U}(\mathsf{s}).$$

Thus, pulling the sum into the definition of the paths, we get

$$\mathbb{P}_{\mathsf{s}}^{\sigma,\tau^{\mathsf{U}}}[\Diamond^{i} F] \leq \mathsf{U}(\mathsf{s}).$$

Note that as this statement holds for all i and all target states are absorbing, we can also take the limit and consider all paths that reach a target state $\Diamond F$. Considering the initial state s_0 and using that U is correct and ε-optimal, we conclude that τ^{U} is ε-optimal:

$$\mathbb{P}_{\mathsf{s}_0}^{\sigma,\tau^{\mathsf{U}}}[\Diamond F] \leq \mathsf{U}(\mathsf{s}_0) \leq \mathsf{V}(\mathsf{s}_0) + \varepsilon. \quad \Box$$

A.3. Ensuring maximizer makes progress

We already saw in the counter-example in the beginning that the dual of Lemma 8 is not true, as only playing actions according to the approximants is not ε-optimal. We will now show how to augment σ^{L} to become ε-optimal.

We employ a folklore construction which is sometimes called *attractor*, similar[13] to the one used in [15, Section 5.3]. In other words: we perform a backwards search from the target state, using only actions that are ε-optimal according to L.

[13] In that paper, they ensure that the target or sink states are reached almost surely. We require that target states are reached with positive probability.

Formally, we use the following recursive procedure. Let $S_0^{\text{attr}} = F$. For $i \geq 1$, we define

$$S_i^{\text{attr}} = \{\mathsf{s} \in S_\square \mid \exists \mathsf{a} \in \operatorname*{arg\,max}_{\mathsf{a}' \in \mathsf{Av}(\mathsf{s})} \mathsf{L}(\mathsf{s}, \mathsf{a}').\mathsf{Post}(\mathsf{s}, \mathsf{a}) \cap S_{i-1}^{\text{attr}} \neq \emptyset\}$$
$$\cup \{\mathsf{s} \in S_\bigcirc \mid \forall \mathsf{a} \in \mathsf{Av}(\mathsf{s}).\mathsf{Post}(\mathsf{s}, \mathsf{a}) \cap S_{i-1}^{\text{attr}} \neq \emptyset\}.$$

We terminate when $S_j^{\text{attr}} = S_{j-1}^{\text{attr}}$.

For all i, S_i^{attr} contains those states that have a path of length at most i to the target which for Maximizer states uses only ε-optimal actions, and for Minimizer states has no other choice but to continue on a path towards the target. This can be proven by simple induction on the length of the path. Note that the states with no path to the target are not in S_i^{attr} for any i.

Let $\mathcal{P} := \{\mathsf{s} \in S \mid \mathsf{V}(\mathsf{s}) > 0\}$ be all states with a positive value. As every state with a positive value has a path to a target state, it holds that $\mathcal{P} = S_j^{\text{attr}}$, where j is the iteration where the recursive procedure terminates.

We now define the ε-optimal Maximizer strategy $\sigma^{\mathsf{L},\text{attr}}$. Note that for states in F or states with value 0, the strategy is irrelevant as either we anyway have reached target or no strategy can reach the target (against a rational opponent). Thus, here $\sigma^{\mathsf{L},\text{attr}}$ picks an arbitrary available action. For every state in $\mathsf{s} \in \mathcal{P} \setminus F$, we pick an action as follows: let i be the minimum number with $\mathsf{s} \in S_i^{\text{attr}}$. As $\mathsf{s} \in \mathcal{P}$, such an i exists. Then the strategy $\sigma^{\mathsf{L},\text{attr}}(\mathsf{s})$ picks an action from

$$\{\mathsf{a} \in \mathsf{Av}(\mathsf{s}) \mid \mathsf{a} \in \operatorname*{arg\,max}_{\mathsf{a}' \in \mathsf{Av}(\mathsf{s})} \mathsf{L}(\mathsf{s}, \mathsf{a}') \wedge \mathsf{Post}(\mathsf{s}, \mathsf{a}) \cap S_{i-1}^{\text{attr}} \neq \emptyset\}.$$

Such an action exists, because s was added in the i-th step, hence it must have an action leading to S_{i-1}^{attr} by definition of S_i^{attr}. Note that not all successors of the action have to be in S_{i-1}^{attr}, but we require only at least one in order to have a positive probability of making progress. Further, note that $i - 1 \geq 0$, since $\mathsf{s} \notin F$, so the strategy is well defined.

Under this strategy $\sigma^{\mathsf{L},\text{attr}}$, for every Minimizer strategy τ, every state in $\mathsf{s} \in \mathcal{P} \setminus F$ has a positive probability to reach the target states. To prove this, observe that every s is in some S_i^{attr}. For a Maximizer state, by construction it uses an action with a positive chance to reach a state in S_{i-1}^{attr}; for Minimizer, it has no choice but to put positive probability on reaching S_{i-1}^{attr}. Repeating the argument until $i = 0$ shows the claim.

Lemma 9. *$\sigma^{\mathsf{L},attr}$ is an ε-optimal strategy of Maximizer.*

Proof. We again use Lemma 7 and split the sum as in the proof of Lemma 8. However, this time we split S into F, $\mathcal{P}$ and the remaining states with value 0. Note that for states with value 0 also the lower approximant is 0, since by assumption the lower approximant is correct. Hence, the summand considering the value 0 states evaluates to 0. Thus we have that for all states s, all Minimizer strategies τ and all $i \in \mathbb{N}_0$:

$$\left(\sum_{\mathsf{s}' \in F} \mathbb{P}_{\mathsf{s}}^{\sigma^{\mathsf{L},\text{attr}},\tau}[\Diamond^i\{\mathsf{s}'\}] \cdot \mathsf{L}(\mathsf{s}')\right) + \left(\sum_{\mathsf{s}' \in \mathcal{P} \setminus F} \mathbb{P}_{\mathsf{s}}^{\sigma^{\mathsf{L},\text{attr}},\tau}[\Diamond^i\{\mathsf{s}'\}] \cdot \mathsf{L}(\mathsf{s}')\right) \geq \mathsf{L}(\mathsf{s})$$

Here is the crucial difference between constructing a Maximizer and Minimizer strategy: for Maximizer, it is possible that the whole probability mass still remains in $\mathcal{P}$. Following a strategy that maximizes L for a finite number of steps does not decrease the chance of reaching the targets, but it might be necessary to eventually switch to the optimal strategy in order to actually realize the value. This is why we used the attractor construction to ensure that under $\sigma^{\mathsf{L},\text{attr}}$ every state in $\mathcal{P} \setminus F$ has a positive probability to reach the target. Then, for $i \to \infty$, the second summand converges to 0. Intuitively, more and more probability leaves towards the target or sink states. Formally, as from every state there exists a path with positive probability to reach a target, the probability to reach F in $|S|$ steps is at least $q := p^{|S|}$, where p is the minimum probability occurring in the SG. Since $\sum_i^\infty q \cdot (1-q)^i = 1$ (geometric series), the probability to remain in $\mathcal{P} \setminus F$ is 0.

Having ensured that the probability mass actually arrives at the target, we can consider the infinite behaviour and conclude:

$$\mathbb{P}_{\mathsf{s}}^{\sigma^{\mathsf{L},\text{attr}},\tau}[\Diamond F] \geq \mathsf{L}(\mathsf{s})$$

Note that we again pulled the sum into the definition of the paths and used that for target states the lower approximant is initialized to 1. Finally, considering the initial state and using that L is correct and ε-close, we arrive at our goal:

$$\mathbb{P}_{\mathsf{s}_0}^{\sigma^{\mathsf{L},\text{attr}},\tau}[\Diamond F] \geq \mathsf{L}(\mathsf{s}_0) \geq \mathsf{V}(\mathsf{s}_0) - \varepsilon. \quad \square$$

Note that, given the true values V, all inequalities in the proofs become equalities. Thus, we also showed how to construct optimal strategies from the true value vector.

Appendix B. Definition of COLLAPSE

The way in which we define COLLAPSE is not only able to collapse ECs in MDPs, but also simple ECs (SEC, see Definition 5) in SGs. Note that every EC in an MDP is a SEC. If there are no actions of Maximizer leaving the SEC, we have to keep staying actions, so the SEC becomes an absorbing state (we do not want a state without actions, as we assumed the game is non-blocking). Otherwise all staying actions are removed and the SEC becomes a single state, whose available actions are all the exiting actions of Maximizer states in the SEC. Note that we have to remove Minimizer actions, because otherwise we would allow Maximizer to use an exit that Minimizer wants to prevent.

Definition 6 *(*COLLAPSE*).* Let $\mathsf{G} = (S, S_\square, S_\bigcirc, \mathsf{s}_0, \mathsf{A}, \mathsf{Av}, \delta)$ be an SG and T a SEC in G. Then $\mathsf{COLLAPSE}(\mathsf{G}, T) = \mathsf{G}' = (S', S'_\square, S'_\bigcirc, \mathsf{s}'_0, \mathsf{A}', \mathsf{Av}', \delta')$, where G' is defined as follows:

- $S' = (S \setminus T) \cup \{\mathsf{s}_T\}$
- $S'_\square = (S_\square \setminus T) \cup \{\mathsf{s}_T\}$
- $S'_\bigcirc = S_\bigcirc \setminus T$
- $\mathsf{s}'_0 = \begin{cases} \mathsf{s}_T & \text{if } \mathsf{s}_0 \in T \\ \mathsf{s}_0 & \text{otherwise} \end{cases}$
- $\mathsf{A}' = \mathsf{A} \cup \{\bot\}$, where $\bot \notin \mathsf{A}$ is a new action.
- $\mathsf{Av}'(\mathsf{s})$ is defined for all $\mathsf{s} \in S'$ by:
  - $\mathsf{Av}(\mathsf{s})$,
    if $\mathsf{s} \in (S \setminus T)$, i.e. $\mathsf{s} \neq \mathsf{s}_T$
    (Rest stays the same)
  - $\bigcup_{t \in T_\square} \{\mathsf{a} \in \mathsf{Av}(t) \mid (t, \mathsf{a}) \text{ exits } T\}$,
    if $\mathsf{s} = \mathsf{s}_T$ and $\exists t \in T_\square : (t, \mathsf{a}) \text{ exits } T$
    (Keep leaving Maximizer actions, if there is an exit for Maximizer)
  - $\{\bot\}$,
    if $\mathsf{s} = \mathsf{s}_T$ and $\neg\exists t \in T_\square : (t, \mathsf{a}) \text{ exits } T$
    (Keep a staying action, if there is no exit for Maximizer)
- δ' is defined for all $\mathsf{s} \in S'$ and $\mathsf{a} \in \mathsf{Av}'(\mathsf{s})$ by:
  - $\delta'(\mathsf{s}, \mathsf{a})(\mathsf{s}') = \delta(\mathsf{s}, \mathsf{a})(\mathsf{s}')$
    for all $\mathsf{s}' \in S'$ with $\mathsf{s}, \mathsf{s}' \neq \mathsf{s}_T$
    (Rest stays the same)
  - $\delta'(\mathsf{s}, \mathsf{a})(\mathsf{s}_T) = \sum_{\mathsf{s}' \in T} \delta(\mathsf{s}, \mathsf{a})(\mathsf{s}')$,
    if $\mathsf{s} \neq \mathsf{s}_T$
    (going to T)
  - $\delta'(\mathsf{s}_T, \mathsf{a})(\mathsf{s}') = \delta(\mathsf{s}, \mathsf{a})(\mathsf{s}')$
    for all $\mathsf{a} \in \mathsf{Av}'(\mathsf{s}_T)$, the unique $\mathsf{s} \in T$ with $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$ and all $\mathsf{s}' \in S' \setminus \{\mathsf{s}_T\}$
    (leaving from T)
  - $\delta'(\mathsf{s}_T, \mathsf{a})(\mathsf{s}_T) = \sum_{t \in T} \delta(\mathsf{s}, \mathsf{a})(t)$
    for all $\mathsf{a} \in \mathsf{Av}'(\mathsf{s}_T)$ and the unique $\mathsf{s} \in T$ with $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$
    (staying in T when using an exit)
  - $\delta'(\mathsf{s}_T, \bot)(\mathsf{s}_T) = 1$
    if $\bot \in \mathsf{Av}'(\mathsf{s}_T)$
    (staying in T in the case of no Maximizer exits)

Note that when defining δ in the case of leaving from T, we assume that no action is available for multiple states in T. If there are duplicates, we rename the actions before collapsing, e.g. by indexing them with their respective state.

When computing the reachability probability of reaching a target set F, we also have to adjust the target set as follows: $F' = (F \setminus T) \cup \begin{cases} \emptyset & \text{if } T \cap F = \emptyset \\ \{\mathsf{s}_T\} & \text{otherwise} \end{cases}$.

We now argue about the correctness of this definition. Firstly, note that G' is well-defined. All states in $S \setminus T$ remain a part of the state space, and the controlling player as well as the available actions remain unchanged. All states from T have been replaced by the newly added s_T. To argue about the transition function, we make a case distinction:

- Going to T: if an action previously lead to a state $\mathsf{s}' \in T$, now this probability mass leads to s_T. Note that an action might have multiple different successors in T, thus we sum over all $\mathsf{s}' \in T$ when defining $\delta'(\mathsf{s}, \mathsf{a})(\mathsf{s}_T)$. Thus, δ' still is a valid probability distribution, since all probability mass previously going to a state in T now goes to s_T.

- Using an exit from T: the third and fourth items in the definition of δ' deal with the case that we use an action $\mathsf{a} \in \mathsf{Av}'(\mathsf{s}_T)$ that was an exit from T. Note that for this case we use the additional assumption that — possibly by renaming actions — there were no states $\mathsf{s}, \mathsf{s}' \in T$ such that $\mathsf{a} \in \mathsf{Av}(\mathsf{s}) \cap \mathsf{Av}(\mathsf{s}')$, i.e. actions are unique for the states in T. So for an action a there is a unique state-action pair (s, a) with $\mathsf{s} \in T$ that identifies this exit. Thus, for transitions leading to states s' outside of T, we can reuse the old transition function $\delta(\mathsf{s}, \mathsf{a})(\mathsf{s}')$; this is done by the third item. For transitions leading back into T (which is possible, since for a to be an exit we do not require all successors to be outside of T, but only at least one of them), we use the same idea as in the previous item: we sum the probability of all transitions leading to any state $t \in T$ and assign it to s_T; this is done by the fourth item. Thus, δ' still is a valid probability distribution, since (i) it is defined for every available action of s_T, and (ii) all probability mass leaving T remains unchanged, and all probability mass staying in T now leads to s_T.
- If there is no exit for Maximizer in T, formally $\neg\exists t \in T_\square : (t, \mathsf{a})\,\text{exits}\;\, T$, then we use the newly added action $\perp$ that loops surely.

Appendix C. Description of the experimental models

Our experiments are based on the ones that were conducted in [58]. Most models we use are also analyzed in that thesis, and we obtained them from the website,[14] where the author of the thesis made them available for download. We used the exact models from that website, but partly modified the properties to be of a form that our implementation can handle. These modifications did not change the semantics of the property, e.g. instead of formulating a property that a probability is greater than a certain number (P>0.999) we compute the maximal probability (Pmax=?), and then manually check whether it is greater than the number. The only models not from [58] are the MDPs csma and leader and the MC hm.

We consider six MDP models, namely firewire, wlan, zeroconf, csma, leader and mer. The first five are part of the PRISM benchmark suite [45], mer is from [31]. The four SG models are mdsm, cdmsn, team-form and cloud, and the first three are contained in the PRISM-games case studies.[15] Cloud is from [13]. Note that some of the SG models actually contain more than two players. However, since there are at most two coalitions, they can be viewed as an SG with only two players. We will now shortly describe all models, the properties we check and the parameters we use for scaling.

firewire [48]:
This case study models the protocol known as FireWire, which is a leader election protocol of the IEEE 1394 High Performance Serial Bus. Several devices connected to a bus can use the protocol to dynamically elect a leader. We compute the probability Pmax=? [Fleader_elected], so the maximal probability with which a leader gets elected before the deadline. By this one can check the property that a leader gets elected with a certain, optimally high, probability. To scale the model up, we raise the deadline.

wlan [47]:
This model describes the two-way handshake mechanism of the IEEE 802.11 medium access control (WLAN protocol). Two stations try to communicate with each other; however, if both of them send at once, a collision occurs. We are interested in computing the maximum probability that both stations transmit their messages correctly, i.e. Pmax=? [F s1=12 & s2=12], where s1 and s2 describe the state of the stations, and 12 is the final state where the transmission was successful. To scale the model up, we increase the maximal backoff k and the maximal number of collisions COL.

zeroconf [46]:
Zeroconf is a protocol for dynamically assigning an IP address to a device, provided that several other hosts have already blocked some IP addresses. The device picks some IP randomly and then sends probes to check whether this address is already in use. The parameters that we use to scale the model are N, the number of other hosts already possessing an IP address and K, the number of probes sent. The probability we are interested in is Pmin=? [F configured], so the minimum probability with which the device obtains an IP address.

csma: [45]
This case study concerns the IEEE 802.3 CSMA/CD (Carrier Sense, Multiple Access with Collision Detection) protocol. N is the number of stations and K is the maximum backoff. Pmin=? [F min_backoff_after_success<=K] is the probability we are interested in, namely that a message of some station is eventually delivered before K backoffs.

leader [45]:
This case study is based on the asynchronous leader election protocol of [37]. This protocol solves the following problem. Given an asynchronous ring of N processors design a protocol such that they will be able to elect a leader (a uniquely designated processor) by sending messages around the ring. The probability we are interested in is Pmax=? [F "elected"], so that at some point a leader is elected.

mer [31]:
In the Mars Exploration Rover there is a resource arbiter that handles distributing resources to different users. There is a probability x that the communication between the arbiter and the users fails. We change this probability to influence the

[14] http://www.prismmodelchecker.org/files/thesismujma/.
[15] `prismmodelchecker.org/games/casestudies.php`.

structure of the MDP. The probability we compute is Pmax=? [F err_G], which is the maximum probability that an error occurs.

hm [33]:
An example that was handcrafted to show when value iteration without guarantees fails. It consists of two chains of states of length n that lead to the target and a sink state; however, at every position in the chain there is some probability to go back to the initial state. This also is an adversarial example for simulation based algorithms, as the probability to reach a sink state and make any progress the simulation has to be lucky n times in a row, which is unlikely.

mdsm [20]:
This case study models multiple households which all consume different amounts of energy over time. To minimize the peak energy consumption, they utilize the distributed energy management "Microgrid Demand-Side Management" (mdsm). The property we check is the maximal probability with which the first household can deviate from the management algorithm, i.e. Pmax=? [F deviated], which should be smaller than 0.01. We check the property once for player 1 and once for player 2.

cdmsn [20,54]:
This model describes a set of agents which have different sites available and different preferences over these sites. The collective decision making algorithm of this case study is utilized so that the agents agree on one decision. We analyse the model to find the strategy for player 1 to make the agents agree on the first site with a high probability, so <<p1>> Pmax=? [F all_prefer_1].

team-form [21]:
As in the previous case study, there is a set of agents in a distributed environment. They need to form teams so they are able to perform a set of tasks together. We want to compute a strategy so that the first task is completed with the maximal possible probability, so we check the property <<p1,p2,p3>> Pmax=? [F task1_completed]. The model can be scaled using the number of agents N. However, for this model PRISM's pre-computations already solve the problem and hence it cannot be used to compare the approaches; thus, it is not included in any of our tables.

cloud [13]:
This model describes several servers and virtual machines forming a cloud system. The controller of the system wants to deploy a web application on one of the virtual machines, but it is possible that the servers fail due to hardware failures. We compute the strategy and the maximal probability for the controller to successfully deploy his software, so <<controller>> Pmax=? [F deployed]. The model can be scaled by increasing the number of virtual machines N.

References

[1] D. Andersson, P.B. Miltersen, The complexity of solving stochastic games on graphs, in: ISAAC, Springer, 2009, pp. 112–121.

[2] G. Arslan, S. Yüksel, Decentralized Q-learning for stochastic teams and games, IEEE Trans. Autom. Control 62 (2017) 1545–1558, https://doi.org/10.1109/TAC.2016.2598476.

[3] P. Ashok, K. Chatterjee, P. Daca, J. Kretínský, T. Meggendorfer, Value iteration for long-run average reward in Markov decision processes, in: CAV (1), Springer, 2017, pp. 201–221.

[4] P. Ashok, K. Chatterjee, J. Kretínský, M. Weininger, T. Winkler, Approximating values of generalized-reachability stochastic games, in: LICS, ACM, 2020, pp. 102–115.

[5] P. Ashok, P. Daca, J. Kretínský, M. Weininger, Statistical model checking: black or white?, in: ISoLA (1), Springer, 2020, pp. 331–349.

[6] P. Ashok, J. Kretínský, M. Weininger, PAC statistical model checking for Markov decision processes and stochastic games, in: CAV (1), Springer, 2019, pp. 497–519.

[7] C. Baier, J. Katoen, Principles of Model Checking, MIT Press, 2008.

[8] C. Baier, J. Klein, L. Leuschner, D. Parker, S. Wunderlich, Ensuring the reliability of your model checker: interval iteration for Markov decision processes, in: CAV (1), Springer, 2017, pp. 160–180.

[9] N. Balaji, S. Kiefer, P. Novotný, G.A. Pérez, M. Shirmohammadi, On the complexity of value iteration, in: ICALP, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 102:1–102:15.

[10] R.I. Brafman, M. Tennenholtz, A near-optimal polynomial time algorithm for learning in certain classes of stochastic games, Artif. Intell. 121 (2000) 31–47, https://doi.org/10.1016/S0004-3702(00)00039-4.

[11] T. Brázdil, K. Chatterjee, M. Chmelik, V. Forejt, J. Kretínský, M.Z. Kwiatkowska, D. Parker, M. Ujma, Verification of Markov decision processes using learning algorithms, in: ATVA, Springer, 2014, pp. 98–114.

[12] L. Busoniu, R. Babuska, B.D. Schutter, A comprehensive survey of multiagent reinforcement learning, IEEE Trans. Syst. Man Cybern. Part C 38 (2008) 156–172, https://doi.org/10.1109/TSMCC.2007.913919.

[13] R. Calinescu, S. Kikuchi, K. Johnson, Compositional reverification of probabilistic safety properties for large-scale complex IT systems, in: Monterey Workshop, Springer, 2012, pp. 303–329.

[14] J. Cámara, G.A. Moreno, D. Garlan, Stochastic game analysis and latency awareness for proactive self-adaptation, in: SEAMS, ACM, 2014, pp. 155–164.

[15] K. Chatterjee, L. de Alfaro, T.A. Henzinger, Strategy improvement for concurrent reachability and turn-based stochastic safety games, J. Comput. Syst. Sci. 79 (2013) 640–657, https://doi.org/10.1016/j.jcss.2012.12.001.

[16] K. Chatterjee, N. Fijalkow, A reduction from parity games to simple stochastic games, in: GandALF, 2011, pp. 74–86.

[17] K. Chatterjee, M. Henzinger, Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification, in: SODA, SIAM, 2011, pp. 1318–1336.

[18] K. Chatterjee, T.A. Henzinger, Value iteration, in: O. Grumberg, H. Veith (Eds.), 25 Years of Model Checking - History, Achievements, Perspectives, Springer, 2008, pp. 107–138.

[19] K. Chatterjee, T.A. Henzinger, B. Jobstmann, A. Radhakrishna, Gist: a solver for probabilistic games, in: CAV, 2010, pp. 665–669.

[20] T. Chen, V. Forejt, M.Z. Kwiatkowska, D. Parker, A. Simaitis, Automatic verification of competitive stochastic systems, Form. Methods Syst. Des. 43 (2013) 61–92, https://doi.org/10.1007/s10703-013-0183-7.

[21] T. Chen, M.Z. Kwiatkowska, D. Parker, A. Simaitis, Verifying team formation protocols with probabilistic model checking, in: CLIMA, Springer, 2011, pp. 190–207.

[22] T. Chen, M.Z. Kwiatkowska, A. Simaitis, C. Wiltsche, Synthesis for multi-objective stochastic games: an application to autonomous urban driving, in: QEST, 2013, pp. 322–337.
[23] C. Cheng, A.C. Knoll, M. Luttenberger, C. Buckl, GAVS+: an open platform for the research of algorithmic game solving, in: TACAS, Springer, 2011, pp. 258–261.
[24] A. Condon, The complexity of stochastic games, Inf. Comput. 96 (1992) 203–224, https://doi.org/10.1016/0890-5401(92)90048-K.
[25] A. Condon, On algorithms for simple stochastic games, in: Advances in Computational Complexity Theory, DIMACS/AMS, 1993, pp. 51–71.
[26] P. Daca, T.A. Henzinger, J. Kretínský, T. Petrov, Faster statistical model checking for unbounded temporal properties, ACM Trans. Comput. Log. 18 (2017) 12:1–12:25, https://doi.org/10.1145/3060139.
[27] B.A. Davey, H.A. Priestley, Introduction to Lattices and Order, Cambridge University Press, 2002.
[28] C. Dehnert, S. Junges, J. Katoen, M. Volk, A storm is coming: a modern probabilistic model checker, in: CAV (2), Springer, 2017, pp. 592–600.
[29] T. van Dijk, Attracting tangles to solve parity games, in: CAV (2), Springer, 2018, pp. 198–215.
[30] J. Eisentraut, J. Kretínský, A. Rotar, Stopping criteria for value and strategy iteration on concurrent stochastic reachability games, CoRR, arXiv:1909.08348 [abs], http://arxiv.org/abs/1909.08348, 2019.
[31] L. Feng, M.Z. Kwiatkowska, D. Parker, Automated learning of probabilistic assumptions for compositional reasoning, in: FASE, Springer, 2011, pp. 2–17.
[32] J. Filar, K. Vrieze, Competitive Markov Decision Processes, Springer Science & Business Media, 2012.
[33] S. Haddad, B. Monmege, Interval iteration algorithm for mdps and imdps, Theor. Comput. Sci. 735 (2018) 111–131, https://doi.org/10.1016/j.tcs.2016.12.003.
[34] E.M. Hahn, A. Hartmanns, C. Hensel, M. Klauck, J. Klein, J. Kretínský, D. Parker, T. Quatmann, E. Ruijters, M. Steinmetz, The 2019 comparison of tools for the analysis of quantitative formal models - (QCOMP 2019 competition report), in: TACAS (3), Springer, 2019, pp. 69–92.
[35] A.J. Hoffman, R.M. Karp, On nonterminating stochastic games, Manag. Sci. 12 (1966) 359–370, https://doi.org/10.1287/mnsc.12.5.359.
[36] A. Hordijk, L. Kallenberg, Linear programming and Markov decision chains, Manag. Sci. 25 (1979) 352–362, https://doi.org/10.1287/mnsc.25.4.352.
[37] A. Itai, M. Rodeh, Symmetry breaking in distributed networks, Inf. Comput. 88 (1990) 60–87, https://doi.org/10.1016/0890-5401(90)90004-2.
[38] M. Kattenbelt, M.Z. Kwiatkowska, G. Norman, D. Parker, A game-based abstraction-refinement framework for Markov decision processes, Form. Methods Syst. Des. 36 (2010) 246–280, https://doi.org/10.1007/BF01415989.
[39] E. Kelmendi, J. Krämer, J. Kretínský, M. Weininger, Value iteration for simple stochastic games: stopping criterion and learning algorithm, in: CAV (1), Springer, 2018, pp. 623–642.
[40] J. Kretínský, T. Meggendorfer, Efficient strategy iteration for mean payoff in Markov decision processes, in: ATVA, Springer, 2017, pp. 380–399.
[41] J. Kretínský, T. Meggendorfer, Of cores: a partial-exploration framework for Markov decision processes, Log. Methods Comput. Sci. 16 (2020), https://lmcs.episciences.org/6833.
[42] J. Kretínský, E. Ramneantu, A. Slivinskiy, M. Weininger, Comparison of algorithms for simple stochastic games, in: GandALF, 2020, pp. 131–148.
[43] M. Kwiatkowska, G. Norman, D. Parker, G. Santos, Prism-games 3.0: stochastic game verification with concurrency, equilibria and time, in: CAV (2), Springer, 2020, pp. 475–487.
[44] M.Z. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: verification of probabilistic real-time systems, in: CAV, Springer, 2011, pp. 585–591.
[45] M.Z. Kwiatkowska, G. Norman, D. Parker, The PRISM benchmark suite, in: QEST, IEEE Computer Society, 2012, pp. 203–204.
[46] M.Z. Kwiatkowska, G. Norman, D. Parker, J. Sproston, Performance analysis of probabilistic timed automata using digital clocks, Form. Methods Syst. Des. 29 (2006) 33–78, https://doi.org/10.1007/s10703-006-0005-2.
[47] M.Z. Kwiatkowska, G. Norman, J. Sproston, Probabilistic model checking of the IEEE 802.11 wireless local area network protocol, in: PAPM-PROBMIV, Springer, 2002, pp. 169–187.
[48] M.Z. Kwiatkowska, G. Norman, J. Sproston, Probabilistic model checking of deadline properties in the IEEE 1394 firewire root contention protocol, Form. Asp. Comput. 14 (2003) 295–318, https://doi.org/10.1007/s001650300007.
[49] S.M. LaValle, Robot motion planning: a game-theoretic foundation, Algorithmica 26 (2000) 430–465, https://doi.org/10.1007/s004539910020.
[50] J. Li, W. Liu, A novel heuristic Q-learning algorithm for solving stochastic games, in: IJCNN, 2008, pp. 1135–1144.
[51] H.B. McMahan, M. Likhachev, G.J. Gordon, Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees, in: ICML, ACM, 2005, pp. 569–576.
[52] K. Phalakarn, T. Takisaka, T. Haas, I. Hasuo, Widest paths and global propagation in bounded value iteration for stochastic games, in: CAV (2), Springer, 2020, pp. 349–371.
[53] M.L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, Wiley Series in Probability and Statistics, Wiley, 1994.
[54] F. Saffre, A. Simaitis, Host selection through collective decision, ACM Trans. Auton. Adapt. Syst. 7 (2012) 4:1–4:16, https://doi.org/10.1145/2168260.2168264.
[55] A.L. Strehl, L. Li, E. Wiewiora, J. Langford, M.L. Littman, PAC model-free reinforcement learning, in: ICML, ACM, 2006, pp. 881–888.
[56] M. Svorenová, M. Kwiatkowska, Quantitative verification and strategy synthesis for stochastic games, Eur. J. Control 30 (2016) 15–30, https://doi.org/10.1016/j.ejcon.2016.04.009.
[57] A. Tcheukam, H. Tembine, One swarm per queen: a particle swarm learning for stochastic games, in: SASO, 2016, pp. 144–145.
[58] M. Ujma, On verification and controller synthesis for probabilistic systems at runtime, Ph.D. thesis, University of Oxford, UK, 2015, http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.711811.
[59] L.G. Valiant, A theory of the learnable, Commun. ACM 27 (1984) 1134–1142, https://doi.org/10.1145/1968.1972.
[60] O. Vrieze, S. Tijs, T.E. Raghavan, J. Filar, A finite algorithm for the switching control stochastic game, OR Spektrum 5 (1983) 15–24, https://doi.org/10.1007/BF01720283.
[61] M. Wen, U. Topcu, Probably approximately correct learning in stochastic games with temporal logic specifications, in: IJCAI, IJCAI/AAAI Press, 2016, pp. 3630–3636, http://www.ijcai.org/Abstract/16/511.

B Comparison of Algorithms for Simple Stochastic Games (InC 2022)

This is an exact reprinting of the paper which has been published as a **peer reviewed journal paper**.

Jan Kretinsky, Emanuel Ramneantu, Alexander Slivinskiy, and Maximilian Weininger. “Comparison of Algorithms for Simple Stochastic Games”. In: Information and Computation. Elsevier, 2022. doi: 10.1016/j.ic.2022.104885

Summary

Simple stochastic games are turn-based $2\frac{1}{2}$-player zero-sum graph games with a reachability objective. The problem is to compute the winning probability as well as the optimal strategies of both players. In this paper, we compare the three known classes of algorithms – value iteration, strategy iteration and quadratic programming – both theoretically and practically. Further, we suggest several improvements for all algorithms, including the first approach based on quadratic programming that avoids transforming the stochastic game to a stopping one. Our extensive experiments show that these improvements can lead to significant speed-ups. We implemented all algorithms in PRISM-games 3.0, thereby providing the first implementation of quadratic programming for solving simple stochastic games.

For more details on this publication, we refer to Chapter 3 of the main body.

Contribution

Contribution of Maximilian Weininger	
Development and conceptual design of the research project	90%
Discussion and development of ideas	50%
Composition and revision of the manuscript	80%
Proofs	60%
Implementation	15%
Experimental evaluation	15%

Information and Computation ••• (••••) ••••••

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Comparison of algorithms for simple stochastic games [☆]

Jan Křetínský, Emanuel Ramneantu, Alexander Slivinskiy, Maximilian Weininger [*]

Technical University of Munich, Germany

A R T I C L E   I N F O

Article history:
Received 4 November 2020
Received in revised form 16 September 2021
Accepted 23 February 2022
Available online xxxx

Keywords:
Formal methods
Probabilistic verification
Stochastic games
Algorithms
Value iteration
Strategy iteration
Quadratic programming

A B S T R A C T

Simple stochastic games are turn-based 2½-player zero-sum graph games with a reachability objective. The problem is to compute the winning probabilities as well as the optimal strategies of both players. In this paper, we compare the three known classes of algorithms – value iteration, strategy iteration and quadratic programming – both theoretically and practically. Further, we suggest several improvements for all algorithms, including the first approach based on quadratic programming that avoids transforming the stochastic game to a stopping one. Our extensive experiments show that these improvements can lead to significant speed-ups. We implemented all algorithms in PRISM-games 3.0, thereby providing the first implementation of quadratic programming for solving simple stochastic games.

1. Introduction

Simple stochastic games (SGs), e.g. [18], are zero-sum games played on a graph by players Maximizer and Minimizer, who choose actions in their respective vertices (also called states). Each action is associated with a probability distribution determining the next state to move to. The objective of Maximizer is to maximize the probability of reaching a given target state; the objective of Minimizer is the opposite.

The basic decision problem is to determine whether Maximizer can ensure a reachability probability above a certain threshold if both players play optimally. This problem is among the rare and intriguing combinatorial problems that are in **NP** $\cap$ **co-NP** [19], but whether it belongs to **P** is a major and long-standing open problem. Further, several other important problems can be reduced to SG, for instance parity games, mean-payoff games, discounted-payoff games and their stochastic extensions [8].

Besides the theoretical interest, SGs are a standard model in control and verification of stochastic reactive systems [23, 11]; see e.g. [44] for an overview over various recent case studies. Further, since Markov decision processes (MDP) [40] are

[☆] This research was funded in part by the German Research Foundation (DFG) projects 383882557 *Statistical Unbounded Verification (SUV)* and 427755713 *Group-By Objectives in Probabilistic Verification (GOPro)*. It is the extended version of the paper of the same name that appeared at GandALF 20 [37]. On the theoretical side, we added the full technical proofs and more description of quadratic programming. For the experiments, we extended the description of the experimental setup, included the complete tables for the runtime comparison and added an analysis of memory consumption. Moreover, we investigate the behaviour of quadratic programming on models with many actions per state.

[*] Corresponding author.
E-mail address: maximilian.weininger@tum.de (M. Weininger).

https://doi.org/10.1016/j.ic.2022.104885

a special case with only one player, SGs can serve as abstractions of large MDPs [31] or provide robust versions of MDPs when precise transition probabilities are not known [14,45].

There are three classes of algorithms for computing the reachability probability in simple stochastic games, as surveyed in [19]: value iteration (VI), strategy iteration (SI, also known as policy iteration) and quadratic programming (QP). In [19], they all required the SG to be transformed into a certain normal form, among other properties ensuring that the game is stopping. This not only blows up the size of the SG, but also changes the reachability probability; however, it is possible to infer the original probability. For VI and SI, this requirement has since been lifted, e.g. [10,7], but not for QP.

While searching for a polynomial algorithm, there were several papers on VI and SI; however, the theoretical improvements so far are limited to subexponential variants [38,21] and variants that are fast for SG with few random vertices [24,30], i.e. games where most actions yield a successor deterministically. QP was not looked at, as it is considered common knowledge that it performs badly in practice.

There exist several tools for solving games: GAVS+ [17] offers VI and SI for SGs, among other things. However, it is more for educational purposes than large case studies and currently not maintained. GIST [12] performs qualitative analysis of stochastic games with ω-regular objectives. For MDPs (games with a single player), we refer to [26] for an overview of existing tools. Most importantly, PRISM-games 3.0 [36] is a recent tool that offers algorithms for several classes of games. However, for SGs with a reachability objective, it offers only variants of value iteration, none of which give a guarantee, so the result might be arbitrarily far off. Thus, currently no tool offers a precise solution method for large simple stochastic games.

Our contribution is the following:

- We provide an extension of the quadratic programming approach that does not require the transformation of the SG into a stopping SG in normal form.
- We propose several optimizations for the algorithms, inspired by [34], including an extension of topological value iteration from MDPs [22,4].
- We implement VI with guarantees on precision as well as SI, QP and our optimizations in PRISM-games 3.0—thereby providing the first implementations of QP for SGs—and experimentally compare them on both the realistic case studies of PRISM-games and on interesting handcrafted corner cases.

Related work

We sketch the recent developments of each class of algorithms and then several further directions.

Value iteration is a standard solution method also for MDPs [40]. For a long time, the stopping criterion of VI was such that it could return arbitrarily imprecise results [25]. In principle, the computation has to run for an exponential number of steps in order to be able to give a precise answer [10]. However, recently several heuristics that give guarantees and are typically fast were proposed for MDPs [6,25,41,28], as well as for SGs [32,39]. A learning-based variant of VI for MDPs [6] was also extended to SGs [32]. The variant of VI from [30] requires the game to be in normal form, blowing up its size, and is good only if there are few random vertices, as it depends on the factorial of this number; it is impractical for almost all case studies considered in this paper.

Strategy iteration was introduced in [29]. A randomized version was given in [19], and subexponential versions in [38,21]. Another variant of SI was proposed in [43], however there is no evidence that it runs in polynomial time. The idea of considering games with few random vertices (as in the already mentioned works of [21,30]) was first introduced in [24], where they exhaustively search a subspace of strategies, which is called f-strategies.

To the best of our knowledge, quadratic programming as solution method for simple stochastic games was not investigated further after the first mention in [19]. However, convex QPs are solvable in polynomial time [33]. If it was possible to encode the problem in a convex QP of polynomial size, this would result in a polynomial algorithm. The encoding we provide in this paper can however be exponential in the size of the game.

Further related works consider other variants of the model, for example concurrent stochastic games, see e.g. [27] for the complexity of SI and VI and [9] for strategy complexity, or games with limited information [2]. Furthermore, one can consider other objectives, e.g. ω-regular objectives [11], total reward [4], mean payoff [46] or combinations of objectives [16, 1].

2. Preliminaries

We now introduce the model of stochastic games, define the semantics by the standard means of infinite paths and strategies and then define the important concept of end components, which are subgraphs of stochastic games that are problematic for all three classes of algorithms.

A *probability distribution* on a finite set X is a mapping $\delta : X \to [0,1]$, such that $\sum_{x\in X} \delta(x) = 1$. The set of all probability distributions on X is denoted by $\mathcal{D}(X)$.

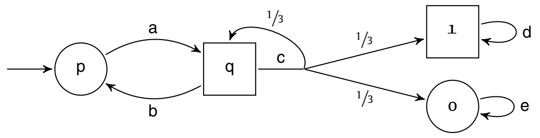


Fig. 1. An example of an SG with $S = \{p, q, 1, 0\}$, $S_\square = \{q, 1\}$, $S_\bigcirc = \{p, 0\}$, the initial state p and the set of actions $A = \{a, b, c, d, e\}$; $Av(p) = \{a\}$ with $\delta(p, a)(q) = 1$; $Av(q) = \{b, c\}$ with $\delta(q, b)(p) = 1$ and $\delta(q, c)(q) = \delta(q, c)(1) = \delta(q, c)(0) = \frac{1}{3}$. For actions with only one successor, we do not depict the transition probability 1.

2.1. Stochastic games

Now we define stochastic games, in literature often referred to as simple stochastic games or turn-based stochastic two-player games with a reachability objective. As opposed to the notation of e.g. [18], we do not have special stochastic nodes, but rather a probabilistic transition function. See Fig. 1 for an example of an SG.

Definition 1 *(SG). A stochastic game (SG)* is a tuple $(S, S_\square, S_\bigcirc, s_0, A, Av, \delta, F)$ where

- S is a finite set of *states* partitioned[1] into the sets $S_\square$ and $S_\bigcirc$ of states of the player *Maximizer* and *Minimizer*[2] respectively
- $s_0 \in S$ is the *initial* state
- A is a finite set of *actions*
- $Av : S \to 2^A$ assigns to every state a set of *available* actions
- $\delta : S \times A \to \mathcal{D}(S)$ is a *transition function* that given a state s and an action $a \in Av(s)$ yields a probability distribution over *successor* states. We slightly abuse notation and write $\delta(s, a, s')$ instead of $\delta(s, a)(s')$.
- $F \subseteq S$ is a set of target states. Without loss of generality we can assume that every target state only has one action that is a self-loop with probability 1, because we consider the reachability objective.

A *Markov decision process (MDP)* is a special case of SG where $S_\bigcirc = \emptyset$, and a Markov chain (MC) is a special case of an MDP, where for all $s \in S : |Av(s)| = 1$.

Without loss of generality we assume that SGs are non-blocking, so for all states s we have $Av(s) \neq \emptyset$. For a state s and an available action $a \in Av(s)$, we denote the set of successors by $Post(s, a) := \{s' \mid \delta(s, a, s') > 0\}$. Finally, for any set of states $T \subseteq S$, we use $T_\square$ and $T_\bigcirc$ to denote the states in T that belong to Maximizer and Minimizer, whose states are drawn in the figures as squares $\square$ and circles $\bigcirc$, respectively.

2.2. Semantics: paths, strategies and values

The semantics of SGs is given in the usual way by means of strategies, the induced Markov chain and the respective probability space, as follows: An *infinite path* ρ is an infinite sequence $\rho = s_0 a_0 s_1 a_1 \dots \in (S \times A)^\omega$, such that for every $i \in \mathbb{N}$ we have $a_i \in Av(s_i)$ and $s_{i+1} \in Post(s_i, a_i)$. *Finite paths* are defined analogously as elements of $(S \times A)^* \times S$.

As this paper deals with the probabilistic reachability objective, we can restrict our attention to memoryless deterministic strategies, which are optimal for this objective [18]. A *strategy* of Maximizer, respectively Minimizer, is a function $\sigma : S_\square \to A$, respectively $S_\bigcirc \to A$, such that $\sigma(s) \in Av(s)$ for all s. A pair (σ, τ) of memoryless deterministic strategies of Maximizer and Minimizer induces a Markov chain $G^{\sigma,\tau}$, as for every state the strategies select a single action. The Markov chain induces a unique probability distribution $\mathbb{P}_s^{\sigma,\tau}$ over measurable sets of infinite paths [3, Ch. 10].

We write $\Diamond F := \{\rho \mid \rho = s_0 a_0 s_1 a_1 \dots \in (S \times A)^\omega \wedge \exists i \in \mathbb{N}.\ s_i \in F\}$ to denote the (measurable) set of all paths which eventually reach F. For each $s \in S$, we define the *value* in s as

$$V(s) := \sup_\sigma \inf_\tau \mathbb{P}_s^{\sigma,\tau}(\Diamond F) = \inf_\tau \sup_\sigma \mathbb{P}_s^{\sigma,\tau}(\Diamond F),$$

where the equality follows from [18]. Note that SGs with a reachability objective are *determined*, i.e. knowing the strategy of the opponent does not increase the chances of winning, provided the opponent plays optimally. Moreover, since there are only finitely many memoryless deterministic strategies, one could replace sup and inf with max and min.

[1] I.e., $S_\square \subseteq S$, $S_\bigcirc \subseteq S$, $S_\square \cup S_\bigcirc = S$, and $S_\square \cap S_\bigcirc = \emptyset$.

[2] The names are chosen, because Maximizer maximizes the probability of reaching the given target states, and Minimizer minimizes it.

The value is the least fixpoint of the so called *Bellman equations* [19]:

$$\mathsf{V}(\mathsf{s}) = \begin{cases} 1 & \text{if } \mathsf{s} \in F \\ \max_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{V}(\mathsf{s},\mathsf{a}) & \text{if } \mathsf{s} \in S_{\Box} \setminus F \\ \min_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{V}(\mathsf{s},\mathsf{a}) & \text{if } \mathsf{s} \in S_{\bigcirc} \setminus F, \end{cases} \tag{1}$$

$$\text{with } \mathsf{V}(\mathsf{s},\mathsf{a}) := \sum_{s' \in S} \delta(\mathsf{s},\mathsf{a},\mathsf{s}') \cdot \mathsf{V}(\mathsf{s}') \tag{2}$$

The states with no path to the target, so called sinks, are of special interest, as they have a value of 0. Moreover, they can be computed a priori by standard graph analysis. For example, the sinks are all states that are not visited by a backwards depth-first search beginning at the target states F. We denote the set of sinks as Z.

We are interested not only in the values $\mathsf{V}(\mathsf{s})$ for all $\mathsf{s} \in S$, but also their ε-approximation, i.e. an approximation $\mathsf{L} : S \to \mathbb{Q}$ with $|\mathsf{V}(\mathsf{s}) - \mathsf{L}(\mathsf{s})| < \varepsilon$. Moreover, we want to provide the corresponding (ε-)optimal strategies for both players. A strategy σ of Maximizer is (ε-)optimal, if for every Minimizer strategy τ we have $\mathbb{P}_{\mathsf{s}}^{\sigma,\tau}(\Diamond F) \geq \mathsf{V}(\mathsf{s})$ (respectively $\geq \mathsf{V}(\mathsf{s}) - \varepsilon$) and an optimal Minimizer strategy is defined analogously. Note that it suffices to have either the values or the optimal strategies, because from one we can infer the other as follows: given a pair of optimal strategies (σ, τ), the values can be computed by solving the induced Markov chain $\mathsf{G}^{\sigma,\tau}$. Given a vector of values for all states, an optimal pair of strategies can be computed by randomizing over all locally optimal actions in each state (e.g. $\sigma(\mathsf{s})$ randomizes over the set $\arg\max_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{V}(\mathsf{s},\mathsf{a})$ for Maximizer states, and dually with $\arg\min$ for Minimizer). To get a deterministic strategy, we cannot only pick some locally optimal action, but additionally we have to ensure that Maximizer is not stuck in some cycle when playing the actions. One way of doing this is by looking at end components, which are the topic of the next subsection.

2.3. End components

When computing the values of states in an SG, we need to take special care of *end components* (EC). Intuitively, an EC is a subset of states of the SG, where the game can remain forever; i.e. given certain strategies of both players, there is no positive probability to exit the EC to some other state. ECs correspond to bottom strongly connected components of the Markov chains induced by some pair of strategies.

Definition 2 *(EC).* A non-empty set $T \subseteq S$ of states is an *end component (EC)* if there exists a non-empty set $B \subseteq \bigcup_{\mathsf{s} \in T} \mathsf{Av}(s)$ of actions[3] such that

1. for each $\mathsf{s} \in T, \mathsf{a} \in B \cap \mathsf{Av}(\mathsf{s})$ we have $\mathsf{Post}(\mathsf{s},\mathsf{a}) \subseteq T$,
2. for each $\mathsf{s}, \mathsf{s}' \in T$ there is a finite path $\mathsf{w} = \mathsf{s}\mathsf{a}_0 \ldots \mathsf{a}_n \mathsf{s}' \in (T \times B)^* \times T$, i.e. the path stays inside T and only uses actions in B.

An end component T is a *maximal end component (MEC)* if there is no other end component T' such that $T \subseteq T'$.

Given an SG G, the set of its MECs is denoted by $\mathsf{MEC}(\mathsf{G})$ and can be computed in polynomial time [20]. ECs are of special interest, because in ECs there can be multiple fixpoints of the Bellman equations (see Equation (1) and (2)).

Example 1. Consider the SG of Fig. 1. The set of states $T = \{\mathsf{p}, \mathsf{q}\}$ is an EC, as when playing only actions from $B = \{\mathsf{a}, \mathsf{b}\}$ the play remains in T forever. It is even a MEC, as there is no superset of T with this property.

We now show that setting the value of both states in T to any x with $\frac{1}{2} \leq x \leq 1$ is a fixpoint of the Bellman equations. Note that since $\mathsf{1} \in F$, $\mathsf{V}(\mathsf{1}) = 1$, and since $\mathsf{o} \in Z$, $\mathsf{V}(\mathsf{o}) = 0$. State p only has action a that goes with probability 1 to q. So, since $\mathsf{V}(\mathsf{q}) = x$, we also have $\mathsf{V}(\mathsf{p}) = x$. Similarly, $\mathsf{V}(\mathsf{q},\mathsf{b}) = x$. Finally, observe that $\mathsf{V}(\mathsf{q},\mathsf{c}) = \frac{1}{3} + \frac{1}{3} \cdot x$, which is equal to x for $x = \frac{1}{2}$ and less than x for all $x > \frac{1}{2}$. Thus, for every $x \geq \frac{1}{2}$, it will be optimal for the Maximizer state q to use action b and we have $\mathsf{V}(\mathsf{q}) = x$.

So we see that there are infinitely many fixpoints which are greater than the value, which is the least fixpoint of the Bellman equations, i.e. $\mathsf{V}(\mathsf{p}) = \mathsf{V}(\mathsf{q}) = \frac{1}{2}$. △

As there can be multiple greater fixpoints of the Bellman equations, methods relying on iterative over-approximation of the value may not converge, as they can be stuck at some greater fixpoint than the value (cf. [32, Lemma 1] and Example 2 in the next section). This is why the original description of the algorithms [19] considered only stopping games. Stopping

[3] Note that this assumes that action names are unique. This can always be achieved by renaming actions, e.g. prepending every action with the state it is played from.

games are SGs where the set $F \cup Z$ is reached with probability 1, or equivalently games without ECs in the set $S \setminus (F \cup Z)$. The algorithms are theoretically applicable to arbitrary SGs, as for every non-stopping SG one can construct a stopping SG and infer the original value from solving the stopping SG. See Section 3.3.2 for a description of this approach and Section 4.1.4 for a discussion of the practical drawbacks.

3. State of the art algorithms

In this section, we describe the existing algorithms for solving simple stochastic games, namely value iteration, strategy iteration and quadratic programming (Section 3.1, 3.2 and 3.3 respectively). The input for all of them is an SG $\mathsf{G} = (S, S_\square, S_\bigcirc, \mathsf{s}_0, \mathsf{A}, \mathsf{Av}, \delta, F)$. Value iteration additionally needs a precision ε, given as a rational number. After termination, all of them return a vector of values for each state (ε-precise for BVI) and the corresponding (ε-)optimal strategies. Quadratic programming in its current form only works on stopping SGs in a certain normal form.

3.1. Bounded value iteration

Value Iteration (VI), see e.g. [40], is the most common algorithm for solving MDPs and SGs, and the only method implemented in PRISM-games [36]. Originally, VI only computed a convergent sequence of under-approximations; however, as it was unclear how to stop, results returned by model checkers could be off by arbitrary amounts [25]. Thus, it was extended to also compute a convergent over-approximation [32]. The resulting algorithm is called *bounded value iteration (BVI).*

The basic idea of BVI is to start from a vector L_0 respectively U_0 that definitely is an under-/over-approximation of the value, i.e. for every state $\mathsf{L}_0(\mathsf{s}) \leq \mathsf{V}(\mathsf{s}) \leq \mathsf{U}_0(\mathsf{s})$. Then the algorithm repeatedly applies so called *Bellman updates*, i.e. it uses a version of Equation (1) as follows (the equation for the over-approximation is obtained by replacing L with U):

$$\mathsf{L}_n(\mathsf{s}) = \begin{cases} \max_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{L}_{n-1}(\mathsf{s}, \mathsf{a}) & \text{if } \mathsf{s} \in S_\square \\ \min_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{L}_{n-1}(\mathsf{s}, \mathsf{a}) & \text{if } \mathsf{s} \in S_\bigcirc \end{cases}, \tag{3}$$

where $\mathsf{L}_{n-1}(\mathsf{s}, \mathsf{a})$ is computed from $\mathsf{L}_{n-1}(\mathsf{s})$ as in Equation (2). Since V is the least fixpoint of the Bellman equations, $\lim_{n\to\infty} \mathsf{L}_n$ converges to V. However, the over-approximation U need not converge in the presence of ECs.

Example 2. Consider the SG of Fig. 1 with the EC $\{\mathsf{p}, \mathsf{q}\}$. Let $\mathsf{U}_0 = 1$ for p, q and $\mathsf{1}$ and $\mathsf{U}_0 = 0$ for $\mathsf{0}$. Then we have $\mathsf{U}_0(\mathsf{q}, \mathsf{b}) = 1$ and $\mathsf{U}_0(\mathsf{q}, \mathsf{c}) = 2/3$. Thus, q will pick action b, as it promises a higher value, and the over-approximation does not change. This happens, because looking at the current upper bound, Maximizer is under the impression that staying in the EC yields a higher value. However, it actually reduces the probability to reach the target to 0. So the algorithm has to perform an additional step to inform states in an EC that they should not depend on each other, but on the best exit. In this case, $\mathsf{U}_1(\mathsf{q}) = \mathsf{U}_0(\mathsf{q}, \mathsf{c}) = 2/3$. △

In fact, it does not suffice to only decrease the upper bound of ECs, but a more in-depth graph analysis is required to detect the problematic subsets of states, so called *simple end components* (SEC) [32, Definition 5]. These SECs cannot be found a priori, because they depend on the values of their exits. Thus, candidate SECs are guessed according to the current lower bound. When the lower bound eventually converges to the value, the true SECs are found. The over-approximation of states in the SECs has to be decreased ("deflated" in the words of [32]) to the over-approximation of the best exit from the SEC. Formally, the best exit is defined as

$$\mathsf{best\_exit}_{\mathsf{U}}(T) = \max_{\substack{\mathsf{s} \in T_\square \\ \neg \mathsf{Post}(\mathsf{s},\mathsf{a}) \subseteq T}} \mathsf{U}(\mathsf{s}, \mathsf{a}),$$

where U is the over-approximation in the current iteration. Decreasing the approximation for the SEC candidates to the best exit always results in a correct over-approximation [32, Lemma 3] and suffices to converge to the true value in the limit [32, Theorem 2]. Algorithm 1 shows the full BVI algorithm from [32].

There also is a simulation based asynchronous version of BVI (see [32, Section 4.4]) which can perform very well on models with a certain structure [35]. It updates states encountered by simulations, and guides those simulations to explore only the relevant part of the state space. If only few states are relevant for convergence, this algorithm is fast; if large parts of the state space are relevant or there are many cycles in the game graph that slow the simulations down, this adaption of BVI performs badly.

3.2. Strategy iteration

In contrast to value iteration, the approach of *strategy iteration (SI)* [29] does not compute a sequence of value-vectors, but instead a sequence of strategies. Starting from an arbitrary strategy of Maximizer, we repeatedly compute the best response of Minimizer and then greedily improve Maximizer's strategy. The resulting sequence of Maximizer strategies is monotonic and converges to the optimal strategy [7, Theorem 3]. The pseudocode for strategy iteration is given in Algorithm 2.

Algorithm 1 Bounded value iteration algorithm from [32].

```
1: procedure BVI(precision $\varepsilon > 0$)
2:     for $s \in S$ do      # Initialization
3:         $L(s) = 0$      # Lower bound
4:         $U(s) = 1$      # Upper bound
5:     for $s \in F$ do $L(s) = 1$      # Value of targets is determined a priori
6:     repeat
7:         $L, U$ get updated according to Eq. (3)      # Bellman updates
8:         for every SEC candidate $T$ do
9:             for $s \in T$ do
10:                $U(s) \leftarrow \mathsf{best\_exit}_U(T)$      # Decrease U to best exit
11:    until $U(s) - L(s) < \varepsilon$ for all $s \in S$      # Guaranteed error bound
```

Algorithm 2 Strategy iteration.

```
1: procedure SI
2:     $\sigma' \leftarrow$ arbitrary Maximizer *attractor strategy*      # Proper initial strategy
3:     repeat
4:         $\sigma \leftarrow \sigma'$
5:         for $s \in S$ do
6:             $L(s) \leftarrow \inf_\tau \mathbb{P}_s^{\sigma,\tau}(\Diamond F)$      # Solve MDP for opponent
7:         for $s \in S_\Box$ do
8:             if $\sigma(s) \in \arg\max_{a \in Av(s)} L(s,a)$ then
9:                 $\sigma'(s) \leftarrow \sigma(s)$ # Only change $\sigma$ on strict improvement
10:            else
11:                $\sigma'(s) \leftarrow$ any element of $\arg\max_{a \in Av(s)} L(s,a)$
12:    until $\sigma = \sigma'$
```

Note that in non-stopping SGs (games with ECs) the initial Maximizer strategy cannot be completely arbitrary, but it has to be *proper*, i.e. ensure that either a target or a sink state is reached almost surely; it must not stay in some EC, as otherwise the algorithm might not converge to the optimum due to problems similar to those described in Example 2. In Algorithm 2, we use the construction of the *attractor strategy* [7, Section 5.3] to ensure that our initial guess is a proper strategy (Line 2). It first analyses the game graph to find the sink states Z. Then, it performs a backwards breadth first search, starting from the set of both target and sink states. A state discovered in the i-th iteration of the search has to choose some action that reaches a state discovered in the (i-1)-th iteration with positive probability. Such a state exists by construction, and the choice ensures that the initial strategy reaches the set of target or sink states almost surely.

When a Maximizer strategy is fixed, the main loop of Algorithm 2 solves the induced MDP G^σ (Line 6). We discuss the different ways to do this in Section 4.2. The algorithm need not remember the Minimizer strategy, but only uses the computed value estimates L to greedily update Maximizer's strategy (Line 11); note that here $L(s,a)$ is again computed from $L(s)$ as in Equation (2). For termination, it is necessary that we only update the strategy if switching strictly improves the value (see Line 8), as otherwise we might cycle infinitely between equivalent strategies. The algorithm stops when the Maximizer strategy does not change any more in one iteration. We can then compute the values and the corresponding Minimizer strategy by solving the induced MDP G^σ.

3.3. Quadratic programming

As mentioned before, the approach of Quadratic programming currently requires that the input SG is in a certain normal form. We first recall how the approach works for a game in that normal form and then describe how to transform an arbitrary SG into normal form.

3.3.1. The quadratic program

Quadratic programming (QP) [19] works by encoding the graph of the SG in a system of constraints. The only global (and local) optimum of the objective function under these constraints is 0, and it is attained if and only if the variable for every state is set to the value of that state [19, Section 3.1]. The proof as well as the construction of the quadratic program relies on the game being in a certain normal form, which consists of four conditions:

- **2Act**: For all $s \in S : |Av(s)| \leq 2$.
- **No1Act**: If $|Av(s)| = 1$, then s is a target or a sink.
- **½Probs**: For all $s, s' \in S, a \in Av(s) : \delta(s,a,s') \in \{0, 0.5, 1\}$.
- **Stopping game**: There are no ECs in G (except for the sinks and targets).

We shortly discuss the advantage of each condition of the normal form: the reason for **2Act** and **No1Act** is that the objective function of the QP requires every (non-sink and non-target) state to have exactly 2 successors. Arguing about a game with

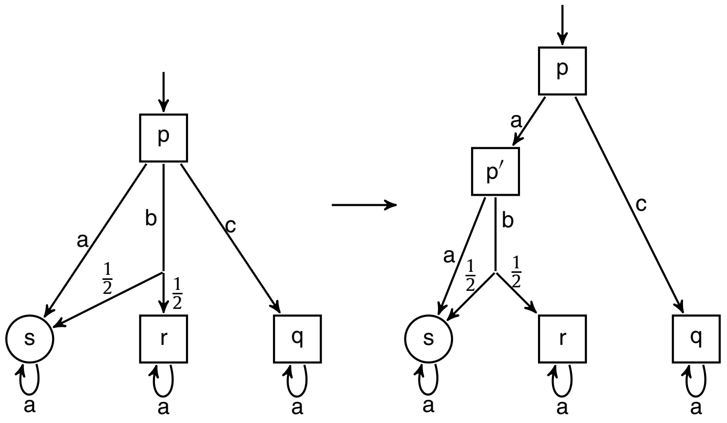


Fig. 2. An example of transforming an SG into one fulfilling the **2Act**-constraint. p has more than two actions, so a binary subtree is built up where every state except for the leaves s, r and q has exactly two actions.

average nodes instead of actions mapping to arbitrary probability distribution simplified the proofs, which is the advantage of **½Probs**. **Stopping game** was necessary, because of the problem of non-convergence in end components, as in Example 2. We describe the procedure from [18] to transfer an arbitrary SG into a polynomially larger SG in normal form in the next Section 3.3.2.

We now state the quadratic program for an SG in normal form as given in [19], but adjusted to our notation. Intuitively, the objective function is 0 if all summands are 0. And the summand for some state s is 0 if its value V(s) is equal to the value of one of its actions V(s, a) or V(s, b). The program is quadratic, since all states (except targets and sinks) have exactly two successors and hence the summand for every state is a quadratic term. The constraints encode the game, ensuring that Maximizer/Minimizer states use the action with the highest/lowest value and fixing the values of targets and sinks. Note the additional definition of V(s, a), which assumes that all occurring non-trivial probabilities are $1/2$.

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{\substack{s\in S\\ \mathsf{Av}(s)=\{a,b\}}} (\mathsf{V}(s)-\mathsf{V}(s,a))(\mathsf{V}(s)-\mathsf{V}(s,b)) & \\ \text{subject to} & \mathsf{V}(s)\geq \mathsf{V}(s,a) & \forall s\in S_{\square}: |\mathsf{Av}(s)|=2, \forall a\in\mathsf{Av}(s) \\ & \mathsf{V}(s)\leq \mathsf{V}(s,a) & \forall s\in S_{\bigcirc}: |\mathsf{Av}(s)|=2, \forall a\in\mathsf{Av}(s) \\ & \mathsf{V}(s)=1 & \forall s\in F \\ & \mathsf{V}(s)=0 & \forall s\in Z \end{array}$$

$$\text{where } \mathsf{V}(s,a)=\left\{\begin{array}{ll} \mathsf{V}(s') & \text{for } \mathsf{Post}(s,a)=\{s'\} \\ 1/2\mathsf{V}(s')+1/2\mathsf{V}(s'') & \text{for } \mathsf{Post}(s,a)=\{s',s''\} \end{array}\right\}$$

3.3.2. Transforming an arbitrary stochastic game into normal form

We describe the constructions of [18] to transform an arbitrary SG into one in normal form that is only polynomially larger. For each of the conditions that are required for normal form, i.e. for **2Act**, **No1Act**, **½Probs** and **Stopping game**, we give the respective transformation.

2Act: In normal form, every state s that is not an absorbing state must have at most two actions. To transform an arbitrary SG into one complying with **2Act**, take every state s with |Av(s) > 2| and construct a binary tree as illustrated in Fig. 2. Two actions of s are taken and given to a new vertex v'. State s has then one action leading to the additional state v' instead of its previous two actions. This can be done iteratively until there is a binary tree where s is the root and has only two actions. Note that every other inner node in this tree also has two actions, and the leaves are exactly $v_1, v_2, \ldots, v_k$. If a state s has $n \geq 2$ actions, then after the transformation there are $n-2$ additional states. Every newly added state belongs to the same player as the original state s.**No1Act:** Every state that is not a target or sink must have more than one action. We can add a second action to every state that is missing one. This action leads to a target in the case of $s \in S_{\bigcirc}$ and otherwise to a sink. If there is no sink in S, we can introduce an artificial sink. In any optimal strategy, neither player would choose the additional action, as the other option is at least as good as the additional one. Therefore, the additional actions do not influence the value of any state.

In Fig. 3, s_0 has an additional action leading to 0. Every state $s \in (S_{\bigcirc} \setminus \{0\})$ should have an action leading to 1, but we omit this to improve the readability of the figure.

½Probs: The normal form requires that transition probabilities are either 0, 0.5 or 1. This implies that every action has either one or two successors. Let $s \in S$ be a state which has an action a that has an arbitrary amount of successors $v_1, v_2, \ldots, v_k$

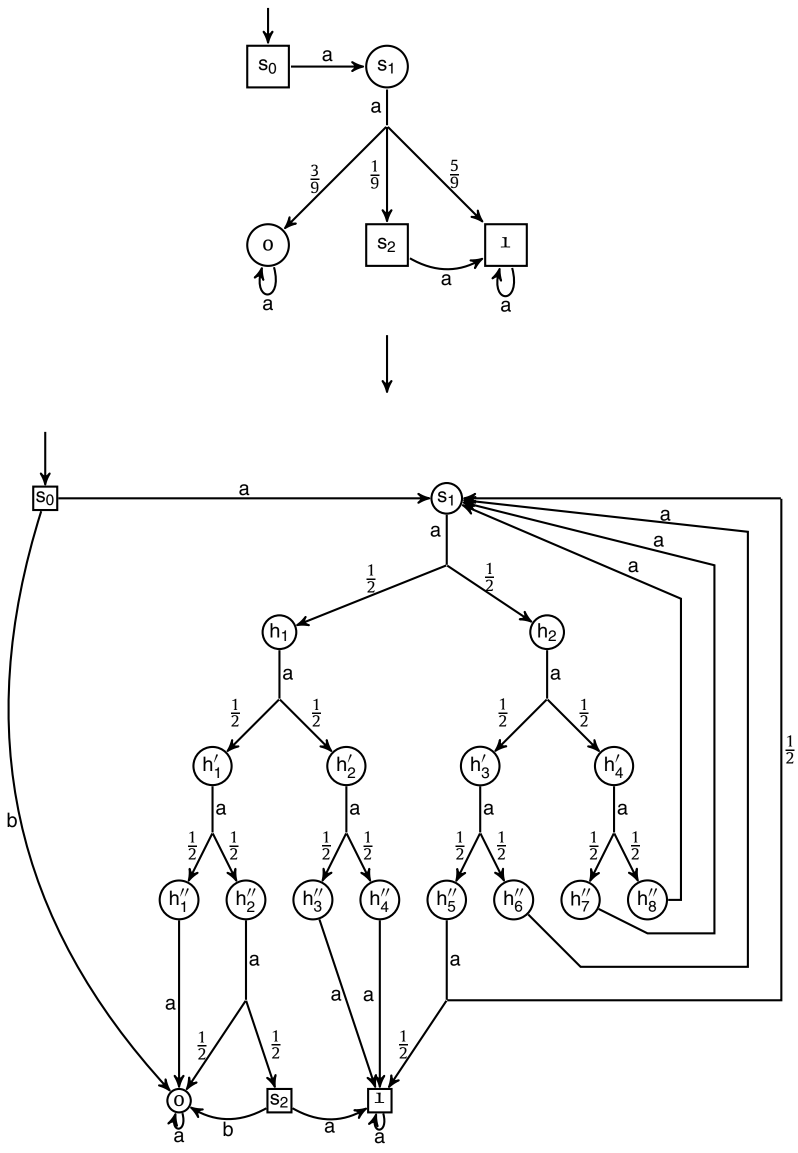


Fig. 3. An example of an arbitrary SG that gets transformed into Condon's normal form. States $h''_1, h''_2, \ldots, h''_8, h'_1, h'_2, h'_3, h'_4$ and h_1, h_2 are introduced to fulfill the **½Probs**-constraint. State s_2 has now a second action leading to sink 0, as otherwise it would not comply with **No1Act**. All the ◯-states with only one action except the sink 0 must have an action leading to 1 but we omit these for the sake of readability.

with rational transitions probabilities $p_i := \delta(\mathsf{s}, \mathsf{a}\ v_i) \in [0,1] \subset \mathbb{Q}$. Consider the greatest common divisor q of all occurring transition probabilities of (s, a). Let q' be the smallest power of 2 such that $q' \geq q$, i.e. $q' = 2^k, k \in \mathbb{N} : 2^{k-1} < q < 2^k = q'$.

Create $1/2 \cdot q'$ new vertices, each with one action and two transitions with probability 0.5. Out of the q' many transitions, $p_i \cdot q$ lead to v_i. If a vertex has two transitions assigned that lead to the same state, the two transitions are unified to one with transition probability 1. The $q' - q$ remaining transitions lead to s. From the new vertices, we build up a binary tree with s as root such that s and every new state have one action each with two transitions that have only transition probabilities of 0, 0.5 or 1, and such that the probability of reaching v_i from s is p_i.

Fig. 3 illustrates an example. s_1 has an action a with transition probabilities $3/9$, $1/9$, $5/9$. The common divisor q of all these fractions is 9. The next power of 2 is 4, so we have $q = 9 \leq 16 = 2^4 = q'$. We need 8 states $h''_1, h''_2, \ldots, h''_8$ that store the 16 transitions. Every transition has a probability of $1/2$.

- The probability to reach 0 from s is $3/9$, therefore 3 transitions have to lead there. We let h''_1 lead to 0 with probability 1 and h''_2 with one of the two transitions.
- The probability to reach s_2 from s is $1/9$, therefore the second transition of h''_2 leads in s_2.
- The probability to reach 1 from s is $5/9$, therefore h''_3, h''_4 and one transition of h''_5 lead to 1.

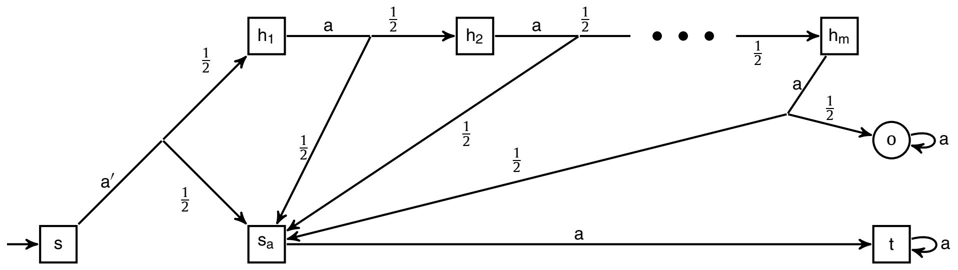


Fig. 4. An example of adding the ε-transitions to a simple SG where initially there were only the initial state s and a target state t and the action a with $\mathsf{Post}(\mathsf{s},\mathsf{a}) = \mathsf{Post}(\mathsf{t},\mathsf{a}) = \mathsf{t}$. The action of $\mathsf{s_a}$ is simulating the outcome of taking action a in s in the original SG while $h_1, h_2, ...h_m$ add the ϵ needed if the initial SG would be non-stopping. For better readability, we omit the superscript s,a for all states h_i.

- The remaining transitions lead back to s.

To connect $h''_1, h''_2, ..., h''_8$ to s a binary tree is constructed with h'_1, h'_2, h'_3, h'_4 and h_1, h_2.
Stopping game: To avoid the possibility of never reaching any absorbing state, we add a transition with a small probability ε leading to a sink-state o to every action. If the players pick strategies that trap the play in a MEC in the original SG, the play would almost surely reach o in the modified SG. If the ε is chosen sufficiently small, one can infer the value in the original SG from the modified SG [18]. This is due to the fact that the value of an SG must be rational, where the denominator is at most $4^{|S|}$, and thus we can round the value in the modified SG to the nearest fraction with denominator $4^{|S|}$.

In Fig. 4, we illustrate the transformation. Although the initial SG is already stopping in this example, we chose this simple game for explanatory purposes. For the ε-transitions that have to be introduced to comply with the **Stopping game** constraint, [19] suggest something smarter instead of constructing a binary tree: for every state-action pair (s,a) add a new state $\mathsf{s_a}$ which has only one action that has the same successors and transition probabilities as (s,a) in the initial game. The state s instead has a new action a' with $1/2$-probabilities of leading either to $\mathsf{s_a}$ or to the state $h_1^{\mathsf{s},\mathsf{a}}$ of a chain of $m \in \mathbb{N}$ (sufficiently large) new states $h_1^{\mathsf{s},\mathsf{a}}, h_2^{\mathsf{s},\mathsf{a}}, ...h_m^{\mathsf{s},\mathsf{a}}$. Each state $h_i^{\mathsf{s},\mathsf{a}}$ with $i \in [m-1]$ has only one action with two $1/2$-probabilities leading to either $h_{i+1}^{\mathsf{s},\mathsf{a}}$ or $\mathsf{s_a}$. The action of $h_m^{\mathsf{s},\mathsf{a}}$ also has probabilities of $1/2$ and leads either to a sink o or to $\mathsf{s_a}$. Thus, playing a' in s now has a $\varepsilon = (1/2)^m$ chance of going to a sink, and with the remaining probability behaves as before.

4. Improvements

In this section, we first generalize QP to be applicable to arbitrary SGs, thereby omitting the costly transformations into the normal form. Then, we identify a hyperparameter of SI, namely that the way in which the MDP is solved after fixing one strategy can be varied. Finally, we provide two optimizations that are applicable to all three algorithms.

4.1. Quadratic programming for general stochastic games

Every transformation into the normal form (see Section 3.3.2) adds additional states or actions to the SG. We want to change the constraints of the QP so that it can deal with arbitrary SGs. This way, we avoid blowing up the SG, which is good since the time for solving the QP depends on the size of the SG.

4.1.1. 2Act

In order to drop the constraint that every state has at most two actions, we can generalize the summand of a state s in the objective function to $\prod_{\mathsf{a}\in\mathsf{Av}(\mathsf{s})}(\mathsf{V}(\mathsf{s}) - \mathsf{V}(\mathsf{s},\mathsf{a}))$. The resulting program is no longer quadratic, as for a state with n actions now the objective function has order n. Thus, in the experiments, we report the verification times of both (i) a higher-order optimization problem for the original SG as well as (ii) a QP for the SG that was transformed to comply with **2Act**.

One more step is needed to ensure that the objective function still is correct: we have to duplicate one term $(\mathsf{V}(\mathsf{s}) - \mathsf{V}(\mathsf{s},\mathsf{a}))$ in Minimizer states with an odd number of actions. This ensures that the objective function cannot become negative. A more detailed explanation is given in the proof of the following lemma.

Lemma 1. *Consider a higher-order optimization problem for an SG* G *which minimizes the objective function* $\sum_{\mathsf{s}\in S}\prod_{\mathsf{a}\in\mathsf{Av}(\mathsf{s})}(\mathsf{V}(\mathsf{s}) - \mathsf{V}(\mathsf{s},\mathsf{a}))$, *where additionally for Minimizer states* s *with an odd number of actions, the term* $(\mathsf{V}(\mathsf{s}) - \mathsf{V}(\mathsf{s},\mathsf{a}))$ *is duplicated for an arbitrary action* $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$. *The constraints are as given*[4] *for the QP in Section 3.3.1, i.e. they encode the game* G.

Then the global optimum of the objective function is 0 and it is attained if and only if for every state $\mathsf{s} \in S$ *and some action* $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$ *we have* $\mathsf{V}(\mathsf{s}) = \mathsf{V}(\mathsf{s},\mathsf{a})$.

[4] Note that the constraint $\mathsf{V}(\mathsf{s}) \sim \mathsf{V}(\mathsf{s},\mathsf{a})$ with $\sim \in \{\leq, \geq\}$ is specified for every action $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$ and now there can be more than two actions.

Proof. The objective function evaluates to 0 if and only if every summand is 0. This is the case if and only if at least one factor of every product is 0, which happens if and only if for every state $\mathsf{s} \in S$ and some action $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$ we have $\mathsf{V}(\mathsf{s}) = \mathsf{V}(\mathsf{s}, \mathsf{a})$. So if the objective function is 0, the solution vector V is a solution to the SG G.

It remains to show that 0 is indeed the global optimum of the higher-order optimization problem. For this, first observe that every summand is non-negative, i.e. greater or equal than 0: For every Maximizer state s, the constraint $\mathsf{V}(\mathsf{s}) \geq \mathsf{V}(\mathsf{s}, \mathsf{a})$ ensures that every factor of the product is non-negative, and thus the whole product is non-negative. For every Minimizer state, dually we know that every factor of the product is non-positive. Moreover, we have ensured that every product has an even number of factors. Thus, the product certainly is non-negative, since either all factors are smaller than 0 and the result is a positive number, or at least one factor is 0 and the result is 0. Finally, the argument that 0 can always be attained is analogous to the one in [19], since the number of actions per state does not affect the proof. □

4.1.2. *No1Act*

The **No1Act**-constraint compels every state $\mathsf{s} \in S$ that is not a target or a sink to have more than one action. The transformation for complying with this constraint adds a second action to every state with only one action; however, the newly added action is chosen in such a way that it does not influence the value of the state, and can thus also be omitted.

Lemma 2. *Let $\mathsf{s} \in S$ be a state in an SG with two actions a and b: a is the action s originally had, and b is the additional action inserted to comply with **No1Act**. Then it holds that $\mathsf{V}(\mathsf{s}) = \mathsf{V}(\mathsf{s}, \mathsf{a})$.*

Proof. We prove this by a case distinction over the player s belongs to. We only provide the proof for the case that $\mathsf{s} \in S_\square$, as the other case is analogous. Let the strategy τ of the Minimizer be arbitrary. Let σ_b be a strategy for the Maximizer in which s takes action b. Per construction, b leads to a sink, and therefore it holds that $\mathsf{V}_{\sigma_b,\tau}(\mathsf{s}) = 0$. Let σ_a be a strategy for the Maximizer in which s takes action a. For every state $\mathsf{s}' \in S$ it holds that $\mathsf{V}(\mathsf{s}') \geq 0$. Thus, $\mathsf{V}_{\sigma_a,\tau}(\mathsf{s}) \geq \mathsf{V}_{\sigma_b,\tau} = 0$. It follows that every optimal Maximizer strategy σ may take action a in s. Since s allows a in every optimal strategy it holds that $\mathsf{V}(\mathsf{s}) = \mathsf{V}(\mathsf{s}, \mathsf{a})$. □

Thus, for a non-absorbing state s with only one action a, we can simplify the program by not including it in the objective function, but only adding a single constraint $\mathsf{V}(\mathsf{s}) = \mathsf{V}(\mathsf{s}, \mathsf{a})$.

4.1.3. *½Probs*

To comply with the **½Probs**-requirement, every transition $\delta(\mathsf{s}, \mathsf{a}, \mathsf{s}')$ with $\mathsf{s}, \mathsf{s}' \in S$ and $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$ must have a probability of 0, 0.5 or 1. We can omit this constraint if we use the general definition of $\mathsf{V}(\mathsf{s}, \mathsf{a})$ as in Equation (2), summing the successors with their respective given transition probability.

Lemma 3. *Let G be an SG that does not comply with **½Probs**. The QP from Section 3.3.1 still is correct, i.e. returns the value vector of G, if we modify it such that $\mathsf{V}(\mathsf{s}, \mathsf{a}) := \sum_{s' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{V}(\mathsf{s}')$.*

Proof. The proof, that the QP computes the value vector of the SG G from [19] does not rely on the fact the probabilities are all 0, 0.5 or 1. This restriction is only due to the different, more restrictive definition of SG. □

4.1.4. *Stopping game*

The final and most complicated constraint of the normal form is that we require the SG to be stopping. The transformation of an arbitrary game into a stopping one adds a transition to a sink with a small probability ε to every action, thus ensuring that a sink (or a target) is reached almost surely. This is problematic not only because the added transitions blow up the quadratic program, but even more so because of the following.

The ε-transitions modify the value of the game. Theoretically, this is no problem, because the value is rational and we know the greatest possible denominator q it can have. Thus, by choosing ε sufficiently small, we ensure that the modified value does not differ by more than $1/q$ and we can obtain the original value by rounding. However, practically, this denominator becomes smaller than machine precision even for small systems, resulting in immense numerical errors. The ε has to be strictly smaller[5] than $(1/4)^{|S|}$ [18]. According to IEEE 754.2019 standard,[6] the commonly used double machine precision is 10^{-16}, so already for 27 states the necessary ε becomes smaller than machine precision. Thus, the transformation to a stopping game is inherently impractical.

Our approach introduces additional constraints for every maximal end component (MEC) to ensure that the QP finds the correct solution.

[5] It actually has to be a lot smaller, since the *value* of every state may differ by at most that amount, but this conservative upper bound suffices to prove our point.

[6] https://standards.ieee.org/content/ieee-standards/en/standard/754-2019.html.

Algorithm 3 Algorithm to add constraints for MECs containing states of both players.

```
1: procedure ADD_MEC_CONSTRAINTS(MEC $T \subseteq S$)
2:     for all strategies $\sigma$ on $T_\square$ do
3:         for all strategies $\tau$ on $T_\bigcirc$ do
4:             for every $(s, a) \in E^T$ do
5:                 Compute $p^{\sigma,\tau}_{e_{(s,a)}}(t)$ for all $t \in T$ by solving the modified induced MC $\mathsf{G}^{\sigma,\tau}$
6:     for every $t \in T$ do
7:         Add constraint: $\mathsf{V}(t) = \max_\sigma \min_\tau \sum_{(s,a) \in E^T} p^{\sigma,\tau}_{e_{(s,a)}}(t) \cdot \mathsf{V}(s, a)$
```

For MECs where all states belong to the same player, the solution is straightforward. All states in MECs with only Minimizer states have a value of 0, as they can choose to remain forever in the EC and not reach the target; they can be identified a priori and are part of the set of sinks Z. All states in MECs with only Maximizer states have the value of the best exit from that MEC [32]. Thus, for all $T \in \mathsf{MEC}(\mathsf{G})$ with $T \cap S_\bigcirc = \emptyset$ we can introduce an additional constraint: $\forall s \in T : \mathsf{V}(s) = \mathsf{best\_exit}_\mathsf{V}(T)$, where best_exit is defined as for BVI (see Section 3.1). Note that max-constraints are expressed through continuous and boolean variables and therefore, the resulting quadratic program is a mixed-integer program.

For MECs containing states of both players, the values of the states depend on the best exit that Maximizer can ensure reaching against the optimal strategy of Minimizer. We cannot just set the value to the best exit of the whole MEC, as Minimizer might prevent some states in the MEC from reaching that best exit. The solution of [32] to analyse the graph and figure out Minimizer's decisions on the fly is not possible, because we have to give the constraints a priori.

We solve the problem as follows: iterate over all strategy-pairs (σ, τ) in the MEC and for each pair describe the corresponding value of every state $\mathsf{V}_{\sigma,\tau}(s)$ depending on the values of the exiting actions $E^T = \{(s, a) \mid s \in T \wedge \mathsf{Post}(s, a) \not\subseteq T\}$. Then constrain the value for all states in the MEC according to the optimal strategies, i.e. $\mathsf{V}(s) = \max_\sigma \min_\tau \mathsf{V}_{\sigma,\tau}(s)$. This ensures that the value of every state is set to the best exit it can reach, because the optimal strategies are chosen.

It remains to define how to describe $\mathsf{V}_{\sigma,\tau}$ depending on the exits E^T. For a pair of strategies (σ, τ) in the MEC, we consider the induced Markov chain $\mathsf{G}^{\sigma,\tau}$. We modify the MC and let every state-action pair $(s, a) \in E^T$ lead to a sink state $e_{(s,a)}$. Then, for every such sink state, we compute the probability $p^{\sigma,\tau}_{e_{(s,a)}}(t)$ to reach it from every state $t \in T$ by solving the MC. Then we set $\mathsf{V}_{\sigma,\tau}(t) = \sum_{(s,a) \in E^T} p^{\sigma,\tau}_{e_{(s,a)}}(t) \cdot \mathsf{V}(s, a)$. We summarize the procedure we just described in Algorithm 3. We use *all strategies on* T to denote every possible mapping that maps every state $t \in T$ to some available action $a \in \mathsf{Av}(t)$.

Note that $\mathsf{V}(s, a)$ is defined as usual (see Equation (2)). As (s, a) is an exit from T, it depends on some successor $s' \notin T$. Thus the states in the MEC do not depend only on each other any more, but they depend on exiting actions. Intuitively, this ensures there is a unique solution.

Lemma 4. *Let* G *be a non-stopping SG. The QP from Section 3.3.1 still is correct, i.e. returns the value vector of* G*, if we modify it as follows: For every MEC* T*, we add the constraint* $\mathsf{V}(s) = \max_\sigma \min_\tau \mathsf{V}_{\sigma,\tau}(s)$ *for all strategies* (σ, τ) *in* T *and all states* $s \in T$.

Proof. We proceed in the following steps:

- Summarize the relevant definitions.
- Prove that $\mathsf{V}_{\sigma,\tau}(t) = \mathbb{P}^{\sigma,\tau}_t(\Diamond F)$ for all $t \in T$.
- Prove that $\mathsf{V}(s) = \max_\sigma \min_\tau \mathsf{V}_{\sigma,\tau}(s)$ for all strategies (σ, τ).
- Prove that adding this constraint ensures the convergence of the QP to a unique correct solution.
- Show why we can restrict to consider only strategies in the MEC T.

1. As before, we consider an SG G. We use T to denote an arbitrary MEC of G, respectively. For strategies, we write σ for Maximizer and τ for Minimizer strategies.
   The proof relies on the following definitions:
   - $E^T = \{(s, a) \mid s \in T \wedge \mathsf{Post}(s, a) \not\subseteq T\}$ is the set of exiting state-action pairs.
   - For every $(s, a) \in E^T$, we introduce a new sink state $e_{(s,a)}$.
   - $p^{\sigma,\tau}_{e_{(s,a)}}(t)$ is the probability to reach such a sink state $e_{(s,a)}$ from state t in the induced Markov chain $\mathsf{G}^{\sigma,\tau}$ modified so that the exiting actions lead to the newly introduced sinks.
   - Note that the symbol V is overloaded in the context of the QP: it can refer both to the value of the SG as well as to the variable of the QP that eventually converges to the value, but that can have other valuations during the computation.
     To be precise, in this proof we distinguish the actual value of the SG – V – and the value of the SG assuming we play (σ, τ) in the MEC – V' – which is what the variables of the QP are set to. However, when adding the constraint to the optimization problem, the distinction between V and V' is not necessary, since the valuation of the variable $\mathsf{V}(s, a)$ of the QP always depends on the current strategies.

- $\mathsf{V}_{\sigma,\tau}(t) = \sum_{(\mathsf{s},\mathsf{a})\in E^T} p^{\sigma,\tau}_{e_{(\mathsf{s},\mathsf{a})}}(t)\cdot \mathsf{V}'(\mathsf{s},\mathsf{a})$ is the value of a state $t \in T$, assuming strategies σ and τ are played.

2. We now prove that $\mathsf{V}_{\sigma,\tau}(t) = \mathbb{P}^{\sigma,\tau}_t(\Diamond F)$ for all $t \in T$. In other words, we prove that the computation we use for $\mathsf{V}_{\sigma,\tau}$ actually captures the concept of probability to reach the target under the strategies (σ,τ). For this, we use the following chain of equations:

$$\begin{aligned}
\mathsf{V}_{\sigma,\tau}(t) &:= \sum_{(\mathsf{s},\mathsf{a})\in E^T} p^{\sigma,\tau}_{e_{(\mathsf{s},\mathsf{a})}}(t)\cdot \mathsf{V}'(\mathsf{s},\mathsf{a}) && \text{(By definition of } \mathsf{V}_{\sigma,\tau}(t)\text{)}\\
&= \sum_{(\mathsf{s},\mathsf{a})\in E^T} p^{\sigma,\tau}_{e_{(\mathsf{s},\mathsf{a})}}(t)\cdot \left(\sum_{\mathsf{s}'\in\mathsf{Post}(\mathsf{s},\mathsf{a})} \delta(\mathsf{s},\mathsf{a},\mathsf{s}')\cdot \mathsf{V}'(\mathsf{s}')\right) && \text{(Unfolding the Bellman equation)}\\
&= \sum_{(\mathsf{s},\mathsf{a})\in E^T} p^{\sigma,\tau}_{e_{(\mathsf{s},\mathsf{a})}}(t)\cdot \left(\sum_{\mathsf{s}'\in\mathsf{Post}(\mathsf{s},\mathsf{a})} \delta(\mathsf{s},\mathsf{a},\mathsf{s}')\cdot \mathbb{P}^{\sigma,\tau}_{\mathsf{s}'}(\Diamond F)\right) && \text{(By definition of } \mathsf{V}'\text{)}\\
&= \mathbb{P}^{\sigma,\tau}_t(\Diamond F)
\end{aligned}$$

In the last step, we pull together the unfolded probability of the path (going to some exit, taking an exiting action and then continuing from the successor); and we use the fact that all paths reaching the target have to pass through some exiting state-action pair, as otherwise they are stuck in the EC forever.
Note that this relies on the assumption that there is no target state in the MEC; this assumption is justified, since we argued in the preliminaries that every target state has only one action which is a self loop, and we exclude these trivial MECs from consideration, because their value is immediately set correctly to 1.

3. It follows from the previous step that for all $t \in T$ we have

$$\mathsf{V}(t) := \sup_\sigma \inf_\tau \mathbb{P}^{\sigma,\tau}_t(\Diamond F) = \sup_\sigma \inf_\tau \mathsf{V}_{\sigma,\tau}(t)$$

We overload the symbol $\mathsf{V}_{\sigma,\tau}$ to also denote the probability for states outside the MEC T to reach the targets under strategies σ and τ, so formally: $\mathsf{V}_{\sigma,\tau}(t) := \begin{cases} \sum_{(\mathsf{s},\mathsf{a})\in E^T} p^{\sigma,\tau}_{e_{(\mathsf{s},\mathsf{a})}}(t)\cdot \mathsf{V}'(\mathsf{s},\mathsf{a}) & \text{if } t \in T \\ \mathbb{P}^{\sigma,\tau}_t(\Diamond F) & \text{otherwise} \end{cases}$.
Then, trivially, we also have that for all state $\mathsf{s} \in S$

$$\mathsf{V}(\mathsf{s}) = \sup_\sigma \inf_\tau \mathsf{V}_{\sigma,\tau}(\mathsf{s})$$

As there are only finitely many memoryless deterministic strategies, and those strategies suffice to attain the optimal value in simple stochastic games, we also have for all state $\mathsf{s} \in S$

$$\mathsf{V}(\mathsf{s}) = \max_\sigma \min_\tau \mathsf{V}_{\sigma,\tau}(\mathsf{s})$$

4. The problem of MECs is that in these state sets there are multiple solutions to the Bellman equations (cf. Example 2), and thus multiple solutions to the quadratic program. By constraining all states in the MECs to $\max_\sigma \min_\tau \mathsf{V}_{\sigma,\tau}(\mathsf{s})$, we constrain them to exactly their value (by the previous step). Thus, we solve the problem, as now the additional solutions are excluded.
5. Note that so far, this proof considered strategies on the whole state space. However, our algorithm only iterates over all strategies *in the MEC*.
This is sufficient, because the states outside the MEC are solved by the rest of the QP, and the newly added constraints depend on those solutions, as they depend on the valuation of $\mathsf{V}(\mathsf{s},\mathsf{a})$ of exiting state-action pairs. □

For a MEC of size n we have to examine at most $(\max_{\mathsf{s}\in T}|\mathsf{Av}(\mathsf{s})|)^n$ pairs of strategies, because it suffices to consider memoryless deterministic strategies. Since the min and max constraints have to be explicitly encoded, we have to add a number of constraints that is exponential in the size of the MEC. Thus, in the worst case, this reduction creates a QP of size exponential in the size of SG, which is worse than the original reduction using the normal form. However, as MECs are typically small, the approach still is more practical.

4.1.5. Summary

In summary, we have shown how to replace every condition of the normal form by modifying the constraints and objective function of the program. The modifications always ensure that the program still computes the correct value, because it still only has a single global optimum in the constrained region, namely if every state variable is set to its value. Thus, we can provide a higher-order program to solve arbitrary SGs, and a quadratic program to solve SGs that only have to satisfy the **2Act** condition.

4.2. Opponent strategy for strategy iteration

We can tune the MDP solution method that is used to compute the opponent strategy in Line 6 of Algorithm 2. We need to ensure that we fix the new choices of Maximizer correctly. For this, we can use the precise MDP solution methods strategy iteration or linear programming (LP, a QP with an objective function of order 1). However, as [34] pointed out for the mean payoff objective, we do not need the precise solution of the induced MDP, but it suffices to know that an action is better than all others. So we can also use bounded value iteration and check that the lower bound of one action is larger than the upper bound of all other actions, and thus we can stop the algorithm earlier. This approximation and the fact that VI tends to be the fastest methods in MDPs can speed up the solving. Using unguaranteed VI is dangerous, as it might stop too early and return a wrong strategy.

4.3. Warm start

All three solution methods can benefit from prior knowledge. VI and quadratic/higher-order programs (QP/HOP) can immediately use initial solution vectors obtained by domain knowledge or any precomputation. For VI, it is necessary to know whether it is an upper or lower estimate to use the prior knowledge correctly. The QP/HOP optimization process can start at the given initial vector.

SI can use the information of an initial estimate to infer a good initial strategy, as already suggested in [34]. This reduces the number of iterations of the main loop and thus the runtime. However, we have to ensure that the resulting strategy is proper. For example, we can check whether the target and sink states are reached almost surely from every state, and if not, we change the strategy to an attractor strategy where necessary, preferring those allowed actions that have a higher value.

We can also improve the MDP-solving for SI (Line 6 of Algorithm 2) by giving it the knowledge we currently have. We anyway save the estimate L of the previous iteration and, since the strategies of Maximizer get monotonically better, L certainly is a lower bound for the values in the MDP.

Even in the absence of domain knowledge or sophisticated precomputations, we can run unguaranteed VI first in order to get some estimates of the values. Then we can use those estimates for SI and QP/HOP. Note that this is similar in spirit to the idea of optimistic value iteration [28]: utilize VI's ability to usually deliver tight lower bounds and then verify them.

4.4. Topological improvement

For MDPs, topological improvements have been proposed for VI [22] and for SI [34, Algorithm 3]. These utilize the fact that the underlying graph of the MDP can be decomposed into a directed acyclic graph of strongly connected components (SCC). Intuitively speaking, there are parts of the graph that, after leaving them, can never be reached again. Their value depends solely on the parts of the state space that come after them. So instead of computing the values on the whole game at once, one can iterate over the SCC in a backwards fashion, starting with the target and sink states and then propagating the information and solving the SCCs one by one according to their topological ordering.

This idea was extended to BVI for MDPs in [4]. The proof generalizes to SGs, as the basic argument of the topological ordering of SCCs is independent from introducing a second player. As the proof relies on the solutions for the later components being ε-precise, solving those components with the precise methods SI or QP is of course also possible.

5. Experimental comparison

First, we give all the meta information about the experimental setup and the way we describe the results in Section 5.1. Then, in Section 5.2 to 5.4 we compare different variants of the same algorithm, including a detailed look at the behaviour of QP on models with many actions. Finally in Section 5.5 we compare the between the algorithm classes, considering runtime, memory and the bigger theoretical context.

5.1. Experimental setup

5.1.1. Implementation

We implemented all our algorithms as an extension of PRISM-games 3.0 [36]. The implementation is available via github.[7] For BVI, we reimplemented the algorithm as described in [32]; for SI and QP, this is the first implementation in PRISM-games. To solve the quadratic program, we used Gurobi[8] or CPLEX.[9] For the higher-order programs, we constructed them with AMPL and solved them with MINOS.[10]

[7] https://github.com/ga67vib/Algorithms-For-Stochastic-Games.
[8] https://www.gurobi.com/.
[9] https://www.ibm.com/analytics/cplex-optimizer.
[10] https://ampl.com/products/solvers/solvers-we-sell/minos/.

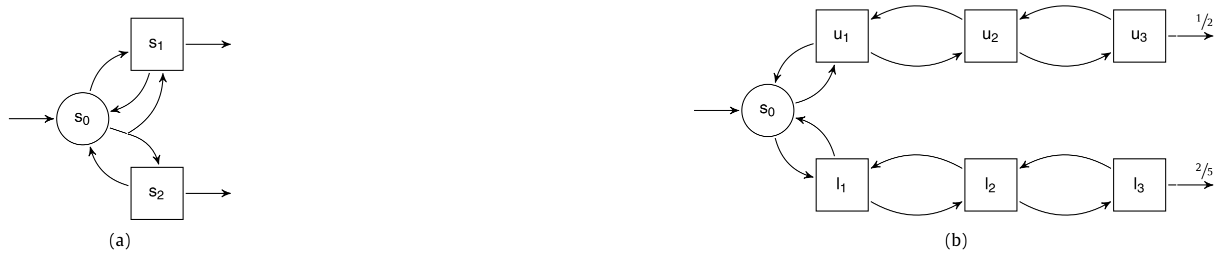


Fig. 5. Fig. 5a depicts the MEC that is used in the handcrafted MulMec model, as well as in cdmsnMEC and dice50MEC. Fig. 5b shows the handcrafted scalable model "BigMec" with $N = 3$. In this model, there are $2 \cdot N + 1$ states in the MEC and a dedicated target and sink. The initial state is a Minimizer state, leading to two chains of length N of Maximizer states. The fractions on the leaving actions denote the values; they are obtained by a transition to the target state with the given probability and to the sink with the remaining probability. The value of every state in the upper chain is 0.5; the initial state and the lower chain have value 0.4.

5.1.2. Setup

All experiments were conducted on a Linux Manjaro server with a 3.60 GHz Intel(R) Xeon(R) W-2123 CPU and 64 GB of RAM. We used a timeout of 15 minutes and set the java heap size for PRISM-games to 32 GB and the stack size to 16 GB.[11] The precision for BVI was set to 10^{-6}. In theory, the other algorithms are precise. In practice, QP and higher-order programming can have numerical problems; we mark these cases with red background colour in the table (this is only visible in the web version of the article).

For SI, even when using strategy iteration to solve the MDP of the opponent, PRISM still solves the resulting Markov chain by value iteration. For all of the models except hm, our solution is still precise, because we detect and fix probabilistic cycles of size 1. The model hm is the only one in our benchmark set that has larger probabilistic cycles that cause issues with convergence. However, this implementation detail does *not* imply that SI is in general not precise. For example, solving the Markov chain with linear programming would result in precise solutions.

5.1.3. Case studies

On the one hand, we used all case studies we are aware of that model a realistic scenario as a simple stochastic game. On the other hand, we generated some examples of interesting theoretical extreme cases ourselves, to see the behaviour the algorithms exhibit and estimate the impact of certain subcomponents.

As realistic case studies, we use the ones that are distributed with PRISM-games 3.0 and available on their case-study-website.[12] These are coins, prison_dil, adt, charlton, cdmsn, cloud, mdsm, dice and two_investors. Moreover, we use the HW and AV models which were introduced in [13]. The first index indicates the size of the square grid a robot moves on, the second index denotes the property that is checked.

As handcrafted models, we prepended the MEC-free models dice and cdmsn with a single small MEC (depicted in Fig. 5a) in order to judge the impact of such a single MEC. The exits lead to the initial state of the original model with some probability, and the remaining probability leads to a sink. Further, we used the adversarial model for value iteration from [25] (called hm[13]) as well as two newly handcrafted models with either one large MEC (BigMec) or many 3 state MECs (MulMec). In BigMec, there is a MEC with two chains of N Maximizer states, see Fig. 5b. In MulMec, a single MEC is repeated N times. For the first N-1 repetitions, both exits lead to s_0 of the next MEC with some probability and to s_0 of the current MEC with the rest. For the last repetition, both exits lead to some probabilistic combination of target and sink.

Note that the largest models we use in the paper have less than 500,000 states, so we can only make limited statements about how the algorithms scale when applied to extremely large, realistic SGs. A reason for few benchmarks being available probably is that so far there were no guaranteed solvers for simple stochastic games. We hope that our implementation will motivate others to model their problems as SGs and thus enable further analysis on realistic case studies. Moreover, we discuss at the end of Section 5.5 that structural properties of a model are more important than its size. This indicates that to make claims about scalability to extremely large models, we also need to take into account their structural properties.

5.1.4. Table layout and considered algorithms

We first analyse the impact of our optimizations by comparing different variants of the same algorithm: value iteration in Section 5.2 and Table 1; strategy iteration in Section 5.3 and Table 2; and quadratic respectively higher-order programming in Section 5.4 and Table 3. Based on this, we select the same variants of each algorithm and compare between the algorithms in Section 5.5; here we consider both runtime in Table 4 and memory in Table 5 and comment on the context of the theoretical properties of the algorithms.

[11] -javamaxmem 32g -javastack 16g.
[12] http://www.prismmodelchecker.org/games/casestudies.php.
[13] hm is short for haddad-monmege, the names of the authors of [25].

Every runtime table includes the verification times (in seconds) of several variations of the algorithms, i.e. different optimizations enabled or disabled. An X in the table denotes that the computation did not finish within 15 minutes. A red background colour indicates that the returned result was wrong, i.e. off by more than the allowed precision (the red background colour is only visible in the web version of this article). All results that are wrong are off significantly, i.e. by more than 0.1. The four left-most columns are shared by all tables and give relevant properties that are interesting when analysing the performance. They show the name of the considered case study, its size, the maximum/average number of actions and the number of MECs; for the latter, note that this number excludes trivial MECs, i.e. sink or target states and MECs in regions of the graph that are either not reachable or identified as trivially having value 0/1 by the precomputation. The case studies are roughly sorted by increasing size/difficulty, with scaled versions of the same model grouped together. For the comparison within the algorithms, we report all case studies we used; for the comparison between the algorithms, we restricted to the most interesting ones.

We consider the following algorithms:

VI	denotes unguaranteed value iteration.
BVI	denotes bounded value iteration.
Sim	denotes the simulation-based variant of value iteration [32]. As this is a randomized algorithm, we report both the maximum and median runtime of three tries.
SI	denotes strategy iteration.
[19]	is the original QP algorithm that transforms the SG into normal form.
[19]$_\varepsilon$	is a small modification of the original algorithm that only introduces the ε transition to make it stopping if necessary.
QP	denotes quadratic programming for arbitrary SGs, only transforming the SG to satisfy the **2Act**-constraint.
HOP	is higher-order programming for arbitrary SGs.

For each algorithm, optimizations are indicated as follows:

T	as a prefix denotes the topological optimization.
W	as a superscript denotes the warm start optimization, i.e. unguaranteed value iteration to guess a good initial strategy.
D	in the subscript of BVI denotes an optimization suggested in [32]: instead of "deflating" (finding SEC candidates and decreasing their over-approximation) in every step, this costly operation is only carried out every 100 steps.
SI/BV	in the subscript of SI indicates the method used to solve the MDP, either SI or BVI.
C/G	in the subscript of QP indicates the used solver, C for CPLEX and G for Gurobi.

5.2. Value iteration (Table 1)

We now analyse the results of the different variants of value iteration. We could reproduce most of the findings of [32]: the overhead of BVI compared to unguaranteed VI is usually negligible and not performing the expensive deflate operation in every step speeds up the computation. Unguaranteed VI fails on three of the models. In contrast to [32], we found no model where the simulation based asynchronous version BRTDP was significantly faster than BVI. In fact, BRTDP is only faster on a single model (cloud6, which is not too significant as BVI also takes only 2 seconds), but significantly slower on many others, often even failing to produce results in time. Note that the implementation of BRTDP was in PRISM-games 2, and thus the disadvantage may also have technical reasons; improvements in the simulation engine or data structures might lead to speed-ups that make BRTDP competitive again.

The new topological variant of BVI (called TBVI) is usually in the same order of magnitude as the default approach. The exception to this are the models AV15_15 and especially MulMec, where TBVI fails. This is because TBVI solves every SCC of the model with a precision of ε. When one SCC is solved and has a difference of almost exactly ε between upper and lower bound, SCCs depending on it have suboptimal information about their exits. This not only slows down convergence, but can even aggregate and lead to precision problems when there is a chain of many SCCs. Indeed, for MulMec with N>15, the progress that TBVI makes in the 15th SCC it considers is smaller than machine precision, and thus the computation is stuck. This problem is not specific to topological VI for SGs, but can also occur for MDPs.

5.3. Strategy iteration (Table 2)

We now analyse the results of the different variants of strategy iteration. Using BVI for the opponent's MDP and the warm start usually lead to small speed-ups. We did not consider using linear programming for the opponent's MDP, as it is not supported by PRISM. The topological variant is significantly faster in two_inv and MulMec_e3, but on the rest of the models performs very similar to the non-topological version. Combining topological SI and BVI for the opponent's MDP leads to the same problems with MulMec as when using topological BVI. In contrast, topological SI with SI for the opponent's

Table 1
Table with all experimental results on variations of value iteration. See Section 5.1.4 for a description of table layout and considered algorithms, and Section 5.2 for the analysis.

Case Study	States	Acts	MECs	VI	Sim	BVI	BVI_D	TBVI	TBVI_D
coins	19	2 | 1.1	0	<1	<1	<1	<1	<1	<1
prison_dil	102	3 | 1.3	0	<1	<1	<1	<1	<1	<1
adt	305	4 | 1.2	0	<1	X	<1	<1	<1	<1
charlton1	502	3 | 1.5	0	<1	<1	<1	<1	<1	<1
charlton2	502	3 | 1.5	0	<1	<1	<1	<1	<1	<1
cdmsn	1,240	2 | 1.6	0	<1	<1	<1	<1	<1	<1
cdmsnMec	1,244	2 | 1.6	1	<1	<1	<1	<1	<1	<1
cloud5	8,842	11 | 3.9	520	<1	9 | 6	<1	<1	<1	<1
cloud6	34,954	13 | 4.4	2176	<1	<1	2	2	8	3
mdsm1	62,245	2 | 1.3	0	4	7 | 7	5	5	3	3
mdsm2	62,245	2 | 1.3	0	<1	15 | 14	<1	<1	<1	<1
dice20	16,915	2 | 1.4	0	<1	262 | 222	<1	<1	<1	<1
dice50	96,295	2 | 1.4	0	6	X	6	6	6	6
dice50Mec	96,299	2 | 1.4	1	6	X	6	6	6	6
two_inv	172,240	3 | 1.3	0	13	X	14	13	13	13
hw5_1	25,000	5 | 2.4	0	<1	87 | 18	<1	<1	<1	<1
hw5_2	25,000	5 | 2.4	0	<1	<1	<1	<1	<1	<1
hw8_1	163,840	5 | 2.5	0	3	X	3	3	3	3
hw8_2	163,840	5 | 2.5	0	<1	<1	<1	<1	<1	<1
hw10_1	400,000	5 | 2.5	0	10	X	10	10	10	11
hw10_2	400,000	5 | 2.5	0	<1	<1	<1	<1	1	1
AV10_1	106,524	6 | 2.1	0	<1	X	<1	<1	<1	<1
AV10_2	106,524	6 | 2.1	6	70	X	81	72	82	70
AV10_3	106,524	6 | 2.1	1	47	X	46	45	52	55
AV15_1	480,464	6 | 2.1	0	1	X	1	1	1	1
AV15_2	480,464	6 | 2.1	6	729	X	X	X	X	X
AV15_3	480,464	6 | 2.1	1	492	X	X	X	X	X
hm_30	61	1 | 1.0	0	<1	X	X	X	X	X
MulMec_e2	302	2 | 1.9	100	<1	X	<1	2	X	X
MulMec_e3	3,002	2 | 2.0	1000	4	X	36	151	X	X
MulMec_e4	30,002	2 | 2.0	10000	591	X	X	X	X	X
BigMec_e2	203	2 | 1.9	1	<1	X	<1	<1	<1	<1
BigMec_e3	2,003	2 | 2.0	1	<1	X	6	1	6	1
BigMec_e4	20,003	2 | 2.0	1	161	X	856	226	871	230

MDP works, because SI solves the SCCs precisely, allowing the SCCs depending on the previous solutions to converge as well.

5.4. Quadratic programming

5.4.1. Runtime comparison (Table 3)

We now analyse the results of the different variants of quadratic or higher-order programming. The original version of quadratic programming [19], that requires to transform the SG into normal form is impractical, producing a result within 15 minutes for only 3 of 34 case studies, and even there taking a lot more time than the improved version. After dropping the constraints **No1Act** and **½Probs**, it can correctly solve 14 of the case studies in time. Both of these variants are exhibit the numerical errors described in Section 4.1.4 and produce incorrect results on some models. The quadratic program obtained after dropping all but the **2Act** constraint is solved successfully by Gurobi in 19 instances; CPLEX on the other hand only solves 7 instances correctly, one time even reporting an incorrect result. The warm start helps Gurobi on the model charlton1. Several other times, Gurobi's internal heuristics are deemed more promising by the program and it discards the given initial suggestion; thus, the warm start sometimes incurs a slight overhead.

Dropping all constraints, the higher-order program (HOP) with the solver Minos gets the correct result on 26 instances. The HOP approaches are typically faster than the QP ones, except on the models HW and AV. The topological variant of both the quadratic programs as well as the higher-order program can lead to significant speed-ups, for example on charlton1, cloud5 and cloud6, mdsm1, two_inv or HW10_10_2. On all models but cloud, the topological HOP is strictly faster than all other algorithms in this subsection.

To estimate the impact of the EC solution method, we used several handcrafted or modified models: A single large MEC (BigMec_e2, with a MEC of size 201) cannot be solved with our approach, as there are too many choices; possibly, some heuristic could help identify reasonable strategies. In contrast, small MECs do not affect runtime a lot. The models cdmsn

Table 2
Table with all experimental results on variations of strategy iteration. See Section 5.1.4 for a description of table layout and considered algorithms, and Section 5.3 for the analysis.

Case Study	States	Acts	MECs	$\mathrm{SI}_{\mathrm{SI}}$	$\mathrm{SI}_{\mathrm{BV}}$	$\mathrm{SI}^{\mathrm{W}}_{\mathrm{SI}}$	$\mathrm{SI}^{\mathrm{W}}_{\mathrm{BV}}$	$\mathrm{TSI}_{\mathrm{SI}}$	$\mathrm{TSI}_{\mathrm{BV}}$	$\mathrm{TSI}^{\mathrm{W}}_{\mathrm{SI}}$	$\mathrm{TSI}^{\mathrm{W}}_{\mathrm{BV}}$
coins	19	2 | 1.1	0	<1	<1	<1	<1	<1	<1	<1	<1
prison_dil	102	3 | 1.3	0	<1	<1	<1	<1	<1	<1	<1	<1
adt	305	4 | 1.2	0	<1	<1	<1	<1	<1	<1	<1	<1
charlton1	502	3 | 1.5	0	<1	<1	<1	<1	<1	<1	<1	<1
charlton2	502	3 | 1.5	0	<1	<1	<1	<1	<1	<1	<1	<1
cdmsn	1,240	2 | 1.6	0	<1	<1	<1	<1	<1	<1	<1	<1
cdmsnMec	1,244	2 | 1.6	1	<1	<1	<1	<1	<1	<1	<1	<1
cloud5	8,842	11 | 3.9	520	<1	<1	<1	<1	<1	<1	<1	<1
cloud6	34,954	13 | 4.4	2176	<1	<1	<1	<1	<1	<1	<1	<1
mdsm1	62,245	2 | 1.3	0	5	5	6	6	3	3	3	3
mdsm2	62,245	2 | 1.3	0	<1	<1	<1	<1	<1	<1	<1	<1
dice20	16,915	2 | 1.4	0	<1	<1	<1	<1	<1	<1	<1	<1
dice50	96,295	2 | 1.4	0	7	6	7	7	6	6	6	6
dice50Mec	96,299	2 | 1.4	1	6	7	7	7	6	6	6	6
two_inv	172,240	3 | 1.3	0	38	19	37	22	11	11	13	12
HW5_1	25,000	5 | 2.4	0	<1	<1	<1	<1	<1	<1	<1	<1
HW5_2	25,000	5 | 2.4	0	<1	<1	<1	<1	<1	<1	<1	<1
HW8_1	163,840	5 | 2.5	0	4	4	4	4	4	4	4	4
HW8_2	163,840	5 | 2.5	0	<1	<1	<1	<1	<1	<1	<1	<1
HW10_1	400,000	5 | 2.5	0	11	11	11	11	10	11	11	11
HW10_2	400,000	5 | 2.5	0	2	2	2	2	1	1	2	2
AV10_1	106,524	6 | 2.1	0	<1	<1	<1	<1	<1	<1	<1	<1
AV10_2	106,524	6 | 2.1	6	83	79	79	80	79	76	77	79
AV10_3	106,524	6 | 2.1	1	55	50	58	52	60	X	60	X
AV15_1	480,464	6 | 2.1	0	2	2	2	2	1	2	2	2
AV15_2	480,464	6 | 2.1	6	797	825	843	791	X	X	X	X
AV15_3	480,464	6 | 2.1	1	529	500	512	497	X	X	X	X
hm_30	61	1 | 1.0	0	X	X	X	X	X	X	X	X
MulMec_e2	302	2 | 1.9	100	<1	X	<1	X	<1	X	<1	X
MulMec_e3	3,002	2 | 2.0	1000	56	X	56	X	3	X	3	X
MulMec_e4	30,002	2 | 2.0	10000	X	X	X	X	X	X	X	X
BigMec_e2	203	2 | 1.9	1	<1	<1	<1	<1	<1	<1	<1	<1
BigMec_e3	2,003	2 | 2.0	1	1	1	1	1	1	1	1	1
BigMec_e4	20,003	2 | 2.0	1	X	X	X	X	X	X	X	X

and dice50 prepended with a single three-state MEC are solved in the same time as the original models. Even a chain of 1000 three-state MECs can be solved quickly (MulMec_e3).

5.4.2. Additional analysis: impact of the number of actions (Fig. 6)

It seems plausible that a high number of actions is problematic for the QP/HOP-algorithms, since for QP the SG has to be blown up and for HOP the degree of the objective function increases. Thus, independent of the runtime comparison, in this subsection we investigate how the number of actions per state affects the performance of the algorithms.

The highest number of actions per state in the real-word case studies is 13 and the highest average is 4.4. We want to analyse the performance with even higher number of actions and on more models. However, the PRISM language, which is used for encoding the models, does not have the feature of scalable number of actions. Hence, we used random generation to obtain models with a large number of actions per state.

Random generation of models: We have the following requirements for the randomly generated models: firstly they should not be trivial, so we filter out models that are solved by standard precomputations or have value 1. Secondly, all models should be of the same size, to ensure that effects we see are not due to an increase in size, but rather due to increasing the number of actions. Similarly, the models should not contain non-trivial MECs, as we have already seen that these strongly influence on the runtime of the algorithms. Their presence could obfuscate the impact of the number of actions.

We use two different approaches to generate the random models, where in both, every state has equal chance of being a Maximizer or Minimizer state; also, both are parametrized by the designated number of actions per state m.

- Random tree: The main underlying topology of this model is a tree. That means that every inner node has m actions that have a non-zero probability of leading to states of the next level. Additionally, there may be transitions to random other states of the tree, where the smallest transition probability is 0.1. The generated tree can consist of a single SCC – if all states have some path back to the root – or of as many SCCs as there are states in the tree – if no state has a transition to an ancestor. Every state has a non-zero probability of reaching a leaf state, which are either targets or sinks. Thus, there can be no non-trivial MECs. We report our results for tree size 1000, because QP could still solve several models of this size. For larger trees, it regularly timed out, i.e. needs more than 15 minutes.
  For every $m \in \{2, 5, 10, 30, 50, 70, 90, 100\}$, we generated 20 trees, resulting in a total of 160 models.

Table 3
Table with all experimental results on variations of quadratic/higher order programming. See Section 5.1.4 for a description of table layout and considered algorithms, and Section 5.4 for the analysis.

Case Study	States	Acts	MECs	[19]	$[19]_\varepsilon$	QP_C	QP_G	QP_G^W	HOP	TQP_G	TQP_G^W	THOP
coins	19	2 | 1.1	0	X	X	<1	<1	<1	<1	<1	<1	<1
prison_dil	102	3 | 1.3	0	X	8	11	9	8	<1	<1	<1	<1
adt	305	4 | 1.2	0	X	<1	<1	<1	<1	<1	<1	<1	<1
charlton1	502	3 | 1.5	0	X	179	X	180	144	<1	<1	<1	<1
charlton2	502	3 | 1.5	0	X	X	X	X	X	<1	X	X	<1
cdmsn	1,240	2 | 1.6	0	17	<1	<1	<1	<1	<1	<1	<1	<1
cdmsnMec	1,244	2 | 1.6	1	X	<1	<1	<1	<1	<1	<1	<1	<1
cloud5	8,842	11 | 3.9	520	X	X	X	X	X	1	3	3	4
cloud6	34,954	13 | 4.4	2176	X	X	X	X	X	8	14	15	21
mdsm1	62,245	2 | 1.3	0	X	X	X	X	X	107	6	7	3
mdsm2	62,245	2 | 1.3	0	X	<1	1	<1	<1	5	<1	<1	<1
dice20	16,915	2 | 1.4	0	X	3	X	3	3	2	<1	<1	<1
dice50	96,295	2 | 1.4	0	X	X	X	X	X	15	6	6	6
dice50Mec	96,299	2 | 1.4	1	X	X	X	X	X	12	6	7	6
two_inv	172,240	3 | 1.3	0	X	X	253	X	X	432	X	X	20
HW5_5_1	25,000	5 | 2.4	0	13	1	X	1	1	3	<1	<1	<1
HW5_5_2	25,000	5 | 2.4	0	X	1	X	1	1	3	<1	<1	<1
HW8_8_1	163,840	5 | 2.5	0	96	22	X	20	20	17	4	4	4
HW8_8_2	163,840	5 | 2.5	0	X	11	X	11	11	47	2	2	<1
HW10_10_1	400,000	5 | 2.5	0	X	98	X	98	98	44	12	12	11
HW10_10_2	400,000	5 | 2.5	0	X	48	X	49	49	286	4	4	1
AV10_10_1	106,524	6 | 2.1	0	X	3	X	3	3	9	<1	<1	<1
AV10_10_2	106,524	6 | 2.1	6	X	X	X	X	X	X	X	X	X
AV10_10_3	106,524	6 | 2.1	1	X	X	X	X	X	X	X	X	X
AV15_15_1	480,464	6 | 2.1	0	X	15	X	10	11	41	3	3	2
AV15_15_2	480,464	6 | 2.1	6	X	X	X	X	X	X	X	X	X
AV15_15_3	480,464	6 | 2.1	1	X	X	X	X	X	X	X	X	X
hm_30	61	1 | 1.0	0	735	<1	<1	<1	<1	<1	<1	<1	<1
MulMec_e2	302	2 | 1.9	100	X	X	X	<1	<1	<1	<1	<1	<1
MulMec_e3	3,002	2 | 2.0	1000	X	X	X	4	7	3	5	8	4
MulMec_e4	30,002	2 | 2.0	10000	X	X	X	X	X	X	X	X	X
BigMec_e2	203	2 | 1.9	1	X	X	X	X	X	X	X	X	X
BigMec_e3	2,003	2 | 2.0	1	X	X	X	X	X	X	X	X	X
BigMec_e4	20,003	2 | 2.0	1	X	X	X	X	X	X	X	X	X

- Repeated tree: The underlying topology of this model is a series of 20 trees, where every tree has 400 states. This ensures the problem has a certain complexity, as there are at least 20 SCCs. Every tree is generated in similar fashion as described in the random-tree-approach. For every $m \in \{2, 5, 10, 30, 50, 70, 100, 200\}$, we generated two models, resulting in a total number of 16.

Analysis: In Fig. 6a, we see the results on the random tree models. For both QP and HOP, there is no visible correlation between number of actions per state. Contrary to the claim that more actions increase the runtime, we see that most time outs occur for models with very low m.

Fig. 6b shows the results on the repeated tree models. QP times out for all of them, while HOP is able to solve all but one. Here, it looks like there is a linear relation between the number of actions per state and the runtime. Still, even for 200 actions per state, HOP is able to solve a benchmark.

In conclusion: in real case studies the average number of actions per state is typically smaller than 10. Since HOP is able to solve models with more than 100 actions per state, this model property will likely not be prohibitive. Moreover, we saw timeouts occur even for small number of actions per state, indicating that other structural properties of the models are more important.

5.5. Comparison between algorithm classes

Comparing all classes of algorithms, we differentiate three topics: First the runtime comparison, similar to the previous subsections, then a memory comparison and finally a comparison of theoretical properties to set the practical results into context.

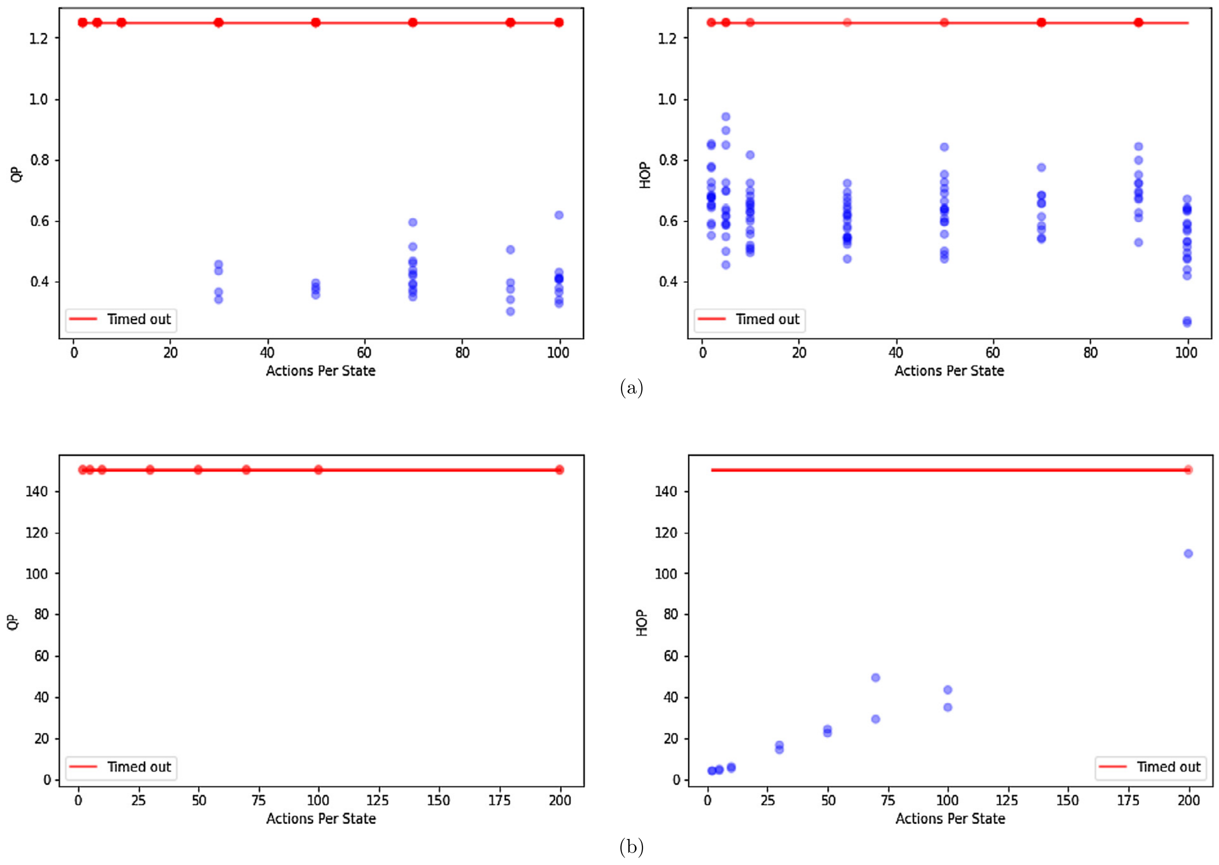


Fig. 6. Analysis of the number of actions per state on the runtime of QP and HOP. Fig. 6a and Fig. 6b show the results on random models generated by the random-tree-approach respectively the repeated-tree-approach. The y-axis of all plots denotes runtime of the algorithm in seconds. The red line denotes a time out, which we report after 15 minutes (The colour is only visible in the web version of the article).

5.5.1. *Runtime comparison (Table 4 and Fig. 7)*

For the runtime comparison between the different classes of algorithms, we provide the experimental data in two forms:

- In Fig. 7 we visualize the number of benchmarks solved as well as the aggregated time to solve them. We refer to the caption of Fig. 7 for a detailed description of the meaning of the lines.
- In Table 4 we present a subset of benchmarks and algorithms from the previous tables. For each algorithm we selected one variant with and one without the topological optimization, because it can have significant positive as well as negative impact. We also included one QP and one HOP, as there is a fundamental difference between the approaches. Regarding the other optimizations, we selected the ones that minimize the aggregated runtime over all benchmarks.

Fig. 7 provides an overview of the performance of all algorithms. Note that this plot immensely depends on the selection of benchmarks. By e.g. including several variants of the hm model, one could skew it and make all VI based algorithms seem worse. Still, to the best of our knowledge our set of benchmarks is somewhat balanced which allows us to extract general trends.

We see that many variants of QP and the simulation based VI variant end significantly earlier than all SI and deterministic VI algorithms. The worst algorithm is the unmodified QP variant from [19] with 3 benchmarks, while using TQP, HOP or THOP at least comes close to the SI and VI variants with 24 respectively 25 solved benchmarks. The SI and VI variants to the right of the plot all solve between 26 and 31 benchmarks. For the first 26 benchmarks they are very comparable in runtime to each other and to TQP and THOP. They only increase above 250 seconds when solving the hardest case studies.

We now turn to the more detailed analysis of Table 4. Observe that BVI and SI succeed/fail on different models. While BVI succeeds in the largest version of BigMec, SI is the only one to solve the large and complicated models AV15_15_2/3, and TSI has the best runtime in MulMec_e3. TSI benefits from a model with small subcomponents that it can quickly solve (as in MulMec), while BVI is fast in subgraphs without probabilistic cycles (as the large chains in BigMec). Depending on the model structure, both of these algorithms are a viable choice.

Table 4
Table for the runtime comparison between the classes of algorithms. See Section 5.1.4 for a description of table layout and considered algorithms, and Section 5.5.1 for the analysis.

Case Study	States	Acts	MECs	BVI_D	$TBVI_D$	SI_{BV}	TSI^{W}_{SI}	QP^{W}_{G}	THOP
prison_dil	102	3/1.3	0	<1	<1	<1	<1	8	<1
charlton1	502	3/1.5	0	<1	<1	<1	<1	144	<1
cdmsn	1,240	2/1.6	0	<1	<1	<1	<1	<1	<1
cdmsnMec	1,244	2/1.6	1	<1	<1	<1	<1	<1	<1
cloud6	34,954	13/4.4	2176	2	3	<1	<1	X	21
mdsm1	62,245	2/1.3	0	5	3	5	3	X	3
dice50	96,295	2/1.4	0	6	6	6	6	X	6
dice50Mec	96,299	2/1.4	1	6	6	7	6	X	6
two_inv	172,240	3/1.3	0	13	13	19	13	X	20
HW10_10_1	400,000	5/2.5	0	10	11	11	11	98	11
HW10_10_2	400,000	5/2.5	0	<1	1	2	2	49	1
AV10_10_1	106,524	6/2.1	0	<1	<1	<1	<1	3	<1
AV10_10_2	106,524	6/2.1	6	72	70	79	77	X	X
AV10_10_3	106,524	6/2.1	1	45	55	50	60	X	X
AV15_15_1	480,464	6/2.1	0	1	1	2	2	11	2
AV15_15_2	480,464	6/2.1	6	X	X	825	X	X	X
AV15_15_3	480,464	6/2.1	1	X	X	500	X	X	X
hm_30	61	1/1.0	0	X	X	X	X	<1	<1
MulMec_e2	302	2/1.9	100	2	X	X	<1	<1	<1
MulMec_e3	3,002	2/2.0	1000	151	X	X	3	7	4
MulMec_e4	30,002	2/2.0	10000	X	X	X	X	X	X
BigMec_e2	203	2/1.9	1	<1	<1	<1	<1	X	X
BigMec_e3	2,003	2/2.0	1	1	1	1	1	X	X
BigMec_e4	20,003	2/2.0	1	226	230	X	X	X	X

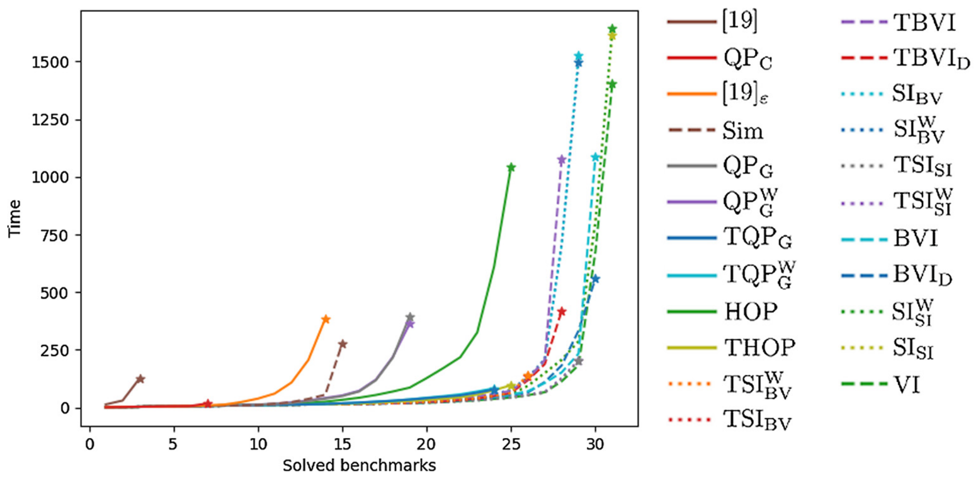


Fig. 7. An overview of the aggregated runtimes of all 23 considered algorithms. For each algorithm, we ordered the benchmarks it solved by time taken to solve them. A point (x, y) indicates that to solve the "first" x benchmarks, the algorithm had an aggregated runtime of y seconds. The star at the end of a line highlights the maximum number of benchmarks the algorithm could solve, i.e. all other case studies could not be solved within the timeout of 15 minutes. VI variants are depicted with dashed, SI with dotted and QP/HOP with solid lines. Note that the legend is ordered by performance, i.e. algorithms with less solved benchmarks (and more required time in case of ties) are listed first.

Topological higher-order programming (THOP), the improved version of QP, is comparable on many case studies, but still a lot worse on several others, e.g. cloud6 and BigMec. This volatility is even more pronounced for the QP approaches. The reason for this can be MECs which blow up the QP, but it can also happen in models with few or no MECs; in the latter case, we do not know which property of the model slows down the solving. Note that QP and HOP are able to solve models with many small MECs (MulMec_e3) quickly, while already one medium sized MEC (BigMec_e2) makes it infeasible.

The model hm_30 from [25] deserves special attention: it only is an MC, i.e. there are no nondeterministic choices. Still, as it is a handcrafted extreme case, we can observe interesting behaviour of the algorithms. Since hm is an adversarial example for VI, of course all variants of VI fail. Moreover, since the solvers for MDPs and MCs in PRISM use VI, and since SI relies on those solvers, the PRISM implementation of SI also fails on the model (see Section 5.1.2 for more details). QP shines on this model, being the only method to solve it. However, for increasing parameter N, the probabilities in hm_N become so small that they are at the border of numerical stability. If the state-chains in the model were prolonged by one

Table 5
Table for the memory comparison between the classes of algorithms. See Section 5.1.4 for a description of considered algorithms, and Section 5.5.2 for the analysis.

	BVI	TBVI	SI	[19]	$[19]_\varepsilon$	QP_C	QP_G	HOP
# min. memory used	31/34	26/34	32/34	4/34	18/34	27/34	24/34	34/34
Maximum overhead	4×	29×	2×	88×	7×	8×	3×	1×

more state (i.e. the parameter N is set to 31), QP has numerical problems and reports an incorrect result. Similarly, noting that THOP reports an incorrect result on hm_30, one can experimentally find out that THOP succeeds only for $N \leq 25$. So if the model exhibits very small probabilities, one has to consider using a solver capable of arbitrary-precision arithmetic.

Note that the size of the model is only loosely correlated with the difficulty of solving it. For example, the largest case study AV15_1 with 480,464 states can be solved in few seconds by all algorithm classes. In contrast, already the small model hm_30 with 61 states takes a long time for all VI-based algorithms and can induce precision problems for all QP-based ones. Further, the medium-sized model MulMec_e4 with 30,002 states is not solved in time by any algorithm. The reason for the difficulty of the hm model is the number of probabilistic loops as well as the small probability of the path from initial state to target. The difficulty of MulMec and BigMec models stems from the number and size of MECs. We see that these structural properties of the underlying graph have a huge effect on the performance of the algorithms.

5.5.2. Memory comparison (Table 5)

For a general overview, we report the minimum and maximum memory used in our experiments: on the smallest model coins, almost all algorithms need around 80 MB. Most memory is needed on the largest models AV and HW, where BVI, SI and HOP need between 1 GB and 1.5 GB, while QP can go up to 9.5 GB.

For a more detailed analysis, consider Table 5. The first row shows the number of times an algorithm needed less than twice as much memory as the algorithm who used the minimum memory. In other words, this is the number of times the algorithm performed “almost optimally” memory-wise. The second row shows the maximum relative overhead when compared to the minimum. For example, on one benchmark BVI needed 4 times as much as the minimum. Note that we omit optimizations in this table when they do not make a difference. For example, all 8 variants of SI perform the same in terms of memory.

We see that the memory usage of BVI, SI and HOP is very similar. HOP actually is close to the minimum memory on all benchmarks, while BVI and SI have a few where they need slightly more, namely BigMec_1e3, ManyMec_1e2 (both SI and BVI) and hm_30 (only BVI). Most QP variants perform worse, needing more memory several times. Moreover, this typically occurs on the large models AV and HW, where even the minimum memory is more than 1 GB, and thus a relative difference of 3 times larger is quite significant.

The topological variant of BVI is significantly worse on AV10_3 (8×) and BigMec_1e4 (29×). Interestingly, this problem does not occur for the topological variants of SI, QP or HOP. The original approach from [19] almost always needs more memory, often significantly so. The need for 88 times the memory occurs in the case study cdmsnMEC, with a difference of 120 MB to 10.5 GB. This highlights another[14] practical drawback of the transformation into normal form which blows up the game.

5.5.3. Theoretical comparison

To set the practical results into a more general context, we recall relevant theoretical properties of all three algorithm classes.

VI has two main drawbacks when compared to the other two: (i) it is an imprecise algorithm, i.e. it only converges in the limit and thus does not return the precise result in finite time. This can be addressed by introducing an additional rounding step after an exponential number of steps [10]; however, this is impractical, as the algorithm then *always* needs exponentially many steps. (ii) There are models where the exponential number of steps is necessary [5], so there is no hope to find a polynomial algorithm based on VI.

In contrast, SI and QP theoretically are precise methods, even though our practical implementation is not, since SI uses an imprecise, VI-based MDP solver and since QP exhibits numerical instability.

Both SI and QP still have the potential to yield a polynomial algorithm. For SI, this could be achieved by finding a polynomial bound for the number of iterations; currently, the best known bound is exponential. For QP, this could be achieved by finding a polynomially sized, convex QP (as solving convex QPs can be done in polynomial time [33]). Currently, our QP is exponential and it is unclear for both our and the previous polynomially sized QP [19] whether they are convex. If they are not, then the solving them is known to be NP-hard [42].

[14] Additionally to the one described in Section 4.1.4.

6. Conclusions

We have extended the three known classes of algorithm – value iteration, strategy iteration and quadratic programming – with several improvements and compared them both theoretically and practically.

In summary, for all algorithms, the structure of the underlying graph is more important than its size; thus knowledge about the model is relevant both for estimating the expected time, as well as the preferred algorithm and combination of optimizations. BVI and SI perform very similar on most models in our practical evaluation; each of them has some models where they are better. Quadratic/higher order programming is volatile and typically slower than the other two; however, the used solver has a huge impact, as we already see when changing between CPLEX, Gurobi and Minos. Thus, advances in the area of optimization problems could make this solution method the most practical.

In future work, we want to extend all algorithms to SGs with other objectives, e.g. total expected reward, mean payoff or parity. We expect the algorithms to work in a similar fashion as the ones presented for reachability in this paper, because for MDPs the other objectives are also similar to reachability. For example bounded value iteration in MDPs with total expected reward objective [4] differs mainly in the fact that one first has to compute an over-approximation, since it cannot start at 1 as the highest possible probability. Strategy iteration for MDPs with mean payoff [34] uses the same underlying idea as for reachability, but keeps track of two metrics (gain and bias) in order to decide how to improve a strategy. A key problem when extending bounded value iteration to total expected reward in SGs is that there are different semantics of the objective [15], and that in some cases the value is not the *least*, but the *greatest* fixpoint of the Bellman equations. Thus, value iteration from below can be stuck. To address this, probably a dual of the "deflating" [32] operation can be derived, which then increases the lower approximants in order to converge to the value.

A further direction for future work is coming up with a polynomially sized, convex QP. Using the normal form of [19], we have a polynomially sized QP. Possibly we could find a more practical polynomial QP by finding a solution for MECs that does not enumerate an exponential number of strategies. The key difficulty is in understanding whether the QP is convex or altering it so that it becomes convex. Overcoming this difficulty would give a positive answer to the long-standing open question whether simple stochastic games can be solved in polynomial time.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:
This research was funded in part by the German Research Foundation (DFG) projects 383882557 Statistical Unbounded Verification (SUV) and 427755713 Group-By Objectives in Probabilistic Verification (GOPro).

References

[1] Pranav Ashok, Krishnendu Chatterjee, Jan Kretínský, Maximilian Weininger, Tobias Winkler, Approximating values of generalized-reachability stochastic games, in: LICS, ACM, 2020, pp. 102–115.
[2] Pranav Ashok, Jan Kretínský, Maximilian Weininger, PAC statistical model checking for Markov decision processes and stochastic games, in: CAV (1), in: Lecture Notes in Computer Science, vol. 11561, Springer, 2019, pp. 497–519.
[3] Christel Baier, Joost-Pieter Katoen, Principles of Model Checking, MIT Press, 2008.
[4] Christel Baier, Joachim Klein, Linda Leuschner, David Parker, Sascha Wunderlich, Ensuring the reliability of your model checker: interval iteration for Markov decision processes, in: CAV (1), in: Lecture Notes in Computer Science, vol. 1042, Springer, 2017, pp. 160–180.
[5] Nikhil Balaji, Stefan Kiefer, Petr Novotný, Guillermo A. Pérez, Mahsa Shirmohammadi, On the complexity of value iteration, in: ICALP, in: LIPIcs, vol. 132, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 102.
[6] Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Kretínský, Marta Z. Kwiatkowska, David Parker, Mateusz Ujma, Verification of Markov decision processes using learning algorithms, in: ATVA, in: Lecture Notes in Computer Science, vol. 8837, Springer, 2014, pp. 98–114.
[7] Krishnendu Chatterjee, Luca de Alfaro, Thomas A. Henzinger, Strategy improvement for concurrent reachability and turn-based stochastic safety games, J. Comput. Syst. Sci. 79 (5) (2013) 640–657, https://doi.org/10.1016/j.jcss.2012.12.001.
[8] Krishnendu Chatterjee, Nathanaël Fijalkow, A reduction from parity games to simple stochastic games, in: GandALF, 2011, pp. 74–86.
[9] Krishnendu Chatterjee, Kristoffer Arnsfelt Hansen, Rasmus Ibsen-Jensen, Strategy complexity of concurrent safety games, in: MFCS, LIPIcs 83, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 55.
[10] Krishnendu Chatterjee, Thomas A. Henzinger, Value iteration, in: 25 Years of Model Checking, in: Lecture Notes in Computer Science, vol. 5000, Springer, 2008, pp. 107–138.
[11] Krishnendu Chatterjee, Thomas A. Henzinger, A survey of stochastic ω-regular games, J. Comput. Syst. Sci. 78 (2) (2012) 394–413, https://doi.org/10.1016/j.jcss.2011.05.002.
[12] Krishnendu Chatterjee, Thomas A. Henzinger, Barbara Jobstmann, Arjun Radhakrishna, Gist: a solver for probabilistic games, in: CAV, in: Lecture Notes in Computer Science, vol. 6174, Springer, 2010, pp. 665–669.
[13] Krishnendu Chatterjee, Joost-Pieter Katoen, Maximilian Weininger, Tobias Winkler, Stochastic games with lexicographic reachability-safety objectives, in: CAV (2), in: Lecture Notes in Computer Science, vol. 12225, Springer, 2020, pp. 398–420.
[14] Krishnendu Chatterjee, Koushik Sen, Thomas A. Henzinger, Model-checking omega-regular properties of interval Markov chains, in: FoSSaCS, in: Lecture Notes in Computer Science, vol. 4962, Springer, 2008, pp. 302–317.
[15] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, David Parker, Aistis Simaitis, Automatic verification of competitive stochastic systems, Form. Methods Syst. Des. 43 (1) (2013) 61–92, https://doi.org/10.1007/s10703-013-0183-7.
[16] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, Aistis Simaitis, Clemens Wiltsche, On stochastic games with multiple objectives, in: MFCS, in: Lecture Notes in Computer Science, vol. 8087, Springer, 2013, pp. 266–277.
[17] Chih-Hong Cheng, Alois Knoll, Michael Luttenberger, Christian Buckl, GAVS+: an open platform for the research of algorithmic game solving, in: TACAS, in: Lecture Notes in Computer Science, vol. 6605, Springer, 2011, pp. 258–261.

[18] Anne Condon, The complexity of stochastic games, Inf. Comput. 96 (2) (1992) 203–224, https://doi.org/10.1016/0890-5401(92)90048-K.
[19] Anne Condon, On algorithms for simple stochastic games, in: Advances in Computational Complexity Theory, in: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 13, DIMACS/AMS, 1993, pp. 51–71.
[20] Costas Courcoubetis, Mihalis Yannakakis, The complexity of probabilistic verification, J. ACM 42 (4) (1995) 857–907, https://doi.org/10.1145/210332.210339.
[21] Decheng Dai, Rong Ge, Another sub-exponential algorithm for the simple stochastic game, Algorithmica 61 (4) (2011) 1092–1104, https://doi.org/10.1007/s00453-010-9413-1.
[22] Peng Dai Mausam, Daniel S. Weld, Judy Goldsmith, Topological value iteration algorithms, J. Artif. Intell. Res. 42 (2011) 181–209, Available at, http://jair.org/papers/paper3390.html.
[23] J. Filar, K. Vrieze, Competitive Markov Decision Processes, Springer-Verlag, 1997.
[24] Hugo Gimbert, Florian Horn, Simple stochastic games with few random vertices are easy to solve, in: FoSSaCS, in: Lecture Notes in Computer Science, vol. 4962, Springer, 2008, pp. 5–19.
[25] Serge Haddad, Benjamin Monmege, Interval iteration algorithm for MDPs and IMDPs, Theor. Comput. Sci. 735 (2018) 111–131, https://doi.org/10.1016/j.tcs.2016.12.003.
[26] Ernst Moritz Hahn, Arnd Hartmanns, Christian Hensel, Michaela Klauck, Joachim Klein, Jan Kretínský, David Parker, Tim Quatmann, Enno Ruijters, Marcel Steinmetz, The 2019 comparison of tools for the analysis of quantitative formal models - (QComp 2019 competition report), in: TACAS (3), in: Lecture Notes in Computer Science, vol. 11429, Springer, 2019, pp. 69–92.
[27] Kristoffer Arnsfelt Hansen, Rasmus Ibsen-Jensen, Peter Bro Miltersen, The complexity of solving reachability games using value and strategy iteration, Theory Comput. Syst. 55 (2) (2014) 380–403, https://doi.org/10.1007/s00224-013-9524-6.
[28] Arnd Hartmanns, Benjamin Lucien Kaminski, Optimistic value iteration, in: CAV (2), in: Lecture Notes in Computer Science, vol. 12225, Springer, 2020, pp. 488–511.
[29] A.J. Hoffman, R.M. Karp, On nonterminating stochastic games, Manag. Sci. 12 (5) (1966) 359–370, https://doi.org/10.1287/mnsc.12.5.359.
[30] Rasmus Ibsen-Jensen, Peter Bro Miltersen, Solving simple stochastic games with few coin toss positions, in: ESA, in: Lecture Notes in Computer Science, vol. 7501, Springer, 2012, pp. 636–647.
[31] Mark Kattenbelt, Marta Z. Kwiatkowska, Gethin Norman, David Parker, A game-based abstraction-refinement framework for Markov decision processes, Form. Methods Syst. Des. 36 (3) (2010) 246–280, https://doi.org/10.1007/s10703-010-0097-6.
[32] Edon Kelmendi, Julia Krämer, Jan Kretínský, Maximilian Weininger, Value iteration for simple stochastic games: stopping criterion and learning algorithm, in: CAV (1), in: Lecture Notes in Computer Science, vol. 10981, Springer, 2018, pp. 623–642.
[33] Mikhail K. Kozlov, Sergei P. Tarasov, Leonid G. Khachiyan, The polynomial solvability of convex quadratic programming, USSR Comput. Math. Math. Phys. 20 (5) (1980) 223–228, https://doi.org/10.1016/0041-5553(80)90098-1.
[34] Jan Kretínský, Tobias Meggendorfer, Efficient strategy iteration for mean payoff in Markov decision processes, in: ATVA, in: Lecture Notes in Computer Science, vol. 10482, Springer, 2017, pp. 380–399.
[35] Jan Kretínský, Tobias Meggendorfer, Of cores: a partial-exploration framework for Markov decision processes, in: CONCUR, in: LIPIcs, vol. 140, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 5.
[36] Marta Kwiatkowska, Gethin Norman, David Parker, Gabriel Santos, PRISM-games 3.0: stochastic game verification with concurrency, equilibria and time, in: CAV (2), in: Lecture Notes in Computer Science, vol. 12225, Springer, 2020, pp. 475–487.
[37] Jan Křetínský, Emanuel Ramneantu, Alexander Slivinskiy, Maximilian Weininger, Comparison of algorithms for simple stochastic games, Electr. Proc. Theor. Comput. Sci. 326 (2020) 131–148, https://doi.org/10.4204/eptcs.326.9.
[38] Ludwig Walter, A subexponential randomized algorithm for the simple stochastic game problem, Inf. Comput. 117 (1) (1995) 151–155, https://doi.org/10.1006/inco.1995.1035.
[39] Kittiphon Phalakarn, Toru Takisaka, Thomas Haas, Ichiro Hasuo, Widest paths and global propagation in bounded value iteration for stochastic games, in: CAV (2), in: Lecture Notes in Computer Science, vol. 12225, Springer, 2020, pp. 349–371.
[40] Martin L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, Wiley Series in Probability and Statistics, Wiley, 1994.
[41] Tim Quatmann, Joost-Pieter Katoen, Sound value iteration, in: CAV (1), in: Lecture Notes in Computer Science, vol. 10981, Springer, 2018, pp. 643–661.
[42] Sartaj Sahni, Computationally related problems, SIAM J. Comput. 3 (4) (1974) 262–279, https://doi.org/10.1137/0203021.
[43] Rafal Somla, New algorithms for solving simple stochastic games, Electron. Notes Theor. Comput. Sci. 119 (1) (2005) 51–65, https://doi.org/10.1016/j.entcs.2004.07.008.
[44] María Svorenová, Marta Kwiatkowska, Quantitative verification and strategy synthesis for stochastic games, Eur. J. Control 30 (2016) 15–30, https://doi.org/10.1016/j.ejcon.2016.04.009.
[45] Maximilian Weininger, Tobias Meggendorfer, Jan Kretínský, Satisfiability bounds for ω-regular properties in bounded-parameter Markov decision processes, in: CDC, IEEE, 2019, pp. 2284–2291.
[46] Uri Zwick, Mike Paterson, The complexity of mean payoff games on graphs, Theor. Comput. Sci. 158 (1,2) (1996) 343–359, https://doi.org/10.1016/0304-3975(95)00188-3.

C PAC Statistical Model Checking for Markov Decision Processes and Stochastic Games (CAV 2019)

This is an exact reprinting of the paper which has been published as a **peer reviewed conference paper**.

Pranav Ashok, Jan Kretinsky, and Maximilian Weininger. "PAC Statistical Model Checking for Markov Decision Processes and Stochastic Games". In: CAV (1). Vol. 11561. Lecture Notes in Computer Science. Springer, 2019, pp. 497–519. doi: 10.1007/978-3-030-25540-4_29

Summary

Statistical model checking (SMC) is a technique for analysis of probabilistic systems that may be (partially) unknown. We present an SMC algorithm for (unbounded) reachability yielding probably approximately correct (PAC) guarantees on the results. We consider both the setting (i) with no knowledge of the transition function (with the only quantity required a bound on the minimum transition probability) and (ii) with knowledge of the topology of the underlying graph. On the one hand, it is the first algorithm for stochastic games. On the other hand, it is the first practical algorithm even for Markov decision processes. Compared to previous approaches where PAC guarantees require running times longer than the age of universe even for systems with a handful of states, our algorithm often yields reasonably precise results within minutes, not requiring the knowledge of mixing time.

For more details on this publication, we refer to Section 4.1 of the main body.

Contribution

Contribution of Maximilian Weininger	
Development and conceptual design of the research project	50%
Discussion and development of ideas	50%
Composition and revision of the manuscript	75%
Proofs	75%
Implementation	50%
Experimental evaluation	15%

PAC Statistical Model Checking for Markov Decision Processes and Stochastic Games

Pranav Ashok, Jan Křetínský, and Maximilian Weininger( )

Technical University of Munich, Munich, Germany
maxi.weininger@tum.de

Abstract. Statistical model checking (SMC) is a technique for analysis of probabilistic systems that may be (partially) unknown. We present an SMC algorithm for (unbounded) reachability yielding probably approximately correct (PAC) guarantees on the results. We consider both the setting (i) with no knowledge of the transition function (with the only quantity required a bound on the minimum transition probability) and (ii) with knowledge of the topology of the underlying graph. On the one hand, it is the first algorithm for stochastic games. On the other hand, it is the first practical algorithm even for Markov decision processes. Compared to previous approaches where PAC guarantees require running times longer than the age of universe even for systems with a handful of states, our algorithm often yields reasonably precise results within minutes, not requiring the knowledge of mixing time.

1 Introduction

Statistical model checking (SMC) [YS02a] is an analysis technique for probabilistic systems based on

1. simulating finitely many finitely long runs of the system,
2. statistical analysis of the obtained results,
3. yielding a confidence interval/probably approximately correct (PAC) result on the probability of satisfying a given property, i.e., there is a non-zero probability that the bounds are incorrect, but they are correct with probability that can be set arbitrarily close to 1.

One of the advantages is that it can avoid the state-space explosion problem, albeit at the cost of weaker guarantees. Even more importantly, this technique is applicable even when the model is not known (*black-box* setting) or only

This research was funded in part by TUM IGSSE Grant 10.06 (PARSEC), the Czech Science Foundation grant No. 18-11193S, and the German Research Foundation (DFG) project KR 4890/2-1 "Statistical Unbounded Verification".

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 497–519, 2019.
https://doi.org/10.1007/978-3-030-25540-4_29

qualitatively known (*grey-box* setting), where the exact transition probabilities are unknown such as in many cyber-physical systems.

In the basic setting of Markov chains [Nor98] with (time- or step-)bounded properties, the technique is very efficient and has been applied to numerous domains, e.g. biological [JCL+09,PGL+13], hybrid [ZPC10,DDL+12,EGF12, Lar12] or cyber-physical [BBB+10,CZ11,DDL+13] systems and a substantial tool support is available [JLS12,BDL+12,BCLS13,BHH12]. In contrast, whenever either (i) infinite time-horizon properties, e.g. reachability, are considered or (ii) non-determinism is present in the system, providing any guarantees becomes significantly harder.

Firstly, for *infinite time-horizon properties* we need a stopping criterion such that the infinite-horizon property can be reliably evaluated based on a finite prefix of the run yielded by simulation. This can rely on the the complete knowledge of the system (*white-box* setting) [YCZ10,LP08], the topology of the system (grey box) [YCZ10,HJB+10], or a lower bound $p_{\min}$ on the minimum transition probability in the system (black box) [DHKP16,BCC+14].

Secondly, for Markov decision processes (MDP) [Put14] with (non-trivial) *non-determinism*, [HMZ+12] and [LP12] employ reinforcement learning [SB98] in the setting of bounded properties or discounted (and for the purposes of approximation thus also bounded) properties, respectively. The latter also yields PAC guarantees.

Finally, for MDP with unbounded properties, [BFHH11] deals with MDP with spurious non-determinism, where the way it is resolved does not affect the desired property. The general non-deterministic case is treated in [FT14, BCC+14], yielding PAC guarantees. However, the former requires the knowledge of mixing time, which is at least as hard to compute; the algorithm in the latter is purely theoretical since before a single value is updated in the learning process, one has to simulate longer than the age of universe even for a system as simple as a Markov chain with 12 states having at least 4 successors for some state.

Our contribution is an SMC algorithm with PAC guarantees for (i) MDP and unbounded properties, which runs for realistic benchmarks [HKP+19] and confidence intervals in orders of minutes, and (ii) is the first algorithm for stochastic games (SG). It relies on different techniques from literature.

1. The increased practical performance rests on two pillars:
   - extending early detection of bottom strongly connected components in Markov chains by [DHKP16] to end components for MDP and simple end components for SG;
   - improving the underlying PAC Q-learning technique of [SLW+06]:
     (a) learning is now model-based with better information reuse instead of model-free, but in realistic settings with the same memory requirements,
     (b) better guidance of learning due to interleaving with precise computation, which yields more precise value estimates.
     (c) splitting confidence over all relevant transitions, allowing for variable width of confidence intervals on the learnt transition probabilities.

2. The transition from algorithms for MDP to SG is possible via extending the over-approximating value iteration from MDP [BCC+14] to SG by [KKKW18].

To summarize, we give an anytime PAC SMC algorithm for (unbounded) reachability. It is the first such algorithm for SG and the first practical one for MDP.

Related Work

Most of the previous efforts in SMC have focused on the analysis of properties with *bounded* horizon [YS02a,SVA04,YKNP06,JCL+09,JLS12,BDL+12].

SMC of *unbounded* properties was first considered in [HLMP04] and the first approach was proposed in [SVA05], but observed incorrect in [HJB+10]. Notably, in [YCZ10] two approaches are described. The first approach proposes to terminate sampled paths at every step with some probability p_{term} and reweight the result accordingly. In order to guarantee the asymptotic convergence of this method, the second eigenvalue λ of the chain and its mixing time must be computed, which is as hard as the verification problem itself and requires the complete knowledge of the system (white box setting). The correctness of [LP08] relies on the knowledge of the second eigenvalue λ, too. The second approach of [YCZ10] requires the knowledge of the chain's topology (grey box), which is used to transform the chain so that all potentially infinite paths are eliminated. In [HJB+10], a similar transformation is performed, again requiring knowledge of the topology. In [DHKP16], only (a lower bound on) the minimum transition probability $p_{\min}$ is assumed and PAC guarantees are derived. While unbounded properties cannot be analyzed without any information on the system, knowledge of $p_{\min}$ is a relatively light assumption in many realistic scenarios [DHKP16]. For instance, bounds on the rates for reaction kinetics in chemical reaction systems are typically known; for models in the PRISM language [KNP11], the bounds can be easily inferred without constructing the respective state space. In this paper, we thus adopt this assumption.

In the case with general *non-determinism*, one approach is to give the nondeterminism a probabilistic semantics, e.g., using a uniform distribution instead, as for timed automata in [DLL+11a,DLL+11b,Lar13]. Others [LP12,HMZ+12, BCC+14] aim to quantify over all strategies and produce an ϵ-optimal strategy. In [HMZ+12], candidates for optimal strategies are generated and gradually improved, but "at any given point we cannot quantify how close to optimal the candidate scheduler is" (cited from [HMZ+12]) and the algorithm "does not in general converge to the true optimum" (cited from [LST14]). Further, [LST14,DLST15,DHS18] randomly sample compact representation of strategies, resulting in useful lower bounds if ε-schedulers are frequent. [HPS+19] gives a convergent model-free algorithm (with no bounds on the current error) and identifies that the previous [SKC+14] "has two faults, the second of which also affects approaches [...] [HAK18,HAK19]".

Several approaches provide SMC for MDPs and unbounded properties with *PAC guarantees*. Firstly, similarly to [LP08,YCZ10], [FT14] requires (1) the mixing time T of the MDP. The algorithm then yields PAC bounds in time polynomial in T (which in turn can of course be exponential in the size of the MDP). Moreover, the algorithm requires (2) the ability to restart simulations also in non-initial states, (3) it only returns the strategy once all states have been visited (sufficiently many times), and thus (4) requires the size of the state space $|S|$. Secondly, [BCC+14], based on delayed Q-learning (DQL) [SLW+06], lifts the assumptions (2) and (3) and instead of (1) mixing time requires only (a bound on) the minimum transition probability $p_{\min}$. Our approach additionally lifts the assumption (4) and allows for running times faster than those given by T, even without the knowledge of T.

Reinforcement learning (without PAC bounds) for stochastic games has been considered already in [LN81,Lit94,BT99]. [WT16] combines the special case of almost-sure satisfaction of a specification with optimizing quantitative objectives. We use techniques of [KKKW18], which however assumes access to the transition probabilities.

2 Preliminaries

2.1 Stochastic Games

A *probability distribution* on a finite set X is a mapping $\delta : X \to [0,1]$, such that $\sum_{x \in X} \delta(x) = 1$. The set of all probability distributions on X is denoted by $\mathcal{D}(X)$. Now we define turn-based two-player stochastic games. As opposed to the notation of e.g. [Con92], we do not have special stochastic nodes, but rather a probabilistic transition function.

Definition 1 (SG). *A stochastic game (SG) is a tuple* $\mathsf{G} = (\mathsf{S}, \mathsf{S}_\square, \mathsf{S}_\bigcirc, \mathsf{s}_0, \mathsf{A}, \mathsf{Av}, \mathbb{T})$, where S is a finite set of *states* partitioned[1] into the sets $\mathsf{S}_\square$ and $\mathsf{S}_\bigcirc$ of states of the player *Maximizer* and *Minimizer*[2], respectively $\mathsf{s}_0 \in \mathsf{S}$ is the *initial* state, A is a finite set of *actions*, $\mathsf{Av} : \mathsf{S} \to 2^{\mathsf{A}}$ assigns to every state a set of *available* actions, and $\mathbb{T} : \mathsf{S} \times \mathsf{A} \to \mathcal{D}(\mathsf{S})$ is a *transition function* that given a state s and an action $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$ yields a probability distribution over *successor* states. Note that for ease of notation we write $\mathbb{T}(\mathsf{s}, \mathsf{a}, \mathsf{t})$ instead of $\mathbb{T}(\mathsf{s}, \mathsf{a})(\mathsf{t})$.

A Markov decision process (MDP) is a special case of SG where $\mathsf{S}_\bigcirc = \emptyset$. A Markov chain (MC) can be seen as a special case of an MDP, where for all $\mathsf{s} \in \mathsf{S} : |\mathsf{Av}(\mathsf{s})| = 1$. We assume that SG are non-blocking, so for all states s we have $\mathsf{Av}(\mathsf{s}) \neq \emptyset$.

For a state s and an available action $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$, we denote the set of successors by $\mathsf{Post}(\mathsf{s}, \mathsf{a}) := \{\mathsf{t} \mid \mathbb{T}(\mathsf{s}, \mathsf{a}, \mathsf{t}) > 0\}$. We say a state-action pair (s, a) is an *exit* of a set of states T, written $(\mathsf{s}, \mathsf{a}) \,\mathrm{exits}\, T$, if $\exists \mathsf{t} \in \mathsf{Post}(\mathsf{s}, \mathsf{a}) : \mathsf{t} \notin T$, i.e., if with some probability a successor outside of T could be chosen.

We consider algorithms that have a limited information about the SG.

[1] I.e., $\mathsf{S}_\square \subseteq \mathsf{S}$, $\mathsf{S}_\bigcirc \subseteq \mathsf{S}$, $\mathsf{S}_\square \cup \mathsf{S}_\bigcirc = \mathsf{S}$, and $\mathsf{S}_\square \cap \mathsf{S}_\bigcirc = \emptyset$.

[2] The names are chosen, because Maximizer maximizes the probability of reaching a given target state, and Minimizer minimizes it.

Definition 2 (Black box and grey box). *An algorithm inputs an SG as* black box *if it cannot access the whole tuple, but*

- *it knows the initial state,*
- *for a given state, an oracle returns its player and available action,*
- *given a state* s *and action* a*, it can sample a successor* t *according to* $\mathbb{T}(\mathsf{s},\mathsf{a})$*,*[3]
- *it knows* $p_{\min} \le \min_{\substack{\mathsf{s}\in\mathsf{S},\mathsf{a}\in\mathsf{Av}(\mathsf{s})\\ \mathsf{t}\in\mathsf{Post}(\mathsf{s},\mathsf{a})}} \mathbb{T}(\mathsf{s},\mathsf{a},\mathsf{t})$*, an under-approximation of the minimum transition probability.*

When input as grey box *it additionally knows the number* $|\mathsf{Post}(\mathsf{s},\mathsf{a})|$ *of successors for each state* s *and action* a*.*[4]

The semantics of SG is given in the usual way by means of strategies and the induced Markov chain [BK08] and its respective probability space, as follows. An *infinite path* ρ is an infinite sequence $\rho = \mathsf{s}_0\mathsf{a}_0\mathsf{s}_1\mathsf{a}_1\cdots \in (\mathsf{S}\times\mathsf{A})^\omega$, such that for every $i \in \mathbb{N}$, $\mathsf{a}_i \in \mathsf{Av}(\mathsf{s}_i)$ and $\mathsf{s}_{i+1} \in \mathsf{Post}(\mathsf{s}_i,\mathsf{a}_i)$.

A *strategy* of Maximizer or Minimizer is a function $\sigma : \mathsf{S}_\Box \to \mathcal{D}(\mathsf{A})$ or $\mathsf{S}_\bigcirc \to \mathcal{D}(\mathsf{A})$, respectively, such that $\sigma(\mathsf{s}) \in \mathcal{D}(\mathsf{Av}(\mathsf{s}))$ for all s. Note that we restrict to memoryless/positional strategies, as they suffice for reachability in SGs [CH12].

A pair (σ,τ) of strategies of Maximizer and Minimizer induces a Markov chain $\mathsf{G}^{\sigma,\tau}$ with states S, s_0 being initial, and the transition function $\mathbb{T}(\mathsf{s})(\mathsf{t}) = \sum_{\mathsf{a}\in\mathsf{Av}(\mathsf{s})} \sigma(\mathsf{s})(\mathsf{a})\cdot\mathbb{T}(\mathsf{s},\mathsf{a},\mathsf{t})$ for states of Maximizer and analogously for states of Minimizer, with σ replaced by τ. The Markov chain induces a unique probability distribution $\mathbb{P}^{\sigma,\tau}$ over measurable sets of infinite paths [BK08, Ch. 10].

2.2 Reachability Objective

For a goal set $\mathsf{Goal} \subseteq \mathsf{S}$, we write $\Diamond\mathsf{Goal} := \{\mathsf{s}_0\mathsf{a}_0\mathsf{s}_1\mathsf{a}_1\cdots \mid \exists i\in\mathbb{N} : \mathsf{s}_i \in \mathsf{Goal}\}$ to denote the (measurable) set of all infinite paths which eventually reach Goal. For each $\mathsf{s}\in\mathsf{S}$, we define the *value* in s as

$$\mathsf{V}(\mathsf{s}) := \sup_\sigma \inf_\tau \mathbb{P}^{\sigma,\tau}_s(\Diamond\mathsf{Goal}) = \inf_\tau \sup_\sigma \mathbb{P}^{\sigma,\tau}_s(\Diamond\mathsf{Goal}),$$

where the equality follows from [Mar75]. We are interested in $\mathsf{V}(\mathsf{s}_0)$, its ε-approximation and the corresponding (ε-)optimal strategies for both players.

[3] Up to this point, this definition conforms to black box systems in the sense of [SVA04] with sampling from the initial state, being slightly stricter than [YS02a] or [RP09], where simulations can be run from any desired state. Further, we assume that we can choose actions for the adversarial player or that she plays fairly. Otherwise the adversary could avoid playing her best strategy during the SMC, not giving SMC enough information about her possible behaviours.

[4] This requirement is slightly weaker than the knowledge of the whole topology, i.e. $\mathsf{Post}(\mathsf{s},\mathsf{a})$ for each s and a.

Let Zero be the set of states, from which there is no finite path to any state in Goal. The value function V satisfies the following system of equations, which is referred to as the *Bellman equations*:

$$\mathsf{V}(\mathsf{s}) = \begin{cases} \max_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{V}(\mathsf{s}, \mathsf{a}) & \text{if } \mathsf{s} \in \mathsf{S}_{\Box} \\ \min_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \mathsf{V}(\mathsf{s}, \mathsf{a}) & \text{if } \mathsf{s} \in \mathsf{S}_{\bigcirc} \\ 1 & \text{if } \mathsf{s} \in \mathsf{Goal} \\ 0 & \text{if } \mathsf{s} \in \mathsf{Zero} \end{cases}$$

with the abbreviation $\mathsf{V}(\mathsf{s}, \mathsf{a}) := \sum_{s' \in S} \mathbb{T}(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{V}(\mathsf{s}')$. Moreover, V is the *least* solution to the Bellman equations, see e.g. [CH08].

2.3 Bounded and Asynchronous Value Iteration

The well known technique of value iteration, e.g. [Put14,RF91], works by starting from an under-approximation of value function and then applying the Bellman equations. This converges towards the least fixpoint of the Bellman equations, i.e. the *value function*. Since it is difficult to give a convergence criterion, the approach of bounded value iteration (BVI, also called interval iteration) was developed for MDP [BCC+14,HM17] and SG [KKKW18]. Beside the under-approximation, it also updates an over-approximation according to the Bellman equations. The most conservative over-approximation is to use an upper bound of 1 for every state. For the under-approximation, we can set the lower bound of target states to 1; all other states have a lower bound of 0. We use the function $\mathsf{INITIALIZE\_BOUNDS}$ in our algorithms to denote that the lower and upper bounds are set as just described; see [AKW19, Algorithm 8] for the pseudocode. Additionally, BVI ensures that the over-approximation converges to the least fixpoint by taking special care of *end components*, which are the reason for not converging to the true value from above.

Definition 3 (End component (EC)). *A non-empty set $T \subseteq \mathsf{S}$ of states is an* end component (EC) *if there is a non-empty set $B \subseteq \bigcup_{\mathsf{s} \in T} \mathsf{Av}(s)$ of actions such that (i) for each $\mathsf{s} \in T, \mathsf{a} \in B \cap \mathsf{Av}(\mathsf{s})$ we do* not *have* (s, a) exits T *and (ii) for each $\mathsf{s}, \mathsf{s}' \in T$ there is a finite path $\mathsf{w} = \mathsf{s}\mathsf{a}_0 \ldots \mathsf{a}_n\mathsf{s}' \in (T \times B)^* \times T$, i.e. the path stays inside T and only uses actions in B.*

Intuitively, ECs correspond to bottom strongly connected components of the Markov chains induced by possible strategies, so for some pair of strategies all possible paths starting in the EC remain there. An end component T is a *maximal end component (MEC)* if there is no other end component T' such that $T \subseteq T'$. Given an SG G, the set of its MECs is denoted by $\mathsf{MEC}(\mathsf{G})$.

Note that, to stay in an EC in an SG, the two players would have to cooperate, since it depends on the pair of strategies. To take into account the adversarial behaviour of the players, it is also relevant to look at a subclass of ECs, the so called *simple end components*, introduced in [KKKW18].

Definition 4 (Simple end component (SEC) [KKKW18]). *An EC T is called* simple*, if for all* $\mathsf{s} \in T$ *it holds that* $\mathsf{V}(\mathsf{s}) = \mathsf{bestExit}(T, \mathsf{V})$*, where*

$$\mathsf{bestExit}(T, f) := \begin{cases} 1 & \text{if } T \cap \mathsf{Goal} \neq \emptyset \\ \max_{\substack{\mathsf{s} \in T \cap \mathsf{S}_\square \\ (\mathsf{s},\mathsf{a}) \text{ exits } T}} f(\mathsf{s}, \mathsf{a}) & \text{else} \end{cases}$$

is called the best exit *(of Maximizer) from* T *according to the function* $f : \mathsf{S} \to \mathbb{R}$*. To handle the case that there is no exit of Maximizer in* T *we set* $\max_\emptyset = 0$.

Intuitively, SECs are ECs where Minimizer does not want to use any of her exits, as all of them have a greater value than the best exit of Maximizer. Assigning any value between those of the best exits of Maximizer and Minimizer to all states in the EC is a solution to the Bellman equations, because both players prefer remaining and getting that value to using their exits [KKKW18, Lemma 1]. However, this is suboptimal for Maximizer, as the goal is not reached if the game remains in the EC forever. Hence we "deflate" the upper bounds of SECs, i.e. reduce them to depend on the best exit of Maximizer. T is called maximal simple end component (MSEC), if there is no SEC T' such that $T \subsetneq T'$. Note that in MDPs, treating all MSECs amounts to treating all MECs.

Algorithm 1. Bounded value iteration algorithm for SG (and MDP)

```
1: procedure BVI(SG $\mathsf{G}$, target set $\mathsf{Goal}$, precision $\epsilon > 0$)
2:     INITIALIZE_BOUNDS
3:     repeat
4:         $X \leftarrow$ SIMULATE until LOOPING or state in Goal is hit
5:         UPDATE($X$)                          ▷ Bellman updates or their modification
6:         for $T \in$ FIND_MSECs($X$) do
7:             DEFLATE($T$)                     ▷ Decrease the upper bound of MSECs
8:     until $\mathsf{U}(\mathsf{s}_0) - \mathsf{L}(\mathsf{s}_0) < \epsilon$
```

Algorithm 1 rephrases that of [KKKW18] and describes the general structure of all bounded value iteration algorithms that are relevant for this paper. We discuss it here since all our improvements refer to functions (in capitalized font) in it. In the next section, we design new functions, pinpointing the difference to the other papers. The pseudocode of the functions adapted from the other papers can be found, for the reader's convenience, in [AKW19, Appendix A]. Note that to improve readability, we omit the parameters $\mathsf{G}, \mathsf{Goal}, \mathsf{L}$ and U of the functions in the algorithm.

Bounded Value Iteration: For the standard bounded value iteration algorithm, Line 4 does not run a simulation, but just assigns the whole state space S to X[5]. Then it updates all values according to the Bellman equations.

[5] Since we mainly talk about simulation based algorithms, we included this line to make their structure clearer.

After that it finds all the problematic components, the MSECs, and "deflates" them as described in [KKKW18], i.e. it reduces their values to ensure the convergence to the least fixpoint. This suffices for the bounds to converge and the algorithm to terminate [KKKW18, Theorem 2].

Asynchronous Bounded Value Iteration: To tackle the state space explosion problem, *asynchronous* simulation/learning-based algorithms have been developed [MLG05,BCC+14,KKKW18]. The idea is not to update and deflate all states at once, since there might be too many, or since we only have limited information. Instead of considering the whole state space, a path through the SG is sampled by picking in every state one of the actions that look optimal according to the current over-/under-approximation and then sampling a successor of that action. This is repeated until either a target is found, or until the simulation is looping in an EC; the latter case occurs if the heuristic that picks the actions generates a pair of strategies under which both players only pick staying actions in an EC. After the simulation, only the bounds of the states on the path are updated and deflated. Since we pick actions which look optimal in the simulation, we almost surely find an ϵ-optimal strategy and the algorithm terminates [BCC+14, Theorem 3].

3 Algorithm

3.1 Model-Based

Given only limited information, updating cannot be done using $\mathbb{T}$, since the true probabilities are not known. The approach of [BCC+14] is to sample for a high number of steps and accumulate the observed lower and upper bounds on the true value function for each state-action pair. When the number of samples is large enough, the average of the accumulator is used as the new estimate for the state-action pair, and thus the approximations can be improved and the results back-propagated, while giving statistical guarantees that each update was correct. However, this approach has several drawbacks, the biggest of which is that the number of steps before an update can occur is infeasibly large, often larger than the age of the universe, see Table 1 in Sect. 4.

Our improvements to make the algorithm practically usable are linked to constructing a partial model of the given system. That way, we have more information available on which we can base our estimates, and we can be less conservative when giving bounds on the possible errors. The shift from model-free to model-based learning asymptotically increases the memory requirements from $\mathcal{O}(|\mathsf{S}| \cdot |\mathsf{A}|)$ (as in [SLW+06,BCC+14]) to $\mathcal{O}(|\mathsf{S}|^2 \cdot |\mathsf{A}|)$. However, for systems where each action has a small constant bound on the number of successors, which is typical for many practical systems, e.g. classical PRISM benchmarks, it is still $\mathcal{O}(|\mathsf{S}| \cdot |\mathsf{A}|)$ with a negligible constant difference.

We thus track the number of times some successor t has been observed when playing action a from state s in a variable $\#(\mathsf{s},\mathsf{a},\mathsf{t})$. This implicitly induces the number of times each state-action pair (s,a) has been played $\#(\mathsf{s},\mathsf{a}) =$

$\sum_{\mathsf{t}\in\mathsf{S}} \#(\mathsf{s},\mathsf{a},\mathsf{t})$. Given these numbers we can then calculate probability estimates for every transition as described in the next subsection. They also induce the set of all states visited so far, allowing us to construct a partial model of the game. See [AKW19, Appendix A.2] for the pseudo-code of how to count the occurrences during the simulations.

3.2 Safe Updates with Confidence Intervals Using Distributed Error Probability

We use the counters to compute a lower estimate of the transition probability for some error tolerance $\delta_{\mathbb{T}}$ as follows: We view sampling t from state-action pair (s,a) as a Bernoulli sequence, with success probability $\mathbb{T}(\mathsf{s},\mathsf{a},\mathsf{t})$, the number of trials $\#(\mathsf{s},\mathsf{a})$ and the number of successes $\#(\mathsf{s},\mathsf{a},\mathsf{t})$. The tightest lower estimate we can give using the Hoeffding bound (see [AKW19, Appendix D.1]) is

$$\widehat{\mathbb{T}}(\mathsf{s},\mathsf{a},\mathsf{t}) := \max(0, \frac{\#(s,a,t)}{\#(s,a)} - c), \tag{1}$$

where the confidence width $c := \sqrt{\frac{\ln(\delta_{\mathbb{T}})}{-2\#(\mathsf{s},\mathsf{a})}}$. Since c could be greater than 1, we limit the lower estimate to be at least 0. Now we can give modified update equations:

$$\widehat{\mathsf{L}}(\mathsf{s},\mathsf{a}) := \sum_{\mathsf{t}:\#(\mathsf{s},\mathsf{a},\mathsf{t})>0} \widehat{\mathbb{T}}(\mathsf{s},\mathsf{a},\mathsf{t}) \cdot \mathsf{L}(\mathsf{t})$$

$$\widehat{\mathsf{U}}(\mathsf{s},\mathsf{a}) := \left(\sum_{\mathsf{t}:\#(\mathsf{s},\mathsf{a},\mathsf{t})>0} \widehat{\mathbb{T}}(\mathsf{s},\mathsf{a},\mathsf{t}) \cdot \mathsf{U}(\mathsf{t})\right) + \left(1 - \sum_{\mathsf{t}:\#(\mathsf{s},\mathsf{a},\mathsf{t})>0} \widehat{\mathbb{T}}(\mathsf{s},\mathsf{a},\mathsf{t})\right)$$

The idea is the same for both upper and lower bound: In contrast to the usual Bellman equation (see Sect. 2.2) we use $\widehat{\mathbb{T}}$ instead of $\mathbb{T}$. But since the sum of all the lower estimates does not add up to one, there is some remaining probability for which we need to under-/over-approximate the value it can achieve. We use

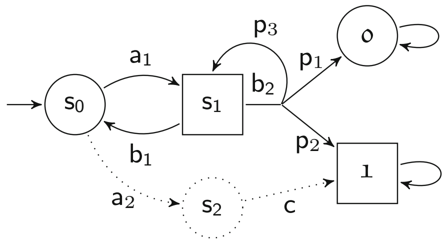


Fig. 1. A running example of an SG. The dashed part is only relevant for the later examples. For actions with only one successor, we do not depict the transition probability 1 (e.g. $\mathbb{T}(\mathsf{s}_0,\mathsf{a}_1,\mathsf{s}_1)$). For state-action pair $(\mathsf{s}_1,\mathsf{b}_2)$, the transition probabilities are parameterized and instantiated in the examples where they are used.

the safe approximations 0 and 1 for the lower and upper bound respectively; this is why in $\widehat{\mathsf{L}}$ there is no second term and in $\widehat{\mathsf{U}}$ the whole remaining probability is added. Algorithm 2 shows the modified update that uses the lower estimates; the proof of its correctness is in [AKW19, Appendix D.2].

Lemma 1 (UPDATE is correct). *Given correct under- and over-approximations* L, U *of the value function* V*, and correct lower probability estimates* $\widehat{\mathbb{T}}$*, the under- and over-approximations after an application of* UPDATE *are also correct.*

Algorithm 2. New update procedure using the probability estimates

```
1: procedure UPDATE(State set $X$)
2:     for $f \in \{\mathsf{L}, \mathsf{U}\}$ do                          ▷ For both functions
3:         for $\mathsf{s} \in X \setminus \mathsf{Goal}$ do      ▷ For all non-target states in the given set
4:             $f(\mathsf{s}) = \begin{cases} \max_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \widehat{f}(\mathsf{s}, \mathsf{a}) & \text{if } \mathsf{s} \in \mathsf{S}_\square \\ \min_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \widehat{f}(\mathsf{s}, \mathsf{a}) & \text{if } \mathsf{s} \in \mathsf{S}_\bigcirc \end{cases}$
```

Example 1. We illustrate how the calculation works and its huge advantage over the approach from [BCC+14] on the SG from Fig. 1. For this example, ignore the dashed part and let $\mathsf{p}_1 = \mathsf{p}_2 = 0.5$, i.e. we have no self loop, and an even chance to go to the target $\mathfrak{1}$ or a sink $\mathfrak{0}$. Observe that hence $\mathsf{V}(\mathsf{s}_0) = \mathsf{V}(\mathsf{s}_1) = 0.5$.

Given an error tolerance of $\delta = 0.1$, the algorithm of [BCC+14] would have to sample for more than 10^9 steps before it could attempt a single update. In contrast, assume we have seen 5 samples of action b_2, where 1 of them went to $\mathfrak{1}$ and 4 of them to $\mathfrak{0}$. Note that, in a sense, we were unlucky here, as the observed averages are very different from the actual distribution. The confidence width for $\delta_{\mathbb{T}} = 0.1$ and 5 samples is $\sqrt{\ln(0.1)/-2 \cdot 5} \approx 0.48$. So given that data, we get $\widehat{\mathbb{T}}(\mathsf{s}_1, \mathsf{b}_2, \mathfrak{1}) = \max(0, 0.2 - 0.48) = 0$ and $\widehat{\mathbb{T}}(\mathsf{s}_1, \mathsf{b}_2, \mathfrak{0}) = \max(0, 0.8 - 0.48) = 0.32$. Note that both probabilities are in fact lower estimates for their true counterpart.

Assume we already found out that $\mathfrak{0}$ is a sink with value 0; how we gain this knowledge is explained in the following subsections. Then, after getting only these 5 samples, UPDATE already decreases the upper bound of $(\mathsf{s}_1, \mathsf{b}_2)$ to 0.68, as we know that at least 0.32 of $\mathbb{T}(\mathsf{s}_1, \mathsf{b}_2)$ goes to the sink.

Given 500 samples of action b_2, the confidence width of the probability estimates already has decreased below 0.05. Then, since we have this confidence width for both the upper and the lower bound, we can decrease the total precision for $(\mathsf{s}_1, \mathsf{b}_2)$ to 0.1, i.e. return an interval in the order of $[0.45; 0.55]$. ◁

Summing up: with the model-based approach we can already start updating after very few steps and get a reasonable level of confidence with a realistic number of samples. In contrast, the state-of-the-art approach of [BCC+14] needs a very large number of samples even for this toy example.

Since for UPDATE we need an error tolerance for every transition, we need to distribute the given total error tolerance δ over all transitions in the current partial model. For all states in the explored partial model $\widehat{\mathsf{S}}$ we know the number of available actions and can over-approximate the number of successors as $\frac{1}{p_{\min}}$. Thus the error tolerance for each transition can be set to $\delta_{\mathbb{T}} := \frac{\delta \cdot p_{\min}}{|\{\mathsf{a} | \mathsf{s} \in \widehat{\mathsf{S}} \wedge \mathsf{a} \in \mathsf{Av}(\mathsf{s})\}|}$. This is illustrated in Example 4 in [AKW19, Appendix B].

Note that the fact that the error tolerance $\delta_{\mathbb{T}}$ for every transition is the same does *not* imply that the confidence width for every transition is the same, as the latter becomes smaller with increasing number of samples $\#(\mathsf{s}, \mathsf{a})$.

3.3 Improved EC Detection

As mentioned in the description of Algorithm 1, we must detect when the simulation is stuck in a bottom EC and looping forever. However, we may also stop simulations that are looping in some EC but still have a possibility to leave it; for a discussion of different heuristics from [BCC+14, KKKW18], see [AKW19, Appendix A.3].

We choose to define LOOPING as follows: Given a candidate for a bottom EC, we continue sampling until we are $\delta_{\mathbb{T}}$-*sure* (i.e. the error probability is smaller than $\delta_{\mathbb{T}}$) that we cannot leave it. Then we can safely deflate the EC, i.e. decrease all upper bounds to zero.

To detect that something is a $\delta_{\mathbb{T}}$-*sure* EC, we do not sample for the astronomical number of steps as in [BCC+14], but rather extend the approach to detect bottom strongly connected components from [DHKP16]. If in the EC-candidate T there was some state-action pair (s, a) that actually has a probability to exit the T, that probability is at least $p_{\min}$. So after sampling (s, a) for n times, the probability to overlook such a leaving transition is $(1 - p_{\min})^n$ and it should be smaller than $\delta_{\mathbb{T}}$. Solving the inequation for the required number of samples n yields $n \geq \frac{\ln(\delta_{\mathbb{T}})}{\ln(1-p_{min})}$.

Algorithm 3 checks that we have seen all staying state-action pairs n times, and hence that we are $\delta_{\mathbb{T}}$-*sure* that T is an EC. Note that we restrict to staying state-action pairs, since the requirement for an EC is only that there exist staying actions, not that all actions stay. We further speed up the EC-detection, because we do not wait for n samples in every simulation, but we use the aggregated counters that are kept over all simulations.

Algorithm 3. Check whether we are $\delta_{\mathbb{T}}$-*sure* that T is an EC

```
1: procedure $\delta_{\mathbb{T}}$-sure EC (State set $T$)
2:     $requiredSamples = \frac{\ln(\delta_{\mathbb{T}})}{\ln(1-p_{\min})}$
3:     $B \leftarrow \{(\mathsf{s}, \mathsf{a}) \mid \mathsf{s} \in T \wedge \neg(\mathsf{s}, \mathsf{a}) \text{ exits } T\}$     ▷ Set of staying state-action pairs
4:     return $\bigwedge_{(\mathsf{s},\mathsf{a}) \in B} \#(\mathsf{s}, \mathsf{a}) > requiredSamples$
```

We stop a simulation, if LOOPING returns true, i.e. under the following three conditions: (i) We have seen the current state before in this simulation ($\mathsf{s} \in X$), i.e. there is a cycle. (ii) This cycle is explainable by an EC T in our current partial model. (iii) We are $\delta_{\mathbb{T}}$-*sure* that T is an EC.

Algorithm 4. Check if we are probably looping and should stop the simulation

```
1: procedure LOOPING(State set $X$, state s)
2:     if $\mathsf{s} \notin X$ then
3:         return false                    ▷ Easy improvement to avoid overhead
4:     return $\exists T \subseteq X. T$ is EC in partial model $\wedge\, \mathsf{s} \in T \wedge \delta_{\mathbb{T}}$-sure $\mathsf{EC}(T)$
```

Example 2. For this example, we again use the SG from Fig. 1 without the dashed part, but this time with $\mathsf{p}_1 = \mathsf{p}_2 = \mathsf{p}_3 = \frac{1}{3}$. Assume the path we simulated is $(\mathsf{s}_0, \mathsf{a}_1, \mathsf{s}_1, \mathsf{b}_2, \mathsf{s}_1)$, i.e. we sampled the self-loop of action b_2. Then $\{\mathsf{s}_1\}$ is a candidate for an EC, because given our current observation it seems possible that we will continue looping there forever. However, we do not stop the simulation here, because we are not yet $\delta_{\mathbb{T}}$-*sure* about this. Given $\delta_{\mathbb{T}} = 0.1$, the required samples for that are 6, since $\frac{\ln(0.1)}{\ln(1-\frac{1}{3})} = 5.6$. With high probability (greater than $(1 - \delta_{\mathbb{T}}) = 0.9$), within these 6 steps we will sample one of the other successors of $(\mathsf{s}_1, \mathsf{b}_2)$ and thus realise that we should not stop the simulation in s_1. If, on the other hand, we are in state o or if in state s_1 the guiding heuristic only picks b_1, then we are in fact looping for more than 6 steps, and hence we stop the simulation. ◁

3.4 Adapting to Games: Deflating MSECs

To extend the algorithm of [BCC+14] to SGs, instead of collapsing problematic ECs we deflate them as in [KKKW18], i.e. given an MSEC, we reduce the upper bound of all states in it to the upper bound of the $\mathsf{bestExit}$ of Maximizer. In contrast to [KKKW18], we cannot use the upper bound of the $\mathsf{bestExit}$ based on the true probability, but only based on our estimates. Algorithm 5 shows how to deflate an MSEC and highlights the difference, namely that we use $\widehat{\mathsf{U}}$ instead of U.

Algorithm 5. Black box algorithm to deflate a set of states

```
1: procedure DEFLATE(State set $X$)
2:     for $\mathsf{s} \in X$ do
3:         $\mathsf{U}(\mathsf{s}) = \min(\mathsf{U}(\mathsf{s}), \mathsf{bestExit}(X, \widehat{\mathsf{U}}))$
```

The remaining question is how to find MSECs. The approach of [KKKW18] is to find MSECs by removing the suboptimal actions of Minimizer according to the current lower bound. Since it converges to the true value function, all

MSECs are eventually found [KKKW18, Lemma 2]. Since Algorithm 6 can only access the SG as a black box, there are two differences: We can only compare our estimates of the lower bound $\widehat{\mathsf{L}}(\mathsf{s},\mathsf{a})$ to find out which actions are suboptimal. Additionally there is the problem that we might overlook an exit from an EC, and hence deflate to some value that is too small; thus we need to check that any state set FIND_MSECs returns is a $\delta_{\mathbb{T}}$-*sure* EC. This is illustrated in Example 3. For a bigger example of how all our functions work together, see Example 5 in [AKW19, Appendix B].

Algorithm 6. Finding MSECs in the game restricted to X for black box setting

```
1: procedure FIND_MSECs(State set $X$)
2:     $suboptAct_{\bigcirc} \leftarrow \{(\mathsf{s}, \{\mathsf{a} \in \mathsf{Av}(\mathsf{s}) \mid \widehat{\mathsf{L}}(\mathsf{s},\mathsf{a}) > \mathsf{L}(\mathsf{s})\} \mid \mathsf{s} \in \mathsf{S}_{\bigcirc} \cap X\}$
3:     $\mathsf{Av}' \leftarrow \mathsf{Av}$ without $suboptAct_{\bigcirc}$
4:     $\mathsf{G}' \leftarrow \mathsf{G}$ restricted to states $X$ and available actions $\mathsf{Av}'$
5:     return $\{T \in \mathsf{MEC}(\mathsf{G}') \mid \delta_{\mathbb{T}}\text{-}sure\ \mathsf{EC}(T)\}$
```

Example 3. For this example, we use the full SG from Fig. 1, including the dashed part, with $\mathsf{p}_1, \mathsf{p}_2 > 0$. Let $(\mathsf{s}_0, \mathsf{a}_1, \mathsf{s}_1, \mathsf{b}_2, \mathsf{s}_2, \mathsf{b}_1, \mathsf{s}_1, \mathsf{a}_2, \mathsf{s}_2, \mathsf{c}, \mathsf{1})$ be the path generated by our simulation. Then in our partial view of the model, it seems as if $T = \{\mathsf{s}_0, \mathsf{s}_1\}$ is an MSEC, since using a_2 is suboptimal for the minimizing state s_0[6] and according to our current knowledge $\mathsf{a}_1, \mathsf{b}_1$ and b_2 all stay inside T. If we deflated T now, all states would get an upper bound of 0, which would be incorrect.

Thus in Algorithm 6 we need to require that T is an EC $\delta_{\mathbb{T}}$-*surely.* This was not satisfied in the example, as the state-action pairs have not been observed the required number of times. Thus we do not deflate T, and our upper bounds stay correct. Having seen $(\mathsf{s}_1, \mathsf{b}_2)$ the required number of times, we probably know that it is exiting T and hence will not make the mistake. $\triangleleft$

3.5 Guidance and Statistical Guarantee

It is difficult to give statistical guarantees for the algorithm we have developed so far (i.e. Algorithm 1 calling the new functions from Sects. 3.2, 3.3 and 3.4). Although we can bound the error of each function, applying them repeatedly can add up the error. Algorithm 7 shows our approach to get statistical guarantees: It interleaves a guided simulation phase (Lines 7–10) with a guaranteed standard bounded value iteration (called BVI phase) that uses our new functions (Lines 11–16).

The simulation phase builds the partial model by exploring states and remembering the counters. In the first iteration of the main loop, it chooses actions randomly. In all further iterations, it is guided by the bounds that the last BVI

[6] For $\delta_{\mathbb{T}} = 0.2$, sampling the path to target once suffices to realize that $\mathsf{L}(\mathsf{s}_0, \mathsf{a}_2) > 0$.

phase computed. After $\mathcal{N}_k$ simulations (see below for a discussion of how to choose $\mathcal{N}_k$), all the gathered information is used to compute one version of the partial model with probability estimates $\widehat{\mathbb{T}}$ for a certain error tolerance δ_k. We can continue with the assumption, that these probability estimates are correct, since it is only violated with a probability smaller than our error tolerance (see below for an explanation of the choice of δ_k). So in our correct partial model, we re-initialize the lower and upper bound (Line 12), and execute a guaranteed standard BVI. If the simulation phase already gathered enough data, i.e. explored the relevant states and sampled the relevant transitions often enough, this BVI achieves a precision smaller than ε in the initial state, and the algorithm terminates. Otherwise we start another simulation phase that is guided by the improved bounds.

Algorithm 7. Full algorithm for black box setting

```
 1: procedure BlackVI(SG $\mathsf{G}$, target set $\mathsf{Goal}$, precision $\varepsilon > 0$, error tolerance $\delta > 0$)
 2:     INITIALIZE_BOUNDS
 3:     $k = 1$                                              ▷ guaranteed BVI counter
 4:     $\widehat{\mathsf{S}} \leftarrow \emptyset$                                          ▷ current partial model
 5:     repeat
 6:         $k \leftarrow 2 \cdot k$
 7:         $\delta_k \leftarrow \frac{\delta}{k}$
    // Guided simulation phase
 8:         for $\mathcal{N}_k$ times do
 9:             $X \leftarrow$ SIMULATE
10:             $\widehat{\mathsf{S}} \leftarrow \widehat{\mathsf{S}} \cup X$
    // Guaranteed BVI phase
11:         $\delta_{\mathbb{T}} \leftarrow \frac{\delta_k \cdot p_{\min}}{\left|\{\mathsf{a} \mid \mathsf{s} \in \widehat{\mathsf{S}} \wedge \mathsf{a} \in \mathsf{Av}(\mathsf{s})\}\right|}$          ▷ Set $\delta_{\mathbb{T}}$ as described in Section 3.2
12:         INITIALIZE_BOUNDS
13:         for $k \cdot \left|\widehat{\mathsf{S}}\right|$ times do
14:             UPDATE($\widehat{\mathsf{S}}$)
15:             for $T \in$ FIND_MSECs($\widehat{\mathsf{S}}$) do
16:                 DEFLATE($T$)
17:     until $\mathsf{U}(\mathsf{s}_0) - \mathsf{L}(\mathsf{s}_0) < \varepsilon$
```

Choice of δ_k: For each of the full BVI phases, we construct a partial model that is correct with probability $(1 - \delta_k)$. To ensure that the sum of these errors is not larger than the specified error tolerance δ, we use the variable k, which is initialised to 1 and doubled in every iteration of the main loop. Hence for the i-th BVI, $k = 2^i$. By setting $\delta_k = \frac{\delta}{k}$, we get that $\sum_{i=1}^{\infty} \delta_k = \sum_{i=1}^{\infty} \frac{\delta}{2^i} = \delta$, and hence the error of all BVI phases does not exceed the specified error tolerance.

When to Stop Each BVI-Phase: The BVI phase might not converge if the probability estimates are not good enough. We increase the number of iterations for each BVI depending on k, because that way we ensure that it eventually is allowed to run long enough to converge. On the other hand, since we always run for finitely many iterations, we also ensure that, if we do not have enough information yet, BVI is eventually stopped. Other stopping criteria could return arbitrarily imprecise results [HM17]. We also multiply with $\left|\widehat{\mathsf{S}}\right|$ to improve the chances of the early BVIs to converge, as that number of iterations ensures that every value has been propagated through the whole model at least once.

Discussion of the Choice of $\mathcal{N}_k$: The number of simulations between the guaranteed BVI phases can be chosen freely; it can be a constant number every time, or any sequence of natural numbers, possibly parameterised by e.g. k, $\left|\widehat{\mathsf{S}}\right|$, ε or any of the parameters of G. The design of particularly efficient choices or learning mechanisms that adjust them on the fly is an interesting task left for future work. We conjecture the answer depends on the given SG and "task" that the user has for the algorithm: E.g. if one just needs a quick general estimate of the behaviour of the model, a smaller choice of $\mathcal{N}_k$ is sensible; if on the other hand a definite precision ε certainly needs to be achieved, a larger choice of $\mathcal{N}_k$ is required.

Theorem 1. *For any choice of sequence for $\mathcal{N}_k$, Algorithm 7 is an anytime algorithm with the following property: When it is stopped, it returns an interval for $\mathsf{V}(\mathsf{s}_0)$ that is PAC*[7] *for the given error tolerance δ and some ε', with $0 \leq \varepsilon' \leq 1$.*

Theorem 1 is the foundation of the practical usability of our algorithm. Given some time frame and some $\mathcal{N}_k$, it calculates an approximation for $\mathsf{V}(\mathsf{s}_0)$ that is probably correct. Note that the precision ε' is independent of the input parameter ε, and could in the worst case be always 1. However, practically it often is good (i.e. close to 0) as seen in the results in Sect. 4. Moreover, in our modified algorithm, we can also give a convergence guarantee as in [BCC+14]. Although mostly out of theoretical interest, in [AKW19, Appendix D.4] we design such a sequence $\mathcal{N}_k$, too. Since this a-priori sequence has to work in the worst case, it depends on an infeasibly large number of simulations.

Theorem 2. *There exists a choice of $\mathcal{N}_k$, such that Algorithm 7 is PAC for any input parameters ε, δ, i.e. it terminates almost surely and returns an interval for $\mathsf{V}(\mathsf{s}_0)$ of width smaller than ε that is correct with probability at least $1 - \delta$.*

[7] Probably Approximately Correct, i.e. with probability greater than $1 - \delta$, the value lies in the returned interval of width ε'.

3.6 Utilizing the Additional Information of Grey Box Input

In this section, we consider the grey box setting, i.e. for every state-action pair (s, a) we additionally know the exact number of successors $|\mathsf{Post}(\mathsf{s}, \mathsf{a})|$. Then we can sample every state-action pair until we have seen all successors, and hence this information amounts to having qualitative information about the transitions, i.e. knowing where the transitions go, but not with which probability.

In that setting, we can improve the EC-detection and the estimated bounds in UPDATE. For EC-detection, note that the whole point of $\delta_{\mathbb{T}}$*-sure* EC is to check whether there are further transitions available; in grey box, we know this and need not depend on statistics. For the bounds, note that the equations for $\widehat{\mathsf{L}}$ and $\widehat{\mathsf{U}}$ both have two parts: The usual Bellman part and the remaining probability multiplied with the most conservative guess of the bound, i.e. 0 and 1. If we know all successors of a state-action pair, we do not have to be as conservative; then we can use $\min_{\mathsf{t} \in \mathsf{Post}(\mathsf{s},\mathsf{a})} \mathsf{L}(\mathsf{t})$ respectively $\max_{\mathsf{t} \in \mathsf{Post}(\mathsf{s},\mathsf{a})} \mathsf{U}(\mathsf{t})$. Both these improvements have huge impact, as demonstrated in Sect. 4. However, of course, they also assume more knowledge about the model.

4 Experimental Evaluation

We implemented the approach as an extension of PRISM-Games [CFK+13a]. 11 MDPs with reachability properties were selected from the Quantitative Verification Benchmark Set [HKP+19]. Further, 4 stochastic games benchmarks from [CKJ12, SS12, CFK+13b, CKPS11] were also selected. We ran the experiments on a 40 core Intel Xeon server running at 2.20 GHz per core and having 252 GB of RAM. The tool however utilised only a single core and 1 GB of memory for the model checking. Each benchmark was ran 10 times with a timeout of 30 min. We ran two versions of Algorithm 7, one with the SG as a black box, the other as a grey box (see Definition 2). We chose $\mathcal{N}_k = 10,000$ for all iterations. The tool stopped either when a precision of 10^{-8} was obtained or after 30 min. In total, 16 different model-property combinations were tried out. The results of the experiment are reported in Table 1.

In the black box setting, we obtained $\varepsilon < 0.1$ on 6 of the benchmarks. 5 benchmarks were 'hard' and the algorithm did not improve the precision below 1. For 4 of them, it did not even finish the first simulation phase. If we decrease $\mathcal{N}_k$, the BVI phase is entered, but still no progress is made.

In the grey box setting, on 14 of 16 benchmarks, it took only 6 min to achieve $\varepsilon < 0.1$. For 8 these, the exact value was found within that time. Less than 50% of the state space was explored in the case of `pacman`, `pneuli-zuck-3`, `rabin-3`, `zeroconf` and `cloud_5`. A precision of $\varepsilon < 0.01$ was achieved on 15/16 benchmarks over a period of 30 min.

Table 1. Achieved precision ε' given by our algorithm in both grey and black box settings after running for a period of 30 min (See the paragraph below Theorem 1 for why we use ε' and not ε). The first set of the models are MDPs and the second set are SGs. '-' indicates that the algorithm did not finish the first simulation phase and hence partial BVI was not called. m is the number of steps required by the DQL algorithm of [BCC+14] before the first update. As this number is very large, we report only $log_{10}(m)$. For comparison, note that the age of the universe is approximately 10^{26} ns; logarithm of number of steps doable in this time is thus in the order of 26.

Model	States	Explored %	Precision		$log_{10}(m)$
		Grey/Black	Grey	Black	
consensus	272	100/100	0.00945	0.171	338
csma-2-2	1,038	93/93	0.00127	0.2851	1,888
firewire	83,153	55/-	0.0057	1	129,430
ij-3	7	100/100	0	0.0017	2,675
ij-10	1,023	100/100	0	0.5407	17
pacman	498	18/47	0.00058	0.0086	1,801
philosophers-3	956	56/21	0	1	2,068
pnueli-zuck-3	2,701	25/71	0	0.0285	5,844
rabin-3	27,766	7/4	0	0.026	110,097
wlan-0	2,954	100/100	0	0.8667	9,947
zeroconf	670	29/27	0.00007	0.0586	5,998
cdmsn	1,240	100/98	0	0.8588	3,807
cloud-5	8,842	49/20	0.00031	0.0487	71,484
mdsm-1	62,245	69/-	0.09625	1	182,517
mdsm-2	62,245	72/-	0.00055	1	182,517
team-form-3	12,476	64/-	0	1	54,095

Figure 2 shows the evolution of the lower and upper bounds in both the grey- and the black box settings for 4 different models. Graphs for the other models as well as more details on the results are in [AKW19, Appendix C].

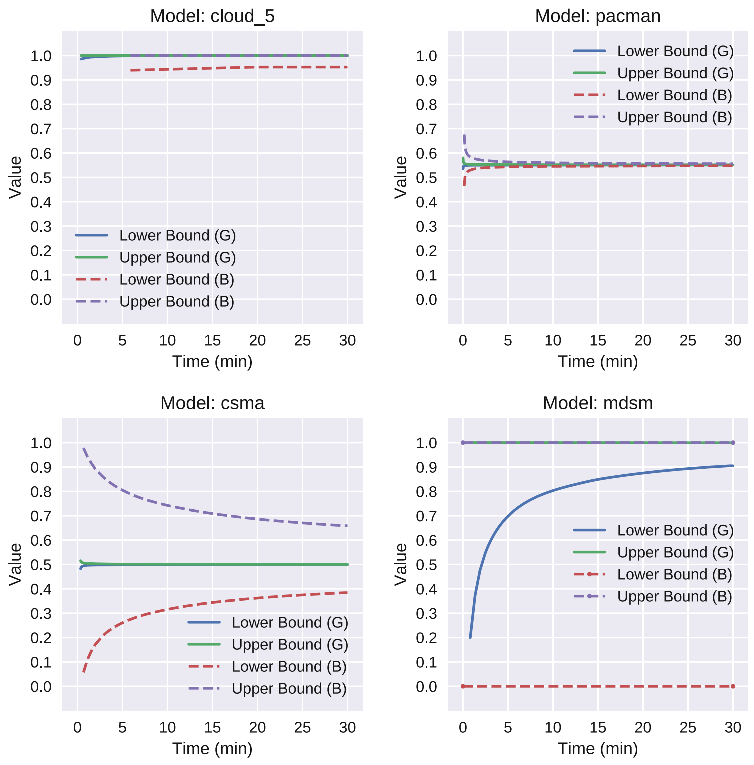


Fig. 2. Performance of our algorithm on various MDP and SG benchmarks in grey and black box settings. Solid lines denote the bounds in the grey box setting while dashed lines denote the bounds in the black box setting. The plotted bounds are obtained after each partial BVI phase, because of which they do not start at $[0,1]$ and not at time 0. Graphs of the remaining benchmarks may be found in [AKW19, Appendix C].

5 Conclusion

We presented a PAC SMC algorithm for SG (and MDP) with the reachability objective. It is the first one for SG and the first practically applicable one. Nevertheless, there are several possible directions for further improvements. For instance, one can consider different sequences for lengths of the simulation phases, possibly also dependent on the behaviour observed so far. Further, the error tolerance could be distributed in a non-uniform way, allowing for fewer visits in rarely visited parts of end components. Since many systems are strongly connected, but at the same time feature some infrequent behaviour, this is the next bottleneck to be attacked. [KM19]

References

[AKW19] Ashok, P., Křetínský, J.: Maximilian Weininger. PAC statistical model checking for markov decision processes and stochastic games. Technical Report arXiv.org/abs/1905.04403 (2019)

[BBB+10] Basu, A., Bensalem, S., Bozga, M., Caillaud, B., Delahaye, B., Legay, A.: Statistical abstraction and model-checking of large heterogeneous systems. In: Hatcliff, J., Zucca, E. (eds.) FMOODS/FORTE 2010. LNCS, vol. 6117, pp. 32–46. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13464-7_4

[BCC+14] Brázdil, T., et al.: Verification of markov decision processes using learning algorithms. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 98–114. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11936-6_8

[BCLS13] Boyer, B., Corre, K., Legay, A., Sedwards, S.: PLASMA-lab: a flexible, distributable statistical model checking library. In: Joshi, K., Siegle, M., Stoelinga, M., DArgenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 160–164. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40196-1_12

[BDL+12] Bulychev, P.E., et al.: UPPAAL-SMC: statistical model checking for priced timed automata. In: QAPL (2012)

[BFHH11] Bogdoll, J., Ferrer Fioriti, L.M., Hartmanns, A., Hermanns, H.: Partial order methods for statistical model checking and simulation. In: Bruni, R., Dingel, J. (eds.) FMOODS/FORTE 2011. LNCS, vol. 6722, pp. 59–74. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21461-5_4

[BHH12] Bogdoll, J., Hartmanns, A., Hermanns, H.: Simulation and statistical model checking for modestly nondeterministic models. In: Schmitt, J.B. (ed.) MMB&DFT 2012. LNCS, vol. 7201, pp. 249–252. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28540-0_20

[BK08] Baier, C., Katoen, J.-P.: Principles of Model Checking. MIT Press (2008). ISBN 978-0-262-02649-9

[BT99] Brafman, R.I., Tennenholtz, M.: A near-optimal poly-time algorithm for learning a class of stochastic games. In: IJCAI, pp. 734–739 (1999)

[CFK+13a] Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: PRISM-games: a model checker for stochastic multi-player games. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 185–191. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36742-7_13

[CFK+13b] Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. Formal Meth. Syst. Des. **43**(1), 61–92 (2013)

[CH08] Chatterjee, K., Henzinger, T.A.: Value iteration. In: Grumberg, O., Veith, H. (eds.) 25 Years of Model Checking. LNCS, vol. 5000, pp. 107–138. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69850-0_7

[CH12] Chatterjee, K., Henzinger, T.A.: A survey of stochastic ω-regular games. J. Comput. Syst. Sci. **78**(2), 394–413 (2012)

[CKJ12] Calinescu, R., Kikuchi, S., Johnson, K.: Compositional reverification of probabilistic safety properties for large-scale complex IT systems. In: Calinescu, R., Garlan, D. (eds.) Monterey Workshop 2012. LNCS, vol. 7539, pp. 303–329. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34059-8_16

[CKPS11] Chen, T., Kwiatkowska, M., Parker, D., Simaitis, A.: Verifying team formation protocols with probabilistic model checking. In: Leite, J., Torroni, P., Ågotnes, T., Boella, G., van der Torre, L. (eds.) CLIMA 2011. LNCS (LNAI), vol. 6814, pp. 190–207. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22359-4_14

[Con92] Condon, A.: The complexity of stochastic games. Inf. Comput. **96**(2), 203–224 (1992)

[CZ11] Clarke, E.M., Zuliani, P.: Statistical model checking for cyber-physical systems. In: ATVA, pp. 1–12 (2011)

[DDL+12] David, A., et al.: Statistical model checking for stochastic hybrid systems. In: HSB, pp. 122–136 (2012)

[DDL+13] David, A., Du, D., Guldstrand Larsen, K., Legay, A., Mikučionis, M.: Optimizing control strategy using statistical model checking. In: Brat, G., Rungta, N., Venet, A. (eds.) NFM 2013. LNCS, vol. 7871, pp. 352–367. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38088-4_24

[DHKP16] Daca, P., Henzinger, T.A., Křetínský, J., Petrov, T.: Faster statistical model checking for unbounded temporal properties. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 112–129. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_7

[DHS18] D'Argenio, P.R., Hartmanns, A., Sedwards, S.: Lightweight statistical model checking in nondeterministic continuous time. In: Margaria, T., Steffen, B. (eds.) ISoLA 2018. LNCS, vol. 11245, pp. 336–353. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03421-4_22

[DLL+11a] David, A., et al.: Statistical model checking for networks of priced timed automata. In: Fahrenberg, U., Tripakis, S. (eds.) FORMATS 2011. LNCS, vol. 6919, pp. 80–96. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24310-3_7

[DLL+11b] David, A., Larsen, K.G., Legay, A., Mikučionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 349–355. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_27

[DLST15] D'Argenio, P., Legay, A., Sedwards, S., Traonouez, L.-M.: Smart sampling for lightweight verification of markov decision processes. STTT **17**(4), 469–484 (2015)

[EGF12] Ellen, C., Gerwinn, S., Fränzle, M.: Confidence bounds for statistical model checking of probabilistic hybrid systems. In: Jurdziński, M., Ničković, D. (eds.) FORMATS 2012. LNCS, vol. 7595, pp. 123–138. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33365-1_10

[FT14] Fu, J., Topcu, U.: Probably approximately correct MDP learning and control with temporal logic constraints. In: Robotics: Science and Systems (2014)

[HAK18] Hasanbeig, M., Abate, A., Kroening, D.: Logically-correct reinforcement learning. CoRR, 1801.08099 (2018)

[HAK19] Hasanbeig, M., Abate, A., Kroening, D.: Certified reinforcement learning with logic guidance. CoRR, abs/1902.00778 (2019)

[HJB+10] He, R., Jennings, P., Basu, S., Ghosh, A.P., Wu, H.: A bounded statistical approach for model checking of unbounded until properties. In: ASE, pp. 225–234 (2010)

[HKP+19] Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The quantitative verification benchmark set. In: TACAS 2019 (2019, to appear)

[HLMP04] Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 73–84. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24622-0_8

[HM17] Haddad, S., Monmege, B.: Interval iteration algorithm for MDPs and IMDPs. Theor. Comput. Sci. (2017)

[HMZ+12] Henriques, D., Martins, J., Zuliani, P., Platzer, A., Clarke, E.M.: Statistical model checking for Markov decision processes. In: QEST, pp. 84–93 (2012)

[HPS+19] Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., Wojtczak, D.: Omega-regular objectives in model-free reinforcement learning. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 395–412. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17462-0_27

[JCL+09] Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., Zuliani, P.: A bayesian approach to model checking biological systems. In: Degano, P., Gorrieri, R. (eds.) CMSB 2009. LNCS, vol. 5688, pp. 218–234. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03845-7_15

[JLS12] Jegourel, C., Legay, A., Sedwards, S.: A platform for high performance statistical model checking – PLASMA. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 498–503. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28756-5_37

[KKKW18] Kelmendi, E., Krämer, J., Křetínský, J., Weininger, M.: Value iteration for simple stochastic games: stopping criterion and learning algorithm. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 623–642. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_36

[KM19] Křetínský, J., Meggendorfer, T.: Of cores: a partial-exploration framework for Markov decision processes. Submitted 2019

[KNP11] Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47

[Lar12] Larsen, K.G.: Statistical model checking, refinement checking, optimization,. for stochastic hybrid systems. In: Jurdziński, M., Ničković, D. (eds.) FORMATS 2012. LNCS, vol. 7595, pp. 7–10. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33365-1_2

[Lar13] Guldstrand Larsen, K.: Priced timed automata and statistical model checking. In: Johnsen, E.B., Petre, L. (eds.) IFM 2013. LNCS, vol. 7940, pp. 154–161. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38613-8_11

[Lit94] Littman, M.L.: Markov games as a framework for multi-agent reinforcement learning. In: ICML, pp. 157–163 (1994)

[LN81] Lakshmivarahan, S., Narendra, K.S.: Learning algorithms for two-person zero-sum stochastic games with incomplete information. Math. Oper. Res. **6**(3), 379–386 (1981)

[LP08] Lassaigne, R., Peyronnet, S.: Probabilistic verification and approximation. Ann. Pure Appl. Logic **152**(1–3), 122–131 (2008)

[LP12] Lassaigne, R., Peyronnet, S.: Approximate planning and verification for large Markov decision processes. In: SAC, pp. 1314–1319, (2012)

[LST14] Legay, A., Sedwards, S., Traonouez, L.-M.: Scalable verification of markov decision processes. In: Canal, C., Idani, A. (eds.) SEFM 2014. LNCS, vol. 8938, pp. 350–362. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15201-1_23

[Mar75] Martin, D.A.: Borel determinacy. Ann. Math. **102**(2), 363–371 (1975)

[MLG05] Mcmahan, H.B., Likhachev, M., Gordon, G.J.: Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In: In ICML 2005, pp. 569–576 (2005)

[Nor98] Norris, J.R.: Markov Chains. Cambridge University Press, Cambridge (1998)

[PGL+13] Palaniappan, S.K., Gyori, B.M., Liu, B., Hsu, D., Thiagarajan, P.S.: Statistical model checking based calibration and analysis of bio-pathway models. In: Gupta, A., Henzinger, T.A. (eds.) CMSB 2013. LNCS, vol. 8130, pp. 120–134. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40708-6_10

[Put14] Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, Hoboken (2014)

[RF91] Raghavan, T.E.S., Filar, J.A.: Algorithms for stochastic games – a survey. Z. Oper. Res. **35**(6), 437–472 (1991)

[RP09] El Rabih, D., Pekergin, N.: Statistical model checking using perfect simulation. In: Liu, Z., Ravn, A.P. (eds.) ATVA 2009. LNCS, vol. 5799, pp. 120–134. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04761-9_11

[SB98] Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)

[SKC+14] Sadigh, D., Kim, E.S., Coogan, S., Sastry, S.S.S., Sanjit, A.: A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. In: CDC, pp. 1091–1096 (2014)

[SLW+06] Strehl, A.L., Li, L., Wiewiora, E., Langford, J., Littman, M.L.: PAC model-free reinforcement learning. In: ICML, pp. 881–888 (2006)

[SS12] Saffre, F., Simaitis, A.: Host selection through collective decision. ACM Trans. Auton. Adapt. Syst. **7**(1), 4:1–4:16 (2012)

[SVA04] Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 202–215. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27813-9_16

[SVA05] Sen, K., Viswanathan, M., Agha, G.: On statistical model checking of stochastic systems. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 266–280. Springer, Heidelberg (2005). https://doi.org/10.1007/11513988_26

[WT16] Wen, M., Topcu, U.: Probably approximately correct learning in stochastic games with temporal logic specifications. In: IJCAI, pp. 3630–3636 (2016)

[YCZ10] Younes, H.L.S., Clarke, E.M., Zuliani, P.: Statistical verification of probabilistic properties with unbounded until. In: Davies, J., Silva, L., Simao, A. (eds.) SBMF 2010. LNCS, vol. 6527, pp. 144–160. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19829-8_10

[YKNP06] Younes, H.L.S., Kwiatkowska, M.Z., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking. STTT **8**(3), 216–228 (2006)

[YS02a] Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 223–235. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45657-0_17

[ZPC10] Zuliani, P., Platzer, A., Clarke, E.M.: Bayesian statistical model checking with application to simulink/stateflow verification. In: HSCC, pp. 243–252 (2010)

D Statistical Model Checking: Black or White? (ISoLA 2020)

This is an exact reprinting of the accepted version of the following paper which has been published as a **peer reviewed conference paper**.

> Pranav Ashok, Przemyslaw Daca, Jan Kretinsky, and Maximilian Weininger. "Statistical Model Checking: Black or White?" In: ISoLA (1). Vol. 12476. Lecture Notes in Computer Science. Springer, 2020, pp. 331–349. doi: 10.1007/978-3-030-61362-4_19

Summary

One of the advantages of statistical model checking (SMC) is its applicability to black-box systems. In this paper, we discuss the advantages gained when SMC is applied to white-box systems, utilizing the knowledge of their internals. We focus on the setting of unbounded-horizon properties such as reachability or LTL. We compare our approach to other statistical and numerical techniques both conceptually as instantiations of the same framework, and experimentally. It not only clearly preserves scalability advantages of black-box SMC compared to classical model checking (while providing high level of guarantees), but it also scales yet better than either of the two for a wide class of models.

For more details on this publication, we refer to Section 4.1 of the main body.

Contribution

Contribution of Maximilian Weininger	
Development and conceptual design of the research project	40%
Discussion and development of ideas	60%
Composition and revision of the manuscript	80%
Implementation	10%
Experimental evaluation	15%

Statistical Model Checking: Black or White?

Pranav Ashok( ), Przemysław Daca, Jan Křetínský( ), and Maximilian Weininger( )

Technical University of Munich, Munich, Germany
ashok@in.tum.de, jan.kretinsky@tum.de, maxi.weininger@tum.de

Abstract. One of the advantages of statistical model checking (SMC) is its applicability to black-box systems. In this paper, we discuss the advantages gained when SMC is applied to white-box systems, utilizing the knowledge of their internals. We focus on the setting of unbounded-horizon properties such as reachability or LTL. We compare our approach to other statistical and numerical techniques both conceptually as instantiations of the same framework, and experimentally. It not only clearly preserves scalability advantages of black-box SMC compared to classical model checking (while providing high level of guarantees), but it also scales yet better than either of the two for a wide class of models.

1 Introduction

Classical *probabilistic verification* techniques rely on iterative approximation algorithms for linear equation systems and linear programs, such as *value iteration (VI)*, e.g. [Put14]. However, the scalability of such numeric analyses is severely limited, compared to standard non-quantitative (hardware or software) verification, since exact transformations, such as abstraction or partial-order reduction, are more difficult to use. Consequently, weaker guarantees such as *probably approximately correct (PAC)* results become acceptable even for completely known systems (*white box*) and not only in contexts where the system is executable but unknown (*black box*), and where thus absolute guarantees are principally impossible.

Example 1. Consider the task of model checking a reachability property of a probabilistic communication protocol, which starts by generating a few, say k, random bits. Thus the execution immediately branches into 2^k states. If there are only few or hard-to-find symmetries in the behaviour, standard analysis quickly becomes infeasible. In the following, we discuss drawbacks of previously studied alternative approaches; then we suggest a new one that overcomes the difficulties for a wide class of models.

This research was funded in part by TUM IGSSE Grant 10.06 (PARSEC) and the German Research Foundation (DFG) project 383882557 *Statistical Unbounded Verification* (KR 4890/2-1).

The exponential state-space explosion quickly renders explicit VI unable to propagate information by more than a single step. Besides, if the transition probabilities depend on the generated bits, even the symbolic variants of VI [BCH+97] cannot help much. There have been two major alternatives proposed, both relying on extensive use of simulations.

- (I) For large and possibly *unknown* systems, *statistical model checking (SMC)* [YS02] reincarnates the Monte Carlo method. It runs simulations of the system; the resulting statistics then yields confidence intervals, i.e. PAC results. However, for unbounded-horizon properties, such as reachability or linear temporal logic (LTL) [Pnu77], performing simulations of finite length requires some information about the model [Kře16]:

  1. Either the second eigenvalue of the transition matrix can be bounded [LP08, YCZ10], which requires essentially the complete knowledge of the system (white box) and is as hard as solving the model checking problem, or
  2. the topology of the underlying state-graph is known [YCZ10, HJB+10] (sometimes called *grey box*, e.g. [AKW19]) and the whole system is preprocessed, which beats the purpose of sublinear analysis, or
  3. a bound on the minimum transition probability $p_{\min}$ is known as is the case in [BCC+14, DHKP17]. This is the closest to black box, thus called *black SMC* here.

  In black SMC, long enough simulations can be run to ensure the system passes from the transient to the recurrent part and reliable information on the whole infinite run is obtained. While the a-priori length is practically infeasible [BCC+14], early detection of recurrent behaviour has been proposed [DHKP17] as follows. Based on the observed part of a simulation run, a hypothesis on the topology of the system is made, answering what bottom strongly connected component (BSCC) this run ends up in. With repetitive observations of transitions over the run, the confidence grows that what currently looks as a BSCC indeed is a BSCC. Since quite a few repetitions of *all* transitions in the BSCC are required, this approach turns out practical only for systems with small BSCCs and not too small $p_{\min}$.
  In this paper, assuming knowledge of the system (white-box setting), we twist the technique to a more efficient one as follows. After quickly gaining (unreliably low) confidence that the run entered a BSCC, we use the knowledge of the topology to confirm this information—again very quickly since not the whole model is built but only the *local* BSCC. Consequently, BSCCs are detected fast even in the case with larger BSCCs or small $p_{\min}$. As the information used turns out quite limited, corresponding to the grey-box setting, we call this approach *grey SMC*.

- (II) The other alternative to VI, now in the context or large but *known* systems, is the *asynchronous value iteration*, e.g. [BT89], a generalization of the Gauss-Seidel method and the core of reinforcement learning and approximate dynamic programming. There, the VI updates on states of the system are performed in varying orders, in particular possibly entirely skipping some states.

The class of algorithms providing guarantees is represented by *bounded real-time dynamic programming (BRTDP)* [MLG05, BCC+14, AKW19] where the states to be updated at each moment are those appearing on a current simulation run. Consequently, states with low probability of visiting and thus low impact on the overall value are ignored. While this allows for treating very "wide" systems with lots of unimportant branches, the scalability problem persists as soon as the branching is very uniform (see also Example 5 on Fig. 2b). From this perspective, grey SMC relaxes the rigorous approximation in the transient part and replaces it with a statistical estimate.

Overall, grey SMC fills the gap in the following spectrum:

VI	BRTDP	grey SMC	black SMC
analysis	analysis with simulation inside	simulation with analysis inside	simulation

On the one end, numeric analysis (VI) provides reliable results; in BRTDP, simulations are additionally used in the analysis to improve the performance while preserving the guarantees. On the other end, simulations (SMC) provide PAC guarantees; grey SMC then improves the performance by additional analysis in the simulation.

Our contribution can be summarized as follows:

- We modify the black SMC for unbounded properties of [DHKP17] to perform better in the white-box (and actually also in the so-called grey-box) setting.
- We compare our grey SMC to black SMC, BRTDP and VI both conceptually, illustrating advantages on examples, as well as experimentally, comparing the runtimes on standard benchmarks.
- We present all algorithms within a unified framework, which in our opinion eases understanding and comparison, provides a more systematic insight, and is pedagogically more valuable.

Outline of the Paper: After recalling necessary definitions in Sect. 2, we describe and exemplify the algorithms in Sect. 3 and the respective key sub-procedure in Sect. 4. Then we compare the algorithms and other related work in Sect. 5, discussing the expected implications, which we confirm experimentally in Sect. 6. For a broader account on related work on SMC in the context of unbounded-horizon properties, we refer the interested reader to the survey [Kře16].

2 Preliminaries

A *probability distribution* on a finite set X is a mapping $\delta : X \to [0,1]$, such that $\sum_{x \in X} \delta(x) = 1$. The set of all probability distributions on X is denoted by $\mathcal{D}(X)$.

Definition 1 (MC). *A* Markov chain *(MC) is a tuple* $(\mathsf{S}, \mathsf{s}_0, \delta)$*, where* S *is a finite set of* states *with a designated* initial state $\mathsf{s}_0 \in \mathsf{S}$*, and* $\delta : \mathsf{S} \to \mathcal{D}(\mathsf{S})$

is a transition function *that given a state* s *yields a probability distribution* $\delta(\mathsf{s})$ *over* successor *states. For ease of notation, we write* $\delta(\mathsf{s},\mathsf{t})$ *instead of* $\delta(\mathsf{s})(\mathsf{t})$ *and* $\mathsf{Post}(\mathsf{s}) := \{\mathsf{t} \mid \delta(\mathsf{s},\mathsf{t}) > 0\}$ *to denote the set of successors of a state.*

The semantics of an MC is given in the usual way by the probability space on paths. An *infinite path* ρ is an infinite sequence $\rho = \mathsf{s}_0\mathsf{s}_1\cdots \in (\mathsf{S})^\omega$, such that for every $i \in \mathbb{N}$ we have $\mathsf{s}_{i+1} \in \mathsf{Post}(\mathsf{s}_i)$. A finite path is a finite prefix of an infinite path. The Markov chain together with a state s induces a unique probability distribution $\mathbb{P}_\mathsf{s}$ over measurable sets of infinite paths [BK08, Ch. 10].

Definition 2 (Reachability probability). *For a target set* $\mathsf{T} \subseteq \mathsf{S}$*, we write* $\Diamond\mathsf{T} := \{\mathsf{s}_0\mathsf{s}_1\cdots \mid \exists i \in \mathbb{N} : \mathsf{s}_i \in \mathsf{T}\}$ *to denote the (measurable) set of all infinite paths which eventually reach* T*. For each* $\mathsf{s} \in \mathsf{S}$*, we define the* value *in* s *as*

$$\mathsf{V}(\mathsf{s}) := \mathbb{P}_s(\Diamond\mathsf{T}).$$

The reachability probability *is then the value of the initial state* $\mathsf{V}(\mathsf{s}_0)$*.*

The value function V satisfies the following system of equations, which is referred to as the *Bellman equations*:

$$\mathsf{V}(\mathsf{s}) = \begin{cases} 1 & \text{if } \mathsf{s} \in \mathsf{T} \\ \sum_{\mathsf{s}' \in \mathsf{S}} \delta(\mathsf{s},\mathsf{s}') \cdot \mathsf{V}(\mathsf{s}') & \text{otherwise} \end{cases} \tag{1}$$

Moreover, V is the *least* solution to the Bellman equations, see e.g. [CH08].

Certain parts of the state space are of special interest for the analysis of MC with respect to unbounded-horizon properties, such as reachability:

Definition 3 (SCC, BSCC). *A non-empty set* $T \subseteq \mathsf{S}$ *of states is* strongly connected *if for every pair* $\mathsf{s},\mathsf{s}' \in \mathsf{S}$ *there is a path (of non-zero length) from* s *to* s'*. Such a set* T *is a* strongly connected component (SCC) *if it is maximal w.r.t. set inclusion, i.e. there exists no strongly connected* T' *with* $T \subsetneq T'$*. An SCC* T *is called* bottom (BSCC)*, if for all states* $\mathsf{s} \in T$ *we have* $\mathsf{Post}(\mathsf{s}) \subseteq T$*, i.e. no transition leaves the SCC.*

Note that the SCCs of an MC are disjoint and that, with probability 1, infinitely often reached states on a path form a BSCC.

We consider algorithms that have a limited information about the MC:

Definition 4 (Black box and grey box setting). *An algorithm inputs an MC as* black box *if it cannot access the whole tuple, but*

– *it knows the initial state,*
– *for a given state, it can sample a successor* t *according to* $\delta(\mathsf{s})$*,*[1]

[1] Up to this point, this definition conforms to black box systems in the sense of [SVA04] with sampling from the initial state, being stricter than [YS02] or [RP09], where simulations can be run from any desired state.

– *it knows* $p_{\min} \leq \min_{\mathsf{s}\in\mathsf{S},\mathsf{t}\in\mathsf{Post}(\mathsf{s})} \delta(\mathsf{s},\mathsf{t})$*, an under-approximation of the minimum transition probability.*

When input as grey box*, it additionally knows the number* $|\mathsf{Post}(\mathsf{s})|$ *of successors for each state* s.[2]

3 Description of Algorithms

In this section, we describe all of the algorithms that we compare in this paper. They all use the framework of Algorithm 1. The differences are in the instantiations of the functions (written in capital letters). This allows for an easy and modular comparison.

Algorithm 1. Framework for all considered algorithms

```
Input: MC M, reachability objective T
Output: (An estimate of) $\mathbb{P}_{\mathsf{s}_0}(\Diamond \mathsf{T})$
1: procedure COMPUTE REACHABILITY PROBABILITY
2:     INITIALIZE
3:     repeat
4:         $X \leftarrow$ GET_STATES
5:         UPDATE($X$)
6:     until TERM_CRIT
```

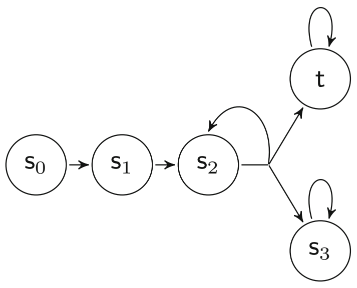
3.1 Value Iteration

Value iteration (VI), e.g. [Put14], computes the value for all states in the MC. As memory, it saves a rational number (the current estimate of the value) for every state. In INITIALIZE, the estimate is set to 1 for target states in T and to 0 for all others. GET_STATES returns the whole state space, as the estimate of all values is updated simultaneously. The UPDATE works by performing a so called *Bellman backup*, i.e.g.iven the current estimate function L_i, the next estimate L_{i+1} is computed by applying the Bellman Equation (1) as follows:

$$\mathsf{L}_{i+1}(\mathsf{s}) = \sum_{\mathsf{s}'\in\mathsf{S}} \delta(\mathsf{s},\mathsf{s}') \cdot \mathsf{L}_i(\mathsf{s}')$$

Example 2. Consider the MC from Fig. 1a, with $\delta(\mathsf{s}_2,\mathsf{s}_2) = \delta(\mathsf{s}_2,\mathsf{t}) = \delta(\mathsf{s}_2,\mathsf{s}_3) = \frac{1}{3}$ and the reachability objective $\{t\}$. The estimates that VI computes in the first 4 iterations are depicted in Fig. 1b. The target state t is initialized to 1, everything else to 0. The estimate for s_3 stays at 0, as it is a BSCC with no possibility to

[2] This requirement is slightly weaker than the knowledge of the whole topology, i.e. $\mathsf{Post}(\mathsf{s})$ for each s.



(a)

Iter.	$L(s_0)$	$L(s_1)$	$L(s_2)$
0	0	0	0
1	0	0	$1/3$
2	0	$1/3$	$4/9$
3	$1/3$	$4/9$	$13/27$
4	$4/9$	$13/27$	$40/81$

(b)

Iter.	$L(s_2)$	$U(s_2)$
0	0	1
1	0.01	0.99
2	0.029	0.980
3	0.039	0.970
4	0.048	0.961

(c)

Fig. 1. (a) Example Markov chain (b) Under approximations computed by value iteration, see Example 2 (c) Under- and over-approximations computed by bounded value iteration, see Example 3.

reach the target state. Since these two states do not change, they are omitted in the figure. In every iteration, the estimates are updated and become more precise, coming closer to the true value 0.5 for s_0, s_1 and s_2. However, they converge to 0.5 only in the limit, as for any finite number of iterations there is a positive probability to remain in s_2. Note that s_0 always is two steps behind s_2, as it takes two iterations to backpropagate the current estimate.

VI converges to the true value only in the limit, hence we need some termination criterion TERM_CRIT to stop when we are close enough. However, to be certain that the estimate is close, one has to perform an exponential number of iterations [CH08], which is infeasible. Hence, usually this version of VI does not give convergence guarantees, but instead just runs until the difference between two successive iterations is small. The result of this heuristic is guaranteed to be a lower bound, but can be arbitrarily imprecise [HM18], as we will also see in Example 3.

3.2 Bounded Value Iteration

To be able to give convergence guarantees, *Bounded value iteration* (BVI, also called interval iteration) was introduced more generally for Markov decision processes in [BCC+14,HM18]. In this paper, we only focus on Markov chains, i.e. Markov decision processes with a single action in every state. In addition to the under-approximation computed by VI, this approach also computes a convergent over-approximation. For this, it stores a second rational number for every state. Dually to the under-approximation, INITIALIZE sets the estimate to 0 in states that cannot reach the target and 1 everywhere else. Note that finding the states with value 0, i.e. BSCC that do not contain the target, BVI has to perform a graph analysis, e.g. a backwards search from the targets. BVI still works on the whole state space and the update is completely analogous to VI, only this time updating both approximations. As TERM_CRIT, BVI checks that difference between the over- and under-approximation in the initial state is smaller than a given precision ε. This guarantees that the returned value is ε-precise.

Example 3. Consider the MC from Fig. 1a with the same objective, but this time with $\delta(\mathsf{s}_2, \mathsf{s}_2) = 0.98$ and $\delta(\mathsf{s}_2, \mathsf{t}) = \delta(\mathsf{s}_2, \mathsf{s}_3) = 0.01$. Note that by pre-processing we set the over approximation $\mathsf{U}(\mathsf{s}_3)$ to 0, as it is a BSCC with no possibility of reaching the target. The estimates BVI computes for s_2 in the first 4 iterations are depicted in Fig. 1c.

If we were running VI only from below, we might stop after iteration 4, as the lower bound changes by less than 0.01 between these iterations and hence it seems to have converged close to the value. However, the difference between upper and lower bounds is still very high, so BVI knows that there still is a huge uncertainty in the values, as it could be anything between 0.048 and 0.961. Eventually, both estimates converge close enough to 0.5; for example, after around 400 iterations the lower bound is 0.49 and the upper bound 0.51. Then BVI can return the value 0.5 (the center of the interval) with a precision of 0.01, as this value is off by at most that.

3.3 Simulation-Based Asynchronous Value Iteration

The biggest drawback of the two variants we introduced so far is that they always work on the whole state space. Because of the state-space explosion, this is often infeasible. In contrast, asynchronous value iteration only updates parts of the state space in every iteration of the loop, i.e. GET_STATES does not return the whole state space, but instead heuristically selects the states to update next. This not only speeds up the main loop, but also allows the algorithm to reduce the memory requirements. Indeed, instead of storing estimates for all states, one stores estimates only for the partial model consisting of previously updated states. In [BBS95, MLG05, BCC+14], the heuristic for selecting the states is based on simulation: a path is sampled in the model, and only the states on that path are updated. The partial model contains all states that have been encountered during some of the simulations. If the part of the state space that is relevant for convergence of value iteration is small, this can lead to enormous speed-ups [BCC+14, KM19]. For more details on why this happens and a formal definition of 'state space relevant for convergence', we refer the interested reader to [KM19].

Algorithm 2. Simulation-based implementation of GET_STATES

```
Input: MC $\mathsf{M}$, reachability objective $\mathsf{T}, \mathsf{s}_0$
Output: A set of states $X \subseteq \mathsf{S}$
1: procedure SIMULATE
2:     $\rho \leftarrow \mathsf{s}_0$
3:     repeat
4:         $\mathsf{s}' \leftarrow$ sample from $\delta(\text{last}(\rho))$ according to NEXT_STATE
5:         $\rho \leftarrow \rho\mathsf{s}'$
6:     until $\text{last}(\rho) \in \mathsf{T}$ or STUCK
7:     return $\rho$
```

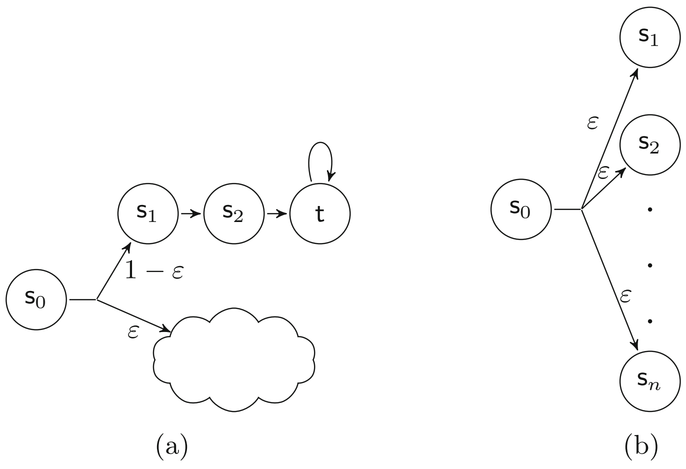


Fig. 2. (a) A Markov chain where exploring the whole state space can be avoided. ε denotes a transition probability. The cloud represents an arbitrarily large state space. (b) A Markov chain with high branching. From s_0, there is a uniform probabilistic choice with $n = \frac{1}{\varepsilon}$ successors.

Algorithm 2 shows how states can be sampled through simulations, as done in [BCC+14]: Starting from the initial state, in every step of the simulation a successor is chosen from the distribution of the last state on the path. Note that this choice depends on another heuristic NEXT_STATE. The successor can be chosen according to the transition probabilities δ, but it has proven to be advantageous to additionally consider the difference between the upper and lower bound in the successor states [MLG05,BCC+14]. In consequence, states where we already know a lot (under- and over-approximations are close to each other) are given less priority than states where we still need information.

The simulation is stopped in two cases: Either (i) it reaches a target state or (ii) it is stuck in a BSCC with no path to the target. Different heuristics for checking whether the simulation is stuck are discussed in depth in Sect. 4. Note that being able to differentiate between targets and non-target BSCCs during the simulations allows us not to do anything in INITIALIZE; we can set the value to 1 when reaching a target and 0 in the other case. The UPDATE function for simulation based asynchronous value iteration again uses the Bellman equation (1) to update the estimates of all states on the path; moreover, it can utilize additional information: Since GET_STATES returns a path, there is a notion of order of the states. Updating the states in reverse order backpropagates information faster.

Example 4. Consider the MC in Fig. 2a, again with reachability objective $\{t\}$. The cloud represents an arbitrarily large state space. However, since it is only reachable with a very small probability ε (and we are interested in an ε-precise solution), it need not be explored. Let the first sampled path be $\mathsf{s}_0\mathsf{s}_1\mathsf{s}_2\mathsf{t}$. This happens with high probability, as the only other possibility would be to select a successor from the cloud in state s_0, but since the selection process depends on the transition probabilities δ, going to s_1 has a higher probability. After the simulation reaches t, this value is backpropagated in reverse order. First the lower estimate $\mathsf{L}(\mathsf{s}_2)$ is set to 1, then $\mathsf{L}(\mathsf{s}_1)$ is set to 1, then $\mathsf{L}(\mathsf{s}_0)$ is set to $1 - \varepsilon$. At this point the algorithm has converged, as difference between the lower and upper bound is ε.

So in this example, sampling the most probable path a single time gives a good approximation. The algorithm avoids exploring the large cloud and backprogagates values faster than synchronous VI.

Example 5. As an adversarial example, consider the MC in Fig. 2b. Here, the model exhibits high branching, so every single path has a low probability, and only by aggregating all paths we actually get a high value. Unlike the previous example, there is no part of the state space that is clearly irrelevant. In fact, to achieve precision of ε the algorithm has to see so many paths that their cumulative probability is $1 - \varepsilon$, which in this case means seeing all but one transition from the starting state. This needs at least $\frac{1}{\varepsilon}$ simulations, but since the successors are chosen probabilistically, most likely a lot more.

Note that similarly to synchronous VI, there are versions of asynchronous VI without (RTDP [BBS95]) and with (BRTDP [MLG05,BCC+14])[3] guaranteed error bounds.

3.4 Statistical Model Checking

Algorithms for statistical model checking (SMC), [YS02], are different from all previously described ones in two ways, namely what they store and what they return. The VI-based algorithms store estimates for every (seen) state and they update these values to be ever more precise. Thus, the returned bounds on the values are certainly correct, although possibly quite loose. In contrast, SMC stores only a single accumulator (for the value of the initial state) and the returned value is probably approximately correct (PAC [Val84]). Being PAC with probability α and approximation $\epsilon > 0$ guarantees the following: with high probability (α), the returned value is close to the true value (off by at most ε). However, the returned confidence interval is not guaranteed to be a valid under- and over-approximation; if we are unlucky (i.e. with the remaining probability $1 - \alpha$), there is no guarantee whatsoever on the returned value.

SMC does not need to do anything in $\mathsf{INITIALIZE}$. It only stores a single accumulator to remember how often a target state was reached. $\mathsf{GET\_STATES}$ works as in Algorithm 2 with NEXT_STATE typically sampling the successor according to the transition probabilities δ (in some settings, importance sampling may also be possible, e.g. [JLS12,BDH17]). UPDATE remembers whether we reached the target or not; in the end we can divide the number of reaches by the total number of samples to get the probability estimate. $\mathsf{TERM\_CRIT}$ is a (typically low) number of samples that depends on the required probability of

[3] While all are more generally applicable to Markov decision processes, [MLG05] only ensures convergence if no end components [BK08] are present (for MC, no BSCCs without a target are present) and [BCC+14] lifts this restriction.

the guarantee and the width of the confidence interval; see [DHKP17, Section 2.2] for details or [JSD19] for more advanced techniques.

Example 6. Consider again the MC depicted in Fig. 1a. Let the first sampled path be $\mathsf{s}_0\mathsf{s}_1\mathsf{s}_2\mathsf{s}_2\mathsf{t}$. At this point the simulation stops, as we have reached a target state, and we remember that we have seen a target once. Let the second path be $\mathsf{s}_0\mathsf{s}_1\mathsf{s}_2\mathsf{s}_2\mathsf{s}_2\mathsf{s}_3\mathsf{s}_3\ldots$. On the one hand, the STUCK function has to let the simulation continue, even though s_2 is seen 3 times and it looks like a cycle. On the other hand, it has to detect that the simulation will loop forever in s_3 and has to stop it. Ways to detect this are discussed in Sect. 4. After detecting that we are stuck, we remember that the simulation did not reach the target.

Let the required probability of the guarantee be $\alpha = 0.9$ and the width of the confidence interval $\varepsilon = 0.1$. Using Hoeffding's inequality [Hoe63] we can show that the required number of samples for this is 461. So assume that after 461 simulations we have seen the target 223 times. Then we know that with probability at least 0.9, the value is in the interval $^{223}/_{461} \pm 0.05$, i.e. $[0.434, 0.534]$. Increasing the number of simulations can both increase the confidence or decrease the width of the interval.

Note that this number of simulations is independent of the system. While 461 simulations are a lot for this small system, the number would be the same if we were considering a model with several billion states where value iteration is impossible.

4 STUCK

In this section, we discuss heuristics for detecting whether a simulation is stuck in a BSCC with no path to a target state. We also propose one new such heuristic with convenient theoretical properties.

For simulation-based asynchronous value iteration, previous work either excluded the existence of non-target BSCCs in their assumptions [BBS95, MLG05] or used a heuristic with no false negatives, but the possibility of false positives [BCC+14]. This means that if the simulation is stuck in a BSCC, the simulation definitely is stopped, which is required for termination. However, if the simulation is not stuck in a BSCC, it might still be stopped, guessing the value of the last state in the path is 0, although it might not be. The STUCK-heuristics used in previous work either depend on the path length ([BCC+14, Ujm15, Chapter 7.5]) or simply stops exploring when any state is seen twice [AKW19, Appendix A.3].

SMC has to be sure with high probability that the simulation is stuck, as otherwise it loses the probabilistic guarantee. In [YCZ10], two approaches are described. The first approach requires knowledge of the second eigenvalue of the MC in order to guarantee asymptotic convergence. However, getting the second eigenvalue is as hard as the verification problem itself. The second approach works in the grey-box setting and pre-processes the MC so that all potentially infinite paths are eliminated. A similar transformation, using white-box information, was suggested in [HJB+10]. However, both of these approaches transform the whole model and thus face problems in the case of very large models.

An alternative was suggested in [DHKP16]. It monitors the finite path sampled during the simulation, implicitly constructing a graph with all seen states as nodes and all seen transitions as edges. The *candidate* of the current path is the (possibly empty) set of states forming the maximal BSCC of this graph. Intuitively, it is what we believe to be a BSCC given the observation of the current simulation. This candidate has to be validated, because as we saw in Example 6, a state set can look like a BSCC for several steps before being exited. In the black-box setting, this validation works by continuing the simulation until the probability of overlooking some transition exiting the candidate becomes very small [DHKP16].

In this paper, we pinpoint that in the grey-box or white-box setting, this costly type of validation is not necessary. Instead of validating the candidate by running around in it for a huge number of steps, one can verify it using the additional information on the model. If no successor of any state in the candidate is outside of the candidate, then it indeed is a BSCC. Formally, for a candidate T, we check that $\{\mathsf{s} \mid \exists \mathsf{t} \in T : \mathsf{s} \in \mathsf{Post}(t)\} \subseteq T$ (if the topology is known), or alternatively that $\forall \mathsf{t} \in T : |\widehat{\mathsf{Post}}(t)| = |\mathsf{Post}(t)|$ (in what we defined as the grey-box setting) where $\widehat{\mathsf{Post}}$ yields the number of successors within the observed candidate.

Example 7. Consider again the MC depicted in Fig. 1a. When a simulation enters s_3, STUCK should return true in order to stop the simulation, as it has reached a BSCC with no path to a target. In the black box setting of [DHKP16], this is only possible after continuing the simulation for another huge amount of steps. For example, even in a BSCC with only a single state, hundreds of further steps can be necessary to reach the required confidence. Given the grey-box information, the algorithm can determine that all successors of the states in the candidate $(\{\mathsf{s}_3\})$ have been seen and conclude that the candidate is indeed a BSCC.

However, this check stops the simulation and can incur an overhead if there are many SCCs in the transient part of the state space. Hence, we can delay it, not checking at the first occurrence of a cycle, but e.g. only when every state in the candidate has been seen twice. Alternatively, one can only allow the check every n (e.g. hundred) steps of the simulation. Depending on the model and the implementation of the algorithm, these heuristics can have some impact on the runtime.

Furthermore, one might modify this heuristic even further. If a state of the BSCC is only reached with low probability, it takes many steps for the simulation to reach it. When we check whether the current candidate is a BSCC, this state might not have occurred in the simulation yet. Instead of concluding that the information is insufficient and the simulation has to continue, one could deterministically explore the unknown successors and *compute* the BSCC. On the one hand, for small to medium sized BSCCs, this could result in a speed-up. On the other hand, it increases the overhead when transient SCCs are checked by STUCK. Consequently, in the available benchmarks, this heuristic did not prove advantageous. Hence we do not even report on it in the evaluation section.

5 Discussion

5.1 Dependency of Simulation Length on Topology

Although the number of samples in SMC is independent of the model size, the length of the simulations is highly dependent on the model size and even more on the structure. Indeed, any kind of cyclic behaviour in the transient part of the state space increases the simulation time for two reasons. Firstly, the simulation loops in transient SCCs and does not make progress towards a target or a BSCC. Secondly, the check whether the simulation is stuck in transient SCCs incurs an overhead. An adversarial handcrafted worst-case example where simulations struggle is given in [HM18, Figure 3]. Moreover, the structure of BSCCs affects the length of the simulation. For cyclic BSCCs, the simulation easily encounters all states of the BSCC and can quickly terminate. For more complex topologies, some states are typically only seen with very low frequency and thus the simulation takes longer.

If the model exhibits many transient SCCs, using any simulation-based technique is problematic.

5.2 Black, Grey and White SMC

The difference between the variants of SMC we report on are their knowledge of the transition system: $p_{\min}$ corresponds to black, the number of successors to grey and the exact successors and probabilities to the white-box setting. This information can be used in the STUCK-check; apart from that, the algorithms are the same.

Comparing grey and black box, it is apparent that simulations in grey box can be much shorter, as upon detection of a candidate that is a BSCCs the simulation is immediately stopped, whereas in the black box setting it has to continue for a number of steps. This number of steps depends on two things: (i) The size of the BSCC, as larger BSCC take longer to explore, especially since all states, no matter how improbable, need to be seen a certain amount of times, and (ii) the given under-approximation of the minimum transition probability p_{min}, as this determines how often every state in the candidate has to be seen until the probability of a false positive is small enough.

Thus, for large BSCCs or small $p_{\min}$, grey SMC is clearly better, as we also experimentally validate in Table 3 (large BSCC) and Table 2 (various $p_{\min}$) in the next section. For small BSCCs (e.g. only of size 1) and not so small $p_{\min}$, black and grey SMC become more comparable, but grey SMC still has shorter simulations. However, practically, the overhead of verifying the candidates in grey SMC can be so large that black SMC can even be slightly faster than grey SMC (see e.g., `leader6_11` in Table 1).

Heuristically reducing the number of checks in grey SMC (as described in Sect. 4) can make it faster again, but the effectiveness of the heuristics depends on the models. So, if it is known that the BSCC-detection is very easy for black SMC (e.g. they are of size 1 or cyclic and $p_{\min}$ is not too small), black SMC can be a viable choice. However, as black SMC is never far better, using grey SMC is the safer variant when facing models with uncertain topology.

5.3 Comparison of Algorithms

Finally, we compare the (dis-)advantages of the different algorithms, giving a practical decision guidance. If hard guarantees are required, then BVI or BRTDP are to be used. The latter is simulation based, and thus good if only a small part of the state space is relevant for convergence. Additionally, if the model is too large for BVI, BRTDP still has a chance, but quite possibly the partial model will also be too large. Conversely, if the model contains lots of transient SCCs, BVI is preferable, as simulation based approaches fail on this kind of model, see Sect. 5.1. Note that, if there are small probabilities present, it might take very long for BVI and BRTDP to converge, see Example 3.

For a quick estimate, or if PAC guarantees are sufficient, or if the system is too large, so that it is not possible to provide hard guarantees, SMC is to be used, if possible (white or grey box setting) in our grey variant. As both the memory and the termination criterion are independent of the size of the system, SMC always has a chance to yield an estimate, which additionally comes with a probabilistic guarantee.

There is no case in which un-guaranteed (synchronous or asynchronous) VI are preferable, as they suffer from the same drawbacks as BVI and BRTDP, but additionally do not provide guarantees. Whenever hard guarantees are not of interest and the system is not strongly connected, grey SMC should be used for a quick estimate.

5.4 Extensions to Other Unbounded-Horizon Properties

For more complex unbounded-horizon properties [BK08], such as Until (avoid-reach), LTL or long-run average reward, (B)VI pre-processes the state space to analyze the BSCCs [BK08] and BRTDP [BCC+14] can either do the same or analyze the encountered BSCCs only. Black SMC of [DHKP17] is applicable through additional analysis of the BSCC candidates after they have been found likely to be BSCCs. This is directly inherited by grey SMC and makes it available for these specifications with low overhead.

6 Experimental Evaluation

We implemented grey SMC in a branch of the PRISM Model Checker [KNP11] extending the implementation of black SMC [DHKP17]. We ran experiments on (both discrete- and continuous-time) Markov chains from the PRISM Benchmark

Table 1. Runtime (in seconds) comparison of black and grey SMC for various benchmarks. BVI runtimes are also presented as a baseline.

Model/property	Size	$p_{\min}$	BSCC (no., max. size)	SMC		BVI
				Black	Grey	
`bluetooth(10)time_qual`	$>569K$	7.81×10^{-3}	$>5.8K, 1$	9	**7**	TO
`brp_nodl(10K,10K)p1_qual`	$>40M$	1×10^{-2}	$>4.5K, 1$	86	**84**	TO
`crowds_nodl(8,20)positive_qual`	68M	5×10^{-2}	$>3K, 1$	10	**8**	TO
`egl(20,20)unfairA_qual`	1719T	5×10^{-1}	$1, 1$	43	**25**	TO
`gridworld(400,0.999)prop_qual`	384M	1×10^{-3}	$796, 160K$	15	**8**	TO
`herman-174tokens`	10G	4.7×10^{-7}	$1, 34$	TO	**73**	98
`leader6_11elected_qual`	$>280K$	5.6×10^{-7}	$1, 1$	**106**	152	OOM
`nand(50,3)reliable_qual`	11M	2×10^{-2}	$51, 1$	11	**10**	455
`tandem(2K)reach_qual`	$>1.7M$	2.4×10^{-5}	$1, >501K$	**7**	**7**	62

Suite [KNP12a]. In addition to a comparison to black SMC, we also provide comparisons to VI and BVI of PRISM and BRTDP of [BCC+14]. An interested reader may also want to refer [DHKP17, Table II] for a comparison of black SMC against two unbounded SMC techniques of [YCZ10].

For every run configuration, we run 5 experiments and report the median. In black SMC, the check for candidates is performed every 1000 steps during path simulations, while in grey SMC the check is performed every 100 steps. Additionally, grey SMC checks if a candidate is indeed a BSCC once every state of the candidate is seen at least twice. In all our tables, 'TO' denotes a timeout of 15 min and 'OOM' indicates that the tool ran out of memory restricted to 1GB RAM.

6.1 Comparison of Black and Grey SMC

Table 1 compares black SMC and grey SMC on multiple benchmarks. One can see that, except in the case of `leader6_11` and `brp_nodl`, grey SMC finishes atleast as soon as black SMC. In `bluetooth`, `gridworld`, `leader` and `tandem`, both the SMC methods are able to terminate without encountering any candidate (i.e. either the target is seen or the left side of the until formula is falsified). In `brp_nodl`, `crowds_nodl` and `nand`, the SMC methods encounter a candidate, however, since the candidate has only a single state (all BSCCs are trivial), black SMC is quickly able to confidently conclude that the candidate is indeed a BSCC. The only interesting behaviour is observed on the `herman-17` benchmark. In this case, every path eventually encounters the only BSCC existing in the model. Grey SMC is able to quickly conclude that the candidate is indeed a BSCC, while black SMC has to sample for a long time in order to be sufficiently confident.

Table 2. Effect of $p_{\min}$ on black SMC runtimes (in seconds) on some of the benchmarks. Lower $p_{\min}$ demands stronger candidates, due to which black SMC has to sample longer paths.

Model	Black SMC/$p_{\min}$				Grey SMC
	1×10^{-2}	1×10^{-3}	1×10^{-4}	1×10^{-5}	
`brp_nodl(10K,10K)`	86	93	183	TO	**84**
`crowds_nodl(8,20)`	11	41	334	TO	**9**
`egl(20,20)`	47	106	875	TO	**44**

The performance of black SMC is also a consequence of the $p_{\min}$ being quite small. Table 2 shows that black SMC is very sensitive towards $p_{\min}$. Note that grey SMC is not affected by the changes in $p_{\min}$ as it always checks whether a candidate is a BSCC as soon as all the states in the candidate are seen twice.

6.2 Grey SMC vs. Black SMC/BRTDP/BVI/VI

We now look more closely at the self-stabilization protocol `herman` [KNP12b, Her90]. The protocol works as follows: `herman-N` contains N processes, each possessing a local boolean variable x_i. A token is assumed to be in place i if $x_i = x_{i-1}$. The protocol proceeds in rounds. In each round, if the current values of x_i and x_{i-1} are equal, the next value of x_i is set uniformly at random, and otherwise it is set equal to the current value of x_{i-1}. The number of states in `herman-N` is therefore 2^N. The goal of the protocol is to reach a stable state where there is exactly one token in place. For example, in case of `herman-5`, a stable state might be $(x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 1)$, which indicates that there is a token in place 2. In every `herman` model, all stable states belong to the single BSCC. The number of states in the BSCC range from 10 states in `herman-5` to 2,000,000 states in `herman-21`.

For all `herman` models in Table 3, we are interested in checking if the probability of reaching an unstable state where there is a token in places 2–5, i.e. $(x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1, x_5 = 1)$ is less than 0.05. This property, which we name `4tokens`, identifies 2^{N-5} states as target in `herman-N`. The results in Table 3 show how well grey SMC scales when compared to black SMC, BRTDP, BVI[4] and VI. Black SMC times out for all models where $N \geq 11$. This is due to the fact that the larger models have a smaller $p_{\min}$, thereby requiring black SMC to sample extremely long paths in order to confidently identify candidates as BSCCs. BVI and VI perform well on small models, but as the model sizes grow and transition probabilities become smaller, propagating values becomes extremely slow. Interestingly, we found that in both grey SMC and black SMC, approximately 95% of the time is spent in computing the next transitions, which

[4] We refrain from comparison to other guaranteed VI techniques such as sound VI [QK18] or optimistic VI [HK19] as the implementations are not PRISM-based and hence would not be too informative in the comparison.

Table 3. Runtime (in seconds) of the various algorithms on the Herman self-stabilization protocol [KNP12b] with the property `4tokens`. The median runtimes are reported for grey SMC, black SMC [DHKP17], BRTDP [BCC+14], Bounded value iteration (BVI) and Value iteration (VI). The SMC algorithms use SPRT method with parameters $\alpha = 0.01$ and $\beta = 0.01$. BRTDP, BVI and VI run until a relative error of 0.01 is obtained.

Model	States	Grey SMC	Black SMC	BRTDP	BVI	VI
`herman-5`	32	11	15	TO	1	1
`herman-7`	128	12	57	TO	1	1
`herman-9`	512	10	775	TO	1	1
`herman-11`	2048	19	TO	TO	1	1
`herman-13`	8192	18	TO	TO	1	1
`herman-15`	33K	17	TO	TO	9	3
`herman-17`	131K	49	TO	TO	98	21
`herman-19`	524K	252	OOM	TO	602	113
`herman-21`	2M	759	OOM	OOM	TO	TO

grow exponentially in number; an improvement in the simulator implementation can possibly slow down the blow up in run time, allowing for a fairer comparison with the extremely performant symbolic value iteration algorithms.

Finally, we comment on the exceptionally poor performance of BRTDP on `herman` models. In Table 4, we run BRTDP on three different properties: (i) tokens in places 2–3 (`2tokens`); (ii) tokens in places 2–4 (`3tokens`); and (iii) tokens in places 2–5 (`4tokens`). The number of states satisfying the property decrease when going from 2 tokens to 4 tokens. The table shows that BRTDP is generally better in situations where the target set is larger.

In summary, the experiments reveal the following:

– For most benchmarks, black SMC and grey SMC perform similar, as seen in Table 1. As expected, the advantages of grey SMC do not show up in these examples, which (almost all) contain only trivial BSCCs.
– The advantage of grey SMC is clearly visible on the `herman-N` benchmarks, in which there are non-trivial BSCCs. Here, black SMC quickly fails while grey SMC is extremely competitive.
– Classical techniques such as VI and BVI fail when either the model is too large or the transition probabilities are too small. However, they are still to be used for strongly connected systems, where the whole state space needs to be analysed for every run in both SMC approaches, but only once for (B)VI.

Table 4. Effect of restrictive properties (satisfied in fewer states) on the runtime (in seconds) of BRTDP in the herman benchmarks.

Model	Property		
	`2tokens`	`3tokens`	`4tokens`
`herman-5`	1	2	TO
`herman-7`	1	1	TO
`herman-9`	1	2	TO
`herman-11`	2	2	TO
`herman-13`	2	2	TO
`herman-15`	3	3	TO
`herman-17`	5	6	TO
`herman-19`	9	9	TO
`herman-21`	104	111	TO
`herman-23`	OOM	OOM	OOM

7 Conclusion

While SMC has found its use also in the white-box setting as a scalable alternative, we introduce the first approach that utilizes the knowledge in a local way, without globally processing the state space, and thus preserves the efficiency advantages of black-box SMC. We call this approach grey SMC since we utilize only the topological information and not the quantitative information (sometimes referred to as grey box). On the one hand, this is useful as the quantitative information is often unavailable or imprecise w.r.t. the modelled reality. On the other hand, while the full quantitative information is irrelevant in BSCCs, it plays a major role in the transient phase and could be used to further enhance the approach. For instance, it could be used for importance sampling in order to handle rare events efficiently [JLS12,BDH17] even in the context of unbounded-horizon properties.

References

[AKW19] Ashok, P., Křetínský, J., Weininger, M.: PAC statistical model checking for Markov decision processes and stochastic games. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 497–519. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_29

[BBS95] Barto, A.G., Bradtke, S.J., Singh, S.P.: Learning to act using real-time dynamic programming. Artif. Intell. **72**(1–2), 81–138 (1995)

[BCC+14] Brázdil, T., et al.: Verification of Markov decision processes using learning algorithms. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 98–114. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11936-6_8

[BCH+97] Baier, C., Clarke, E.M., Hartonas-Garmhausen, V., Kwiatkowska, M., Ryan, M.: Symbolic model checking for probabilistic processes. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 430–440. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63165-8_199

[BDH17] Budde, C.E., D'Argenio, P.R., Hartmanns, A.: Better automated importance splitting for transient rare events. In: Larsen, K.G., Sokolsky, O., Wang, J. (eds.) SETTA 2017. LNCS, vol. 10606, pp. 42–58. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69483-2_3

[BK08] Baier, C., Katoen, J.-P.: Principles of Model Checking. MIT Press, Cambridge (2008)

[BT89] Bertsekas, D.P., Tsitsiklis, J.N.: Parallel and Distributed Computation: Numerical Methods. Prentice-Hall Inc., Upper Saddle River (1989)

[CH08] Chatterjee, K., Henzinger, T.A.: Value iteration. In: Grumberg, O., Veith, H. (eds.) 25 Years of Model Checking. LNCS, vol. 5000, pp. 107–138. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69850-0_7

[DHKP16] Daca, P., Henzinger, T.A., Křetínský, J., Petrov, T.: Faster statistical model checking for unbounded temporal properties. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 112–129. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_7

[DHKP17] Daca, P., Henzinger, T.A., Kretínský, J., Petrov, T.: Faster statistical model checking for unbounded temporal properties. ACM Trans. Comput. Log. **18**(2), 12:1–12:25 (2017)

[Her90] Herman, T.: Probabilistic self-stabilization. Inf. Process. Lett. **35**(2), 63–67 (1990)

[HJB+10] He, R., Jennings, P., Basu, S., Ghosh, A.P., Wu, H.: A bounded statistical approach for model checking of unbounded until properties. In: ASE, pp. 225–234 (2010)

[HK19] Hartmanns, A., Kaminski, B.L.: Optimistic value iteration. CoRR, abs/1910.01100 (2019)

[HM18] Haddad, S., Monmege, B.: Interval iteration algorithm for MDPs and IMDPs. Theor. Comput. Sci. **735**, 111–131 (2018)

[Hoe63] Hoeffding, W.: Probability inequalities for sums of bounded random variables. J. Am. Stat. Assoc. **58**(301), 13–30 (1963)

[JLS12] Jegourel, C., Legay, A., Sedwards, S.: Cross-entropy optimisation of importance sampling parameters for statistical model checking. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 327–342. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_26

[JSD19] Jégourel, C., Sun, J., Dong, J.S.: Sequential schemes for frequentist estimation of properties in statistical model checking. ACM Trans. Model. Comput. Simul. **29**(4), 25:1–25:22 (2019)

[KM19] Křetínský, J., Meggendorfer, T.: Of cores: a partial-exploration framework for Markov decision processes. (2019, Submitted)

[KNP11] Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47

[KNP12a] Kwiatkowska, M.Z., Norman, G., Parker, D.: The PRISM benchmark suite. In: QEST, pp. 203–204. IEEE Computer Society (2012)

[KNP12b] Kwiatkowska, M.Z., Norman, G., Parker, D.: Probabilistic verification of Herman's self-stabilisation algorithm. Formal Asp. Comput. **24**(4–6), 661–670 (2012). https://doi.org/10.1007/s00165-012-0227-6

[Kře16] Křetínský, J.: Survey of statistical verification of linear unbounded properties: model checking and distances. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9952, pp. 27–45. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47166-2_3

[LP08] Lassaigne, R., Peyronnet, S.: Probabilistic verification and approximation. Ann. Pure Appl. Logic **152**(1–3), 122–131 (2008)

[MLG05] Mcmahan, H.B., Likhachev, M., Gordon, G.J.: Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In: ICML 2005, pp. 569–576 (2005)

[Pnu77] Pnueli, A.: The temporal logic of programs. In: FOCS, pp. 46–57 (1977)

[Put14] Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, Hoboken (2014)

[QK18] Quatmann, T., Katoen, J.-P.: Sound value iteration. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 643–661. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_37

[RP09] El Rabih, D., Pekergin, N.: Statistical model checking using perfect simulation. In: Liu, Z., Ravn, A.P. (eds.) ATVA 2009. LNCS, vol. 5799, pp. 120–134. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04761-9_11

[SVA04] Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 202–215. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27813-9_16

[Ujm15] Ujma, M.: On verification and controller synthesis for probabilistic systems at runtime. Ph.D. thesis, University of Oxford, UK (2015)

[Val84] Valiant, L.G.: A theory of the learnable. Commun. ACM **27**(11), 1134–1142 (1984)

[YCZ10] Younes, H.L.S., Clarke, E.M., Zuliani, P.: Statistical verification of probabilistic properties with unbounded until. In: Davies, J., Silva, L., Simao, A. (eds.) SBMF 2010. LNCS, vol. 6527, pp. 144–160. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19829-8_10

[YS02] Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 223–235. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45657-0_17

E Satisfiability Bounds for ω-Regular Properties in Bounded-Parameter Markov Decision Processes (CDC 2019)

This is an exact reprinting of the accepted version of the following paper which has been published as a **peer reviewed conference paper**.

> Maximilian Weininger, Tobias Meggendorfer, and Jan Kretinsky. "Satisfiability Bounds for ω-Regular Properties in Bounded-Parameter Markov Decision Processes". In: CDC. IEEE, 2019, pp. 2284–2291. doi: 10.1109/CDC40024.2019.9029460

Summary

We consider the problem of computing minimum and maximum probabilities of satisfying an ω-regular property in a bounded-parameter Markov decision process (BMDP). BMDP arise from Markov decision processes (MDP) by allowing for uncertainty on the transition probabilities in the form of intervals where the actual probabilities are unknown. ω-regular languages form a large class of properties, expressible as e.g. Rabin or parity automata, encompassing rich specifications such as linear temporal logic. In a BMDP the probability to satisfy the property depends on the unknown transitions probabilities as well as on the policy. In this paper, we compute the extreme values. This solves the problem specifically suggested by Dutreix and Coogan in CDC 2018, extending their results on interval Markov chains with no adversary. The main idea is to reinterpret their work as analysis of interval MDP and accordingly the BMDP problem as analysis of an ω-regular stochastic game, where a solution is provided. This method extends smoothly further to bounded-parameter stochastic games.

For more details on this publication, we refer to Section 4.2 of the main body.

Contribution

Contribution of Maximilian Weininger	
Development and conceptual design of the research project	60%
Discussion and development of ideas	60%
Composition and revision of the manuscript	40%
Proofs	35%
Case study	75%

Home | Help | Live Chat | Sign in | Create Account

Satisfiability Bounds for ω-Regular Properties in Bounded-Parameter Markov Decision Processes

Conference Proceedings: 2019 IEEE 58th Conference on Decision and Control (CDC)
Author: Maximilian Weininger
Publisher: IEEE
Date: Dec. 2019

Copyright © 2019, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK CLOSE WINDOW

Satisfiability Bounds for ω-Regular Properties in Bounded-Parameter Markov Decision Processes

Maximilian Weininger[*] Tobias Meggendorfer[*] Jan Křetínský[*]
Department of Informatics, Technical University of Munich

***Abstract*— We consider the problem of computing minimum and maximum probabilities of satisfying an ω-regular property in a bounded-parameter Markov decision process (BMDP). BMDP arise from Markov decision processes (MDP) by allowing for uncertainty on the transition probabilities in the form of intervals where the actual probabilities are unknown. ω-regular languages form a large class of properties, expressible as, e.g., Rabin or parity automata, encompassing rich specifications such as linear temporal logic. In a BMDP the probability to satisfy the property depends on the unknown transitions probabilities as well as on the policy. In this paper, we compute the extreme values. This solves the problem specifically suggested by Dutreix and Coogan in CDC 2018, extending their results on interval Markov chains with no adversary. The main idea is to reinterpret their work as analysis of interval MDP and accordingly the BMDP problem as analysis of an ω-regular stochastic game, where a solution is provided. This method extends smoothly further to bounded-parameter stochastic games.**

I. Introduction

Markov decision processes (MDP) are a classical formalism encompassing both probabilistic and non-deterministic features: in each state some actions are enabled and each action is assigned a distribution over the successor states. In other words, each action corresponds to a set of transitions, each of which is assigned a transition probability. *Bounded-parameter MDP (BMDP)* [16] are like MDP, but to each transition is instead assigned an interval of possible transition probabilities. Thus each BMDP specifies a set of MDP. There are two interpretations of these intervals. Firstly, in the *uncertain* interpretation, the BMDP specifies MDP with unknown but constant transition probabilities in the intervals. An MDP is thus derived from the BMDP by picking a value in each interval once for all. Secondly, in the *adversarial* interpretation, the BMDP specifies a decision process where the transition probabilities may be different numbers (in the intervals) every time we come to a state. Each interpretation found its use. The former can model, for instance, various degrees of uncertainty for each action or confidence intervals for the transition probabilities learnt from experience. In this case there is one true transition probability, however unknown. The latter can be used as a formalism for abstracting MDP: states with different outgoing transition probabilities can be abstracted into a single state with an interval covering all the values [16]. In this case, the interval can stand for any of the values whenever visiting the state. As such BMDP extend interval Markov chains (IMC) [21], [23] by an adversary (or an underspecified/non-deterministic controller). The uncertain interpretation of IMC then yields uncertain Markov chains (UMC), while the adversarial interpretation of IMC yields interval MDP (IMDP), as distinguished in [33].

ω-regular languages, e.g. [36], form a robust class of rich specifications, which can be represented in various ways, e.g., by formulae of monadic second-order logic or by automata over infinite words. In the setting of probabilistic systems, it is often advantageous to use deterministic Rabin automata (DRA) or their variations. In particular, this class encompasses properties expressible in linear temporal logic (LTL) [27] and there are efficient ways of translating LTL to DRA [24]. Control of MDP with LTL specifications is widely studied, e.g. [35], [22], [38], [30], [40], and typically uses the DRA representation.

In [13], Dutreix and Coogan argue for computing minimum and maximum probabilities of satisfying an ω-regular property in an IMC interpreted as IMDP. In future work, they wish to apply the technique to solve the problem for BMDP, the controllable counterpart of IMC. In this paper, we re-interpret their technique in a different light and using that perspective give a solution to BMDP, in both the uncertain and the adversarial understanding of the intervals. We consider both the upper bound (also called *design choice* of values in intervals [13]) and the lower bound (*antagonistic* in [13]). We present the results for controllers that try to maximize the probability to satisfy the ω-regular property; minimization is analogous as ω-regular languages are closed under complement.

The main idea of our approach is to not only view the IMC as an IMDP, but also as an MDP, since an IMDP is a special case of MDP where the adversary chooses the transition probabilities. We show the standard analysis on the respective MDP coincides with the tailored algorithm of [13] applied to the IMC. Our main contribution is taking this perspective on BMDP, yielding a stochastic game. Since we can solve the stochastic game with an ω-regular objective we can obtain also the solutions. Moreover, for the upper bound, the two players play cooperatively and we can solve the problem in polynomial time using adapted MDP techniques. Finally, we show how the game extension of IMC with two competing agents can be solved analogously to BMDP, this time without the need of introducing an additional agent.

[*]All authors contributed equally to this work. It was funded in part by the German Research Foundation (DFG) project KR 4890/2-1 "Statistical Unbounded Verification".

Further related work: The general model of MDP with imprecise parameters (MDPIP) was introduced in [32]. BMDP [16] are then a subclass where the parametrization is limited to independent intervals. BMDP have been investigated with respect to various objectives, such as stochastic shortest path (minimum expected reward) [4], expected total reward [26], [39], [18], discounted reward [26], [15], LTL [37], PCTL [28], [19], or mean payoff [34].

The special non-controlled case of IMC has also been investigated for various objectives, e.g. PCTL [33], [5], LTL [3] ([2] observes this algorithm may not converge to the optimum) or ω-regular properties [9].

Recent improvements include importance sampling techniques for IMC [20] or topological policy iteration for BMDP [31]. IMC and BMDP are used as abstractions of systems in [25].

Organization of the paper: After recalling the used formalisms in Section II, we state our problem in Section III. We provide the solution in Section IV and an illustrative case study (adjusted from [13]) in Section V. Section VI concludes and presents ideas for future work.

II. Preliminaries

In this section, we recall basics of probabilistic systems and set up the notation. As usual, $\mathbb{N}$ refers to the natural numbers (including 0). A *probability distribution* on a finite set X is a mapping $p : X \to [0, 1]$, such that $\sum_{x \in X} p(x) = 1$. We use $\mathcal{D}(X)$ to denote the set of all probability distributions on the set X. Given some set S, we use $S^\star$ and S^ω to denote the set of all finite and infinite sequences comprising elements of S, respectively.

A. Markov Decision Processes

Definition 1 *A* Markov decision process (MDP) *is a tuple $\mathcal{M} = (S, s_0, A, \mathsf{Av}, \Delta, Acc)$, where S is a finite set of* states, *$s_0 \in S$ is the* initial *state, A is a finite set of* actions, *$\mathsf{Av} : S \to 2^A \setminus \{\emptyset\}$ assigns to every state a non-empty set of* available actions, *$\Delta : S \times A \to \mathcal{D}(S)$ is a* transition function *that for each state s and (available) action $a \in \mathsf{Av}(s)$ yields a probability distribution over successor states, and $Acc \in 2^S \times 2^S$ is the* Rabin acceptance. *Furthermore, for ease of notation we assume w.l.o.g. that actions are unique for each state, i.e. $\mathsf{Av}(s) \cap \mathsf{Av}(s') = \emptyset$ for $s \neq s'$.*[1] *An element $(F_i, I_i) \in Acc$ is called* Rabin pair. *We assume w.l.o.g. that $F_i \cap I_i = \emptyset$ for all pairs.*

In figures, we denote a Rabin pair (F, I) by $F\,I$.

An MDP with $|\mathsf{Av}(s)| = 1$ for all $s \in S$ is called *Markov chain (MC)*. For ease of notation, we overload functions that map to distributions $f : Y \to \mathcal{D}(X)$ by $f : Y \times X \to [0, 1]$, where $f(y, x) := f(y)(x)$. For example, instead of $\Delta(s, a)(s')$ we write $\Delta(s, a, s')$ for the probability of transitioning from state s to s' using action a.

An *infinite path* in an MDP is an infinite sequence $\rho = s_0 a_0 s_1 a_1 \cdots$, such that $a_i \in \mathsf{Av}(s_i)$ and $\Delta(s_i, a_i, s_{i+1}) > 0$ for every $i \in \mathbb{N}$. We use ρ_i to refer to the i-th state s_i in a given path. A *finite path* is a finite prefix of an infinite path. $\mathrm{Inf}(\rho) \subseteq S$ denotes the set of all states which are visited infinitely often in the path ρ.

A path ρ is *accepted*, denoted $\rho \models Acc$, if an only if there exists a Rabin pair $(F_i, I_i) \in Acc$ such that all states in F_i are visited *finitely often*, i.e. $F_i \cap \mathrm{Inf}(\rho) = \emptyset$, and at least one state of I_i is visited *infinitely often*, i.e. $I_i \cap \mathrm{Inf}(\rho) \neq \emptyset$. We call such a Rabin pair *accepting for ρ*.

Remark 1 *Often, system and specification are modelled separately, e.g., by a labelled MDP together with a description of an ω-regular property such as an LTL formula [27] or an automaton. The common approach then is to build the product of the system, the BMDP, and an automaton describing the specification; this results in a system as described in Definition 1. Since our work is largely independent of this construction's details we omit this step for simplicity. We highlight that indeed [13] only refers to the product throughout the main body of their work. Details can be found in Section VII and, e.g., [1].*

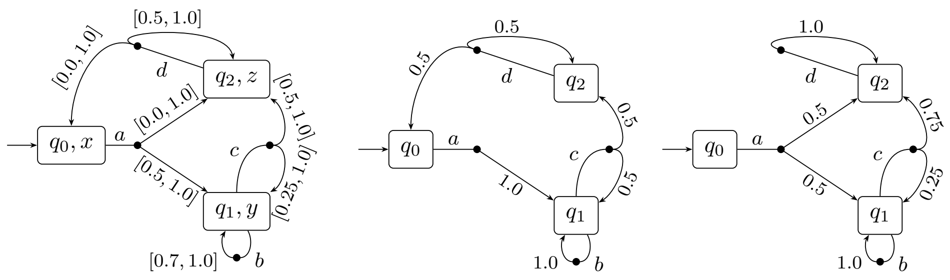
A *policy* (also called *controller*, *strategy*) on an MDP is a function $\pi : (S \times A)^* \times S \to \mathcal{D}(A)$ which given a finite path $\varrho = s_0 a_0 s_1 a_1 \dots s_n$ yields a probability distribution $\pi(\varrho) \in \mathcal{D}(\mathsf{Av}(s_n))$ on the actions to be taken next. We call a policy *memoryless randomized* (or *stationary*) if it is of the form $\pi : S \to \mathcal{D}(A)$, and *memoryless deterministic* (or *positional*) if it is of the form $\pi : S \to A$. Later in the paper, we prove that positional strategies are indeed sufficient for all considered problems. We denote the set of all policies of an MDP by Π. By fixing the policy π in an MDP $\mathcal{M}$, we naturally obtain a MC and thus a probability measure $\mathbb{P}^\pi_\mathcal{M}$ over potential runs [29]. Throughout this work, we are interested in finding policies which maximize the probability of accepting runs, i.e. $\sup_{\pi \in \Pi} \mathbb{P}^\pi_\mathcal{M} [\rho \models Acc]$.

An *end component* in an MDP is a pair (T, A) of a set of states T and a set of actions A such that the system *can* remain within the states T indefinitely, using only actions from A. Formally, a pair (T, A), where $\emptyset \neq T \subseteq S$ and $\emptyset \neq A \subseteq \bigcup_{s \in T} \mathsf{Av}(s)$, is an end component of an MDP $\mathcal{M}$ if (i) for all $s \in T, a \in A \cap \mathsf{Av}(s)$ we have $\{s' \mid \Delta(s, a, s') > 0\} \subseteq T$, and (ii) for all $s, s' \in T$ there is a finite path $\varrho = s a_0 \dots a_n s' \in (T \times A)^* \times T$, i.e. the path stays inside T and only uses actions in A. An end component (T, A) is a *maximal end component (MEC)* if there is no other end component (T', A') such that $T \subseteq T'$ and $A \subseteq A'$. The set of MECs of an MDP $\mathcal{M}$ is denoted by $\mathsf{MEC}(\mathcal{M})$ and can be obtained in polynomial time [12]. For further detail, see [1, Sec. 10.6.3].

B. Bounded-parameter Markov Decision Processes

Definition 2 *A* bounded-parameter Markov decision process (BMDP) *is a tuple $\mathfrak{M} = (S, s_0, A, \mathsf{Av}, \check{\Delta}, \widehat{\Delta}, Acc)$, where S, s_0, A, Av and Acc are as in the definition of MDP, and $\check{\Delta}, \widehat{\Delta} : S \times A \times S \to [0, 1]$ are* lower and upper bounds *on the transition probability in each state. Again, we assume that actions are unique for each state.*

[1]The usual procedure to achieve this in general is to replace A by $S \times A$ and to adapt Av and Δ appropriately.



(a) Example BMDP (b) MDPs consistent with the BMDP

Fig. 1. Examples of an BMDP and its instantiations. The acceptance is $Acc = \{(q_2\, q_1)\}$.

For consistency, we require that $\sum_{s' \in S} \check{\Delta}(s, a, s') \leq 1 \leq \sum_{s' \in S} \widehat{\Delta}(s, a, s')$ *for each state* s *and action* $a \in \mathsf{Av}(s)$.

Given a BMDP $\mathfrak{M} = (S, s_0, A, \mathsf{Av}, \check{\Delta}, \widehat{\Delta}, Acc)$, we call a Markov decision process $\mathcal{M} = (S, s_0, A, \mathsf{Av}, \Delta, Acc)$ *consistent with* $\mathfrak{M}$, denoted $\mathcal{M} \in \mathfrak{M}$, if and only if $\mathcal{M}$'s transition probabilities satisfy $\mathfrak{M}$'s bounds, i.e. $\check{\Delta}(s, a, s') \leq \Delta(s, a, s') \leq \widehat{\Delta}(s, a, s')$ for all states $s, s' \in S$ and actions $a \in \mathsf{Av}(s)$. Note that in general there are uncountably many MDPs consistent with a BMDP $\mathfrak{M}$. A BMDP with $|\mathsf{Av}(s)| = 1$ for all $s \in S$ is called *Interval Markov chain (IMC)* (e.g., [9]).

See Fig. 1a for an example BMDP and Fig. 1b for two MDPs consistent with this BMDP.

C. Stochastic Games

For our analysis, we additionally need the concept of *stochastic games*. These can be understood as an MDP where, instead of a single agent controlling the process, we have two antagonistic players. Intuitively, the first player's aim is to obtain an accepted path, while the second player aims to stop the first player from doing so. Each state in the game is "owned" by one of the two players and the owner of a particular state can decide which action to take in that state.

Definition 3 *A* stochastic game (SG) *is a tuple* $\mathcal{G} = (S, s_0, A, \mathsf{Av}, \Delta, Acc, \mathcal{O})$*, where* $(S, s_0, A, \mathsf{Av}, \Delta, Acc)$ *is an MDP and* $\mathcal{O} : S \to \{1, 2\}$ *is an* ownership function*, assigning each state to either player* 1 *or player* 2*. This naturally gives rise to the sets of states* S_1 *and* S_2*, which are controlled by the respective player.*

The definitions of (in)finite paths directly extend to stochastic games. Policies are slightly modified, since each player can only make decisions in a part of the game. Formally, we have two kinds of policies $\pi_1 : (S \times A)^* \times S_1 \to \mathcal{D}(A)$ and $\pi_2 : (S \times A)^* \times S_2 \to \mathcal{D}(A)$, one for each player.We denote the set of all policies of the respective players by Π_1 and Π_2. Fixing the policy of a single player yields an MDP, denoted $\mathcal{G}(\pi_i)$; fixing both players' policies π_1 and π_2 again yields an MC and measure over the set of runs, denoted $\mathbb{P}_{\mathcal{G}}^{\pi_1, \pi_2}$ [11].

III. Problem Statement

In this work, we are given a BMDP and want to control it such that the probability of an accepting run is maximized. This raises two orthogonal questions:

Firstly, the semantics of the interval constraints have to be fixed. We consider two different popular interpretations, called *uncertain* and *antagonistic*. In the *uncertain* (or *design-choice*) model, an external environment fixes the transition probabilities once and for all, i.e. for a BMDP $\mathfrak{M}$ a particular consistent MDP $\mathcal{M} \in \mathfrak{M}$ is chosen. In the *antagonistic* model, the external environment instead is allowed to change the transition probabilities at every step, taking into account the full path so far. These interpretations have been shown to be yield the same optima for reachability in interval Markov chains [10]. In the following, we show that this also is the case for BMDP with Rabin objectives; hence we do not distinguish the semantics in our formal problem statement.

Secondly, it is not specified whether the aforementioned environment is cooperative or antagonistic. We consider both of these two extreme cases. In particular, we are interested in finding the maximal probability of acceptance while assuming that all transition probabilities are chosen (i) to our liking and (ii) as bad as possible.

Formally, given a BMDP $\mathfrak{M}$ we want to compute

(i) $\widehat{\mathbb{P}}(\mathfrak{M}) = \sup_{\pi \in \Pi} \sup_{\mathcal{M} \in \mathfrak{M}} \mathbb{P}_{\mathcal{M}}^{\pi} [\rho \models Acc]$, and
(ii) $\check{\mathbb{P}}(\mathfrak{M}) = \sup_{\pi \in \Pi} \inf_{\mathcal{M} \in \mathfrak{M}} \mathbb{P}_{\mathcal{M}}^{\pi} [\rho \models Acc]$.

Further, we are interested in the optimal policy and the best- / worst-case MDP consistent with the given BMDP, if it exists, i.e. the witnesses for the above values.

Case (i) can be understood as a "design challenge". We are interested in building our system, i.e. finding an optimal assignment for all transitions, such that we maximize the probability of acceptance. On the other hand, Case (ii) can be thought of as uncertainty about the real world or measurement imprecisions. Here, we rather are interested in optimizing the worst case and want to find a safe strategy, able to reasonably cope with any concrete instantiation of the intervals.

Consider the situation depicted in Fig. 1. We are interested in finding the upper and lower bounds for the BMDP given in Fig. 1a. The upper bound, 1.0, is exhibited by the left MDP of Fig. 1b. There, we end up in q_1 with probability 1 and, by always playing action b, get an accepting run with probability 1. The right MDP shows the lower bound of 0.5, since we get stuck in state q_2 with probability 0.5. Observe that if action d in state q_2 had a non-zero probability of moving to q_0, the resulting run would be accepting with probability 1, since we would almost surely eventually reach q_1.

In the following, we re-interpret existing approaches, and present our unified approach for solving BMDP.

IV. Solution approach

In order to explain our approach, we first shed some light on the simpler case of interval Markov chains (IMC), handled in [13], and provide a different perspective on their approach.

In [13], the authors present a specialized algorithm for dealing with IMCs. In essence, they compute maximal sets of (non)accepting states and then obtain the final value by solving a reachability query for these states. We now provide a different viewpoint on their algorithm which will help us solve the more general case of BMDPs.

The key observation is the following. We can view the intervals in an IMC as a player (the external environment) picking the transition probabilities at every step. This can be understood as an MDP, where in every state the player has an action for each distribution satisfying the interval constraints. This MDP has uncountably many actions in general, as there are infinitely many possibilities to choose the probabilities. However, all possible distributions can be constructed as a convex combination of finitely many special cases which are *basic feasible solutions* (BFS) of a linear program [9], [17] (also known as *corner-point abstraction*). We can identify each of these special solutions and use them to construct a finite MDP. This MDP is often called *interval Markov decision process* (IMDP); not to be confused with BMDP. By model checking the IMDP, using established methods, we obtain the desired result for the IMC.

Indeed, the algorithm presented in [13, Sec. IV-C] actually can be interpreted as a symbolic adaption of the standard model checking procedure for Rabin objectives on the MDP, namely an adapted MEC decomposition together with a reachability query [1, Sec. 10.6.4]. Similar to the methods presented in [17], their specialized algorithm cleverly avoids explicit computation of the exponentially large IMDP, achieving polynomial runtime.

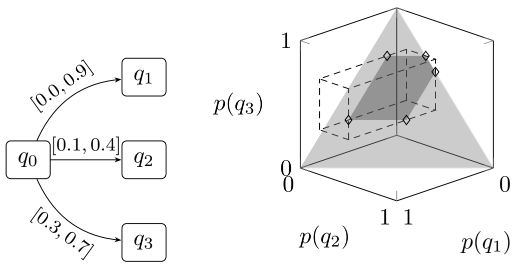
Before we can extend these ideas to BMDP, we explain some details of the basic feasible solutions, since they are essential to our idea.

A. Basic feasible solutions

Given an IMC $(S, s_0, \check{\Delta}, \widehat{\Delta}, \lambda)$ and a state $s \in S$, we are interested in the set of all successor distributions $p \in \mathcal{D}(S)$ consistent with the constraints imposed by the IMC. In particular, the following constraints have to be satisfied:

(I) $\sum_{s' \in S} p(s') = 1$, and
(II) $\check{\Delta}(s, s') \leq p(s') \leq \widehat{\Delta}(s, s')$ for all $s' \in S$.

Geometrically, Item I constrains the solution set to a plane, while Item II bounds it in a box, as shown in Fig. 2. Since each point in the solution corresponds to a valid transition distribution and vice versa, we identify solutions of the constraints with their corresponding distributions. Observe that the resulting solution set



(a) Example IMC (b) Visualized constraints.

Fig. 2. Visualization of the set of basic feasible solutions. The left picture shows a state together with its transition constraints in an IMC, the right picture depicts a geometric representation of the inequalities induced by the interval constraints. The light grey plane corresponds to the distribution constraints (Item I), while the dashed box represents the interval constraints (Item II). Finally, the dark grey area shows the set of consistent distributions and the diamonds mark the basic feasible solutions.

(and the corresponding set of distributions) is convex and any element of this set can be obtained by a convex combination of the corner-points, which are the basic feasible solutions.

There is one dimension per successor state and thus at most exponentially many basic feasible solutions. This set can be computed in exponential time using standard theory of linear programs or simple geometric computations. Due to lack of space we refer the reader to, e.g., [17, Sec. 4] for further detail.

B. Solving BMDP

When solving Rabin objectives on BMDPs, we observe one central problem: The set of end components depends on the choice of intervals. For example, consider the BMDP in Fig. 1a. This BMDP comprises only one MEC containing all three states. However, depending on the choice of probabilities, edges may be removed from the underlying graph, as for example in the right MDP of Fig. 1b. This MDP contains two MECs, namely $(\{q_1\}, \{b\})$ as well as $(\{q_2\}, \{d\})$.

In [13], a similar problem was encountered, as in IMC the set of *bottom strongly connected components* can be modified by the choice of intervals. The authors solved the problem implicitly by considering states where the Rabin objective is not satisfied as "leaky", i.e. not part of any strongly connected component.

In contrast to that, our general solution relies on making the possible choices of the probabilities explicit. We utilize the key observation that the non-determinism induced by intervals essentially corresponds to adding a player, who picks the probabilities for the intervals. Thus, we reduce the BMDP to a stochastic game that can be solved by known model checking methods. However, in the next section we introduce a more sophisticated approach inspired by the ideas of [13].

Intuitively, the reduction from BMDP to SG amounts to replacing every action in the BMDP with an additional state owned by the new player. In this state, the new player can choose from the corresponding basic feasible solutions, allowing him to construct any consistent

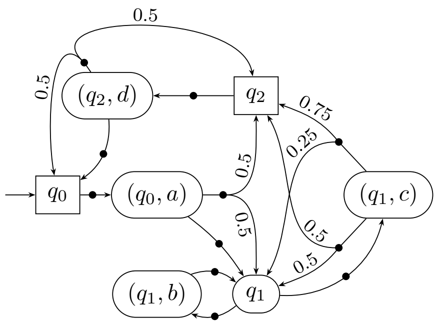


Fig. 3. The stochastic game obtained from the BMDP in Fig. 1a by the construction in the proof of Theorem 2. States owned by the system are depicted by rectangles, while environment states have rounded corners. For readability, we omit all action labels. Furthermore, we omit action nodes for probability 1 transitions.

distribution over the successors. An example of this construction is shown in Fig. 3, where the game constructed from the BMDP in Fig. 1a is depicted. This new player can play against the system (yielding $\check{\mathbb{P}}$) or cooperate (yielding $\widehat{\mathbb{P}}$).

Theorem 1 *For every BMDP $\mathfrak{M}$, there exists an SG $\mathcal{G}(\mathfrak{M})$ such that*

(i) $\widehat{\mathbb{P}}(\mathfrak{M}) = \sup_{\pi_1 \in \Pi_1} \sup_{\pi_2 \in \Pi_2} \mathbb{P}^{\pi_1,\pi_2}_{\mathcal{G}(\mathfrak{M})}[\rho \models Acc]$*, and*

(ii) $\check{\mathbb{P}}(\mathfrak{M}) = \sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} \mathbb{P}^{\pi_1,\pi_2}_{\mathcal{G}(\mathfrak{M})}[\rho \models Acc]$

Proof: Let $\mathfrak{M} = (S, s_0, A, \mathsf{Av}, \check{\Delta}, \widehat{\Delta}, \lambda)$ be a BMDP. In contrast to IMC, where we found the basic feasible solutions for a state, in BMDP we have to consider state-action pairs. Recall that we assumed that each action belongs to a single state. Hence, each $a \in A$ induces a set of basic feasible solutions, given by the constraints $\check{\Delta}(s,a)$ and $\widehat{\Delta}(s,a)$, where s is the unique state with $a \in \mathsf{Av}(s)$. We denote this set by $\mathsf{BFS}(a)$, and it can be computed as described in Section IV-A. Recall that any $p \in \mathsf{BFS}(a)$ corresponds to a distribution over states.

The SG $\mathcal{G}(\mathfrak{M}) = (S', s_0, A', \mathsf{Av}', \Delta, Acc, \mathcal{O})$ is constructed from $\mathfrak{M}$ as follows.

- $S' = S \cup \{(s,a) \mid s \in S \wedge a \in \mathsf{Av}(s)\}$,
- $A' = A \cup \{(a,p) \mid a \in A, p \in \mathsf{BFS}(a)\}$, and
- $\mathcal{O}(s) = 1$ if $s \in S$ and 2 otherwise, i.e. all newly introduced states (s,a) belong to the environment.
- For every old state $s \in S$ we set
  - $\mathsf{Av}'(s) = \mathsf{Av}(s)$ and
  - $\Delta(s, a, (s,a)) = 1$ (and 0 for all other states) for all actions $a \in \mathsf{Av}(s)$.
- For every new state $(s,a) \in S' \setminus S$ we set
  - $\mathsf{Av}'((s,a)) = \{(a,p) \mid p \in \mathsf{BFS}(a)\}$ and
  - $\Delta((s,a),(a,p),s') = p(s')$ for all $p \in \mathsf{BFS}(a)$.

For any consistent MDP $\mathcal{M} \in \mathfrak{M}$, there exists a policy for the other player π_2 inducing the transition function of $\mathcal{M}$, i.e. for all $\pi_1 \in \Pi_1$ we have $\mathbb{P}^{\pi_1}_{\mathcal{M}}[\rho \models Acc] = \mathbb{P}^{\pi_1,\pi_2}_{\mathcal{G}(\mathfrak{M})}[\rho \models Acc]$, and vice versa. This can be proven completely analogously to the proof for IMC, see, e.g., [9, Thm. 8]. Intuitively, randomizing over the BFS exactly yields the set of valid distributions. This immediately yields the desired equality. ■

Corollary 1 *Positional policies suffice to achieve optimal solutions in BMDP. Consequently, the uncertain and antagonistic semantics are equivalent for the optima.*

Proof: Positional policies are sufficient for Rabin objectives in SG [8], and thus by the reduction of Theorem 1 also for BMDP. Hence the best policy for choosing the intervals is positional both for maximization and minimization, and there is no benefit in switching. ■

Remark 2 *This result relies on the fact that the objective Acc is already part of the BMDP. See Appendix VII-C for further discussion.*

To solve ω-regular objectives for SGs, we use the strategy improvement algorithm presented in [7] and the reachability algorithm for MDPs from [1], which yields both the maximum and minimum probability, as well as the optimal controller. Given a BMDP $\mathfrak{M}$, our procedure works as follows:

1) Lower bound:
   a) Use [7, Alg. 2] to solve $\mathcal{G}(\mathfrak{M})$ with the given Rabin objective *Acc*. This yields the optimal player 1 policy π_1, which induces an MDP $\mathcal{M}_{\pi_1}$. Note that the actions of the system, the original non-determinism of the BMDP, are fixed in $\mathcal{M}_{\pi_1}$, and the remaining non-determinism belongs to the environment player choosing the intervals.
   b) Compute the minimum reachability probability with the algorithm from [1, Sec. 10.6.4], to obtain the the value $\check{\mathbb{P}}$ and the worst-case environment policy π_2 for $\mathcal{M}_{\pi_1}$, and thus the worst-case instantiation of the intervals.
2) Upper bound:
   a) Let $\mathcal{M}(\mathfrak{M})$ be the MDP obtained by assigning all nodes in $\mathcal{G}(\mathfrak{M})$ to player 1.
   b) Use standard model checking procedures to obtain the best policy π for $\mathcal{M}(\mathfrak{M})$ and the value $\widehat{\mathbb{P}}$; the policy handles both the non-determinism of the system and the instantiation of the intervals.

Theorem 2 *This procedure correctly computes*

1) $\check{\mathbb{P}}(\mathfrak{M})$*, the optimal policy* π_1 *to achieve it and the worst-case MDP, and*

2) $\widehat{\mathbb{P}}(\mathfrak{M})$*, and* π*, describing the best-case system controller as well as the best choice of intervals.*

It terminates in time exponential in the size of the $\mathcal{G}(\mathfrak{M})$*, which in turn is exponential in the size of* $\mathfrak{M}$*.*

Proof: The correctness follows directly from Theorem 1 and the correctness of the the used algorithms [7, Thm. 3], [1, Sec. 10.6.4]. The complexity is dominated by the computation of the game $\mathcal{G}(\mathfrak{M})$ and its solution process. The SG is exponential in the size of the BMDP (see Section IV-A) and the solution algorithm takes time exponential in the size the game [7, Thm. 3]. ■

Remark 3 *We can immediately apply the presented methods to "bounded-parameter stochastic games", i.e. stochastic games with transition probability intervals. In* *particular, we do not need to introduce another player, as the states added by the construction in the proof of Theorem 1 can be assigned to one of the existing players – player 1 in the uncertain and player 2 in the antagonistic setting. Thus the system remains a stochastic game.*

C. Improving computation of upper bounds

Note that for the computation of the upper bounds, we did not make use of the second player, but instead only introduced new states for the existing player, yielding an exponentially large MDP. By adapting observations from [13], we can improve on this algorithm by directly analysing the MC and avoiding the explicit construction of the SG, yielding a polynomial time algorithm for computing the upper bound.

Intuitively, the improved procedure symbolically identifies for each Rabin pair in $(F, I) \in \mathit{Acc}$ the winning end components. It does so by computing MECs while temporarily excluding states in F. After obtaining all states that are in a winning end-component for some pair, we compute the probability to reach any of these states. The structure of this improved procedure is similar to [13, Alg. 1], and relies on methods to compute the MEC decomposition on a BMDP from [17] and the reachability algorithm from [28].[2]

Given a BMDP $\mathfrak{M}$, our procedure works as follows:

1) For each Rabin pair $(F_i, I_i) \in \mathit{Acc}$:
   a) Construct a modified copy $\mathfrak{M}_i$ of $\mathfrak{M}$ where all (s, q) with $q \in F_i$ absorbing, i.e. all outgoing transitions are replaced with a self-loop.
   b) Compute the MEC decomposition of the resulting BMDP using [17, Alg. 3].
   c) If a MEC (T, A) has a non-empty intersection with I_i, then it is winning and all states T are added to the set of winning states W.
2) Compute the maximal probability of reaching W, using the methods presented in, e.g., [28].

Theorem 3 *This procedure correctly computes $\widehat{\mathbb{P}}(\mathfrak{M})$ and terminates in polynomial time.*

Proof: By [17, Prop. 6], we get that the MEC computation of a BMDP $\mathfrak{M}$ through [17, Alg. 3] is correct, i.e. the computed MECs correspond to the MECs of $\mathcal{G}(\mathfrak{M})$.The MECs identified as winning in Step 1b and 1c thus indeed are winning MECs in $\mathcal{M}(\mathfrak{M})$, where $\mathcal{M}(\mathfrak{M})$ is as in Step 2a of the procedure of Section IV-B. Consequently, we exactly identify the set of potentially winning states. Finally, the correctness of the algorithms used to compute the reachability in Step 2 yields the overall correctness.

For the complexity, observe that each step requires at most polynomial time ([17, Prop. 6] for Step 1b, [28, Thm. 4.1] for Step 2) and there are only linearly many Rabin pairs in Acc. ■

Remark 4 *Note that in the setting of [13] a procedure for computing the upper bound is sufficient, as the lower bound can be computed as 1 minus the maximal probability to satisfy the negation of the specification. However, this idea is not applicable in the BMDP setting due to the alternation of* sup *and* inf *in the definition of $\check{\mathbb{P}}$.*

[2]In [17] and [28], the authors refer to BMDP as “IMDP”.

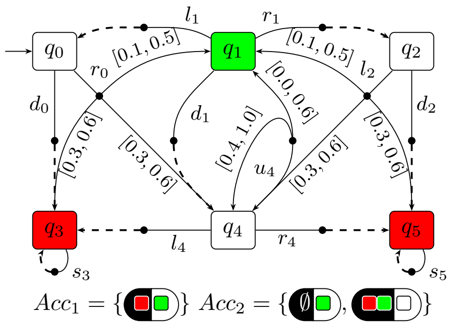


Fig. 4. Graphic representation of our extension of the case study from [13]. A robot navigates through a grid of six states according to the BMDP, using the actions **u**p, **d**own, **l**eft, and **r**ight, where enabled. For readability, all $[1.0, 1.0]$ constraints are omitted and the corresponding edges are drawn dashed. In our analysis, we consider two different acceptance conditions Acc_1 and Acc_2.

TABLE I
RESULTS FOR THE BMDP IN FIG. 4

	$\check{\mathbb{P}}$	$\widehat{\mathbb{P}}$
Acc_1	0.0	0.7
Acc_2	0.4	0.7

Remark 5 *Our algorithm can easily be extended to use* generalized Rabin transition acceptance*, allowing for efficient practical implementation. See [6] for details.*

V. Case Study

We extend the case study from [13], where an agent moves on a coloured grid of six states. We added several actions in each state, modelling a robot which navigates the grid as depicted in Fig. 4. It can choose between going down, up, left or right; it cannot leave the grid, e.g., in q_0 only down and right are available. The robot is pulled towards the red states q_3 and q_5, e.g. when moving right from q_0 there is some chance to be pulled down onto q_4 or even q_3. The strength of the pulling force and hence also the probability distribution over the successor states is unknown and hence modelled by uncertainty intervals.

As a first example, we calculate the probability to reach state q_2 starting from q_0. From q_1 we can surely reach q_2 and from q_3 and q_5 there is no possibility of reaching q_2. From q_4, it depends on the instantiation of the BMDP. The probability to remain in q_4 may equal 1, preventing the robot from reaching q_2 when it is in q_4. On the other hand, we might have $\Delta(q_4, u_4, q_1) = 0.6$ and q_1 can almost surely be reached from q_4. Consequently, q_2 can be reached from q_4 with probability 1. Finally, the best strategy in q_0 is to go right, as otherwise the robot is immediately stuck in q_3. Together, this gives us a lower bound of 0.1 and an upper bound of 0.7.

Now we consider the properties from [13], adjusted to our setting as depicted in Fig. 4. The resulting probabilities are given in Table I and explained below.

Acc_1 corresponds to "The agent visits a green state infinitely often while visiting red states finitely often". In the antagonistic interpretation, the probability of satisfying this property is zero, since there is no MEC containing q_1. Intuitively, actions r_0, d_1 and l_2 all have a positive probability of moving to the bottom row, where we may be forced to remain forever. For the upper bound, observe that by setting $\Delta(q_4, u_4, q_1) = 0.6$ we obtain the winning MEC $(\{q_1, q_4\}, \{d_1, u_4\})$, which can be reached with probability 0.7.

Acc_2 corresponds to "The agent visits a red state infinitely often only if it visits a green state infinitely often". Observe that in this case both q_4 and q_1 are winning in any consistent MDP, as playing actions d_1 and u_4 always leads to a winning path. In contrast to Acc_1, remaining in q_4 is winning by the second pair of Acc_2, since only white states are visited. We thus get a lower and upper bound of 0.4 and 0.7, respectively, by computing the probability to reach q_1 or q_4.

VI. Conclusion

We have presented a solution to the open problem of bounding the probabilities to satisfy an ω-regular property on a bounded-parameter Markov systems. A different perspective on previous approaches enabled us to solve the problem by analysis of ω-regular stochastic games. Future work includes applications of our approach to more general settings such as MDPIP, as well as a practical implementation. For the latter, we plan to apply approaches based on learning and real-time dynamic programming, see e.g. [15].

References

[1] Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

[2] Anicet Bart, Benoît Delahaye, Didier Lime, Eric Monfroy, and Charlotte Truchet. Reachability in parametric interval markov chains using constraints. In *QEST*, pages 173–189, 2017.

[3] Michael Benedikt, Rastislav Lenhardt, and James Worrell. LTL model checking of interval markov chains. In *TACAS*, pages 32–46, 2013.

[4] Olivier Buffet. Reachability analysis for uncertain ssps. In *ICTAI*, pages 515–522, 2005.

[5] Souymodip Chakraborty and Joost-Pieter Katoen. Model checking of open interval markov chains. In *ASMTA*, pages 30–42, 2015.

[6] Krishnendu Chatterjee, Andreas Gaiser, and Jan Kretínský. Automata with generalized rabin pairs for probabilistic model checking and LTL synthesis. In *CAV*, volume 8044 of *LNCS*, pages 559–575. Springer, 2013.

[7] Krishnendu Chatterjee and Thomas A. Henzinger. Strategy improvement for stochastic rabin and streett games. In *CONCUR*, volume 4137 of *LNCS*, pages 375–389. Springer, 2006.

[8] Krishnendu Chatterjee and Thomas A. Henzinger. A survey of stochastic ω-regular games. *J. Comput. Syst. Sci.*, 78(2):394–413, 2012.

[9] Krishnendu Chatterjee, Koushik Sen, and Thomas A. Henzinger. Model-checking omega-regular properties of interval markov chains. In *FOSSACS*, pages 302–317, 2008.

[10] Taolue Chen, Tingting Han, and Marta Z. Kwiatkowska. On the complexity of model checking interval-valued discrete time markov chains. *Inf. Process. Lett.*, 113(7):210–216, 2013.

[11] Anne Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992.

[12] Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, July 1995.

[13] Maxence Dutreix and Samuel Coogan. Satisfiability bounds for co-regular properties in interval-valued markov chains. In *CDC*, pages 1047–1052, 2018.

[14] Javier Esparza, Jan Kretínský, and Salomon Sickert. One theorem to rule them all: A unified translation of LTL into ω-automata. In *LICS*, pages 384–393. ACM, 2018.

[15] Fernando L. Fussuma, Karina Valdivia Delgado, and Leliane Nunes de Barros. B^{2}rtdp: An efficient solution for bounded-parameter markov decision process. In *BRACIS*, pages 128–133, 2014.

[16] Robert Givan, Sonia M. Leach, and Thomas L. Dean. Bounded-parameter markov decision processes. *Artif. Intell.*, 122(1-2):71–109, 2000.

[17] Serge Haddad and Benjamin Monmege. Interval iteration algorithm for MDPs and IMDPs. *Theor. Comput. Sci.*, 735:111–131, 2018.

[18] Ernst Moritz Hahn, Vahid Hashemi, Holger Hermanns, Morteza Lahijanian, and Andrea Turrini. Multi-objective robust strategy synthesis for interval markov decision processes. In *QEST*, pages 207–223, 2017.

[19] Vahid Hashemi, Holger Hermanns, and Andrea Turrini. Compositional reasoning for interval markov decision processes. *CoRR*, abs/1607.08484, 2016.

[20] Cyrille Jégourel, Jingyi Wang, and Jun Sun. Importance sampling of interval markov chains. In *DSN*, pages 303–313, 2018.

[21] Bengt Jonsson and Kim Guldstrand Larsen. Specification and refinement of probabilistic processes. In *LICS*, pages 266–277, 1991.

[22] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Trans. Automat. Contr.*, 53(1):287–297, Feb 2008.

[23] Igor Kozine and Lev V. Utkin. Interval-valued finite markov chains. *Reliable Computing*, 8(2):97–113, 2002.

[24] Jan Kretínský, Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler. Rabinizer 4: From LTL to your favourite deterministic automaton. In *CAV (1)*, volume 10981 of *LNCS*, pages 567–577. Springer, 2018.

[25] Morteza Lahijanian, Sean B. Andersson, and Calin Belta. Formal verification and synthesis for discrete-time stochastic systems. *IEEE Trans. Automat. Contr.*, 60(8):2031–2045, 2015.

[26] Arnab Nilim and Laurent El Ghaoui. Robust control of markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

[27] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977.

[28] Alberto Puggelli, Wenchao Li, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Polynomial-time verification of PCTL properties of MDPs with convex uncertainties. In *CAV*, pages 527–542, 2013.

[29] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[30] Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M. Murray, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Model predictive control with signal temporal logic specifications. In *CDC*, pages 81–87. IEEE, 2014.

[31] Willy Arthur Silva Reis, Leliane Nunes de Barros, and Karina Valdivia Delgado. Robust topological policy iteration for infinite horizon bounded markov decision processes. *Int. J. Approx. Reasoning*, 105:287–304, 2019.

[32] Jay K. Satia and Roy E. Lave Jr. Markovian decision processes with uncertain transition probabilities. *Operations Research*, 21(3):728–740, 1973.

[33] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Model-checking markov chains in the presence of uncertainties. In *TACAS*, pages 394–410, 2006.

[34] Ambuj Tewari and Peter L. Bartlett. Bounded parameter markov decision processes with average reward criterion. In *COLT*, pages 263–277, 2007.

[35] J. G. THISTLE and W. M. WONHAM. Control problems in a temporal logic framework. *International Journal of Control*, 44(4):943–976, 1986.

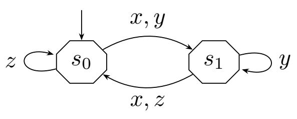


Fig. 5. Example DRA with acceptance $Acc = \{(s_0\, s_1), (s_1\, s_0)\}$.

[36] Wolfgang Thomas. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words.*, pages 389–455. Springer, 1997.

[37] Eric M. Wolff, Ufuk Topcu, and Richard M. Murray. Robust control of uncertain markov decision processes with temporal logic specifications. In *CDC*, pages 3372–3379, 2012.

[38] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon control for temporal logic specifications. In *HSCC*, pages 101–110. ACM, 2010.

[39] Di Wu and Xenofon D. Koutsoukos. Reachability analysis of uncertain systems using bounded-parameter markov decision processes. *Artif. Intell.*, 172(8-9):945–954, 2008.

[40] B. Yordanov, J. Tumova, I. Cerna, J. Barnat, and C. Belta. Temporal logic control of discrete-time piecewise affine systems. *IEEE Transactions on Automatic Control*, 57(6):1491–1504, June 2012.

VII. Appendix – The product construction

In this section, we briefly explain how linear objectives usually are specified and how they are model checked using the product construction. First, we define the concept of *ω-regular languages* and *automata*. Fix a finite *alphabet* Σ. Elements of Σ^ω are called *words* and sets of words $\mathcal{L} \subseteq \Sigma^\omega$ are called *languages*. Such a language is ω-regular if it can be *recognized* by an automaton.

A. Automata & regular languages

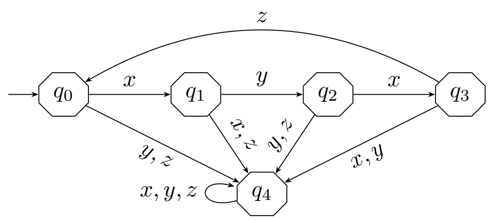
Definition 4 *A* deterministic Rabin automaton (DRA) *is a tuple $\mathcal{A} = (Q, T, q_0, Acc)$, where Q is a finite set of* states*, $T : Q \times \Sigma \to Q$ is a* transition function[3]*, $q_0 \in Q$ is an* initial state*, and $Acc \subseteq 2^Q \times 2^Q$ is the* Rabin condition*. An element $(F_i, I_i) \in Acc$ is called* Rabin pair*. We assume w.l.o.g. that $F_i \cap I_i = \emptyset$.*

Let $\mathcal{A}$ be a DRA and $w \in \Sigma^\omega$ a word. A word w induces a *run*, i.e. a sequence of states $\mathcal{A}(w) = q_0 q_1 q_2 \cdots \in Q^\omega$, where $q_{i+1} = T(q_i, w_i)$. As with MDP, let $\mathrm{Inf}(w)$ denote the set of states occurring *infinitely often* on the run $\mathcal{A}(w)$. A word is accepted by the automaton, denoted $w \models \mathcal{A}$, if there exists a Rabin pair $(F_i, I_i) \in Acc$ with $F_i \cap \mathrm{Inf}(w) = \emptyset$ and $I_i \cap \mathrm{Inf}(w) \neq \emptyset$. Such a Rabin pair is called *accepting for w*.

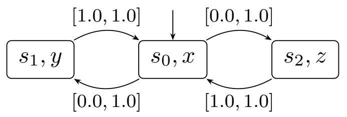
A language $\mathcal{L} \subseteq \Sigma^\omega$ is called *ω-regular* if and only if there exists a DRA $\mathcal{A}$ recognizing $\mathcal{L}$, i.e. some word $w \in \Sigma^\omega$ is accepted by the automaton if and only if it is in $\mathcal{L}$. See Fig. 5 for an example of a DRA recognizing the language "eventually only y or eventually only z".

Remark 6 *A wide variety of specifications are ω-regular. For example, reachability and liveness constraints can easily be translated to an automaton. Moreover, the whole of linear temporal logic is expressible through Rabin automata and efficient translations from LTL to Rabin automata exist [14].*

[3]Recall that the alphabet Σ is already fixed.



(a) Example DRA with acceptance $Acc = \{(q_2\, q_0, q_1)\}$



(b) Example IMC.

Fig. 6. Uncertain/adversarial interval interpretations are not equal.

B. Labelled MDPs & product

For the product construction, we modify the definition of MDPs by replacing the acceptance by a labelling function $\lambda : S \to \Sigma$, assigning to each state of the MDP a letter. We are given such a labelled MDP and a Rabin automaton. We construct the product by tracking both the evolution of the MDP and the automaton, where the automaton progresses based on the letter assigned to the current state.

Definition 5 *Let $\mathcal{M} = (S, s_0, A, \mathsf{Av}, \Delta, \lambda)$ be a labelled MDP and $\mathcal{A} = (Q, T, q_0, Acc)$ a Rabin automaton. The* product *$\mathcal{M} \otimes \mathcal{A} = (S \times Q, (s_0, q_0), A, \mathsf{Av}', \Delta', Acc')$ is an MDP where $\mathsf{Av}'((s, q)) := \mathsf{Av}(s)$, $\Delta((s, q), a, (s', q')) := \Delta(s, a, s')$ if $q' = T(q, \lambda(s))$ and 0 otherwise, and $Acc' = \{(F_i \times S, I_i \times S) \mid (F_i, I_i) \in Acc\}$.*

We analogously define this product construction for BMDP. Observe that the product is now of the form as we defined it in Definition 5. Applying our methods to this product yields a solution for the original system.

C. Caveat

Since this construction modifies the state space, it is not obvious how optimal policies on the product relate to policies on the original MDP. Indeed, while a memoryless policy may be optimal on the product, it might be the case that finite memory is needed to behave optimally in the given system. Moreover, it is the case that for ω-regular objectives, the optimal values for the uncertainty and the antagonistic interpretation are not equal already for IMCs.

Fix the alphabet $\Sigma = \{x, y, z\}$ and consider the language $\mathcal{L} = \{(xyxz)^\omega\}$, i.e. a language containing a single word w which repeats the string $xyxz$ indefinitely. This language is regular and is recognized by the automaton shown in Fig. 6a. Consider now the IMC in Fig. 6b. Clearly, the probability of satisfying the property is zero under the uncertainty interpretation – any MC consistent with the IMC eventually violates the structure of the property with probability 1. On the other hand, interpreted as an IMDP, the transitions can be chosen such that the required word is always produced.

F Approximating Values of Generalized-Reachability Stochastic Games (LICS 2020)

This is an exact reprinting of the paper which has been published as a **peer reviewed conference paper**.

> Pranav Ashok, Krishnendu Chatterjee, Jan Kretinsky, Maximilian Weininger, and Tobias Winkler. "Approximating Values of Generalized-Reachability Stochastic Games". In: LICS. ACM, 2020, pp. 102–115. doi: 10.1145/3373718.3394761

Summary

Simple stochastic games are turn-based $2\frac{1}{2}$-player games with a reachability objective. The basic question asks whether one player can ensure reaching a given target with at least a given probability. A natural extension is games with a conjunction of such conditions as objective. Despite a plethora of recent results on the analysis of systems with multiple objectives, the decidability of this basic problem remains open. In this paper, we present an algorithm approximating the Pareto-frontier of the achievable values to a given precision. Moreover, it is an anytime algorithm, meaning it can be stopped at any time returning the current approximation and its error bound.

For more details on this publication, we refer to Chapter 5 of the main body.

Contribution

Contribution of Maximilian Weininger	
Development and conceptual design of the research project	40%
Discussion and development of ideas	45%
Composition and revision of the manuscript	70%
Proofs	70%

Approximating Values of Generalized-Reachability Stochastic Games

Pranav Ashok
Technical University of Munich
Germany
ashok@in.tum.de

Krishnendu Chatterjee
IST Austria
Austria
Krishnendu.Chatterjee@ist.ac.at

Jan Křetínský
Technical University of Munich
Germany
jan.kretinsky@in.tum.de

Maximilian Weininger
Technical University of Munich
Germany
maxi.weininger@tum.de

Tobias Winkler
RWTH Aachen University
Germany
tobias.winkler@cs.rwth-aachen.de

Abstract

Simple stochastic games are turn-based 2½-player games with a reachability objective. The basic question asks whether one player can ensure reaching a given target with at least a given probability. A natural extension is games with a conjunction of such conditions as objective. Despite a plethora of recent results on the analysis of systems with multiple objectives, the decidability of this basic problem remains open. In this paper, we present an algorithm approximating the Pareto frontier of the achievable values to a given precision. Moreover, it is an anytime algorithm, meaning it can be stopped at any time returning the current approximation and its error bound.

CCS Concepts: • **Theory of computation** → *Algorithmic game theory*; **Verification by model checking**; • **Mathematics of computing** → *Probability and statistics.*

Keywords: Stochastic games; Multiple Reachability Objectives; Pareto frontier; Anytime algorithm

ACM Reference Format:
Pranav Ashok, Krishnendu Chatterjee, Jan Křetínský, Maximilian Weininger, and Tobias Winkler. 2020. Approximating Values of Generalized-Reachability Stochastic Games. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '20), July 8–11, 2020, Saarbrücken, Germany*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3373718.3394761

1 Introduction

Simple stochastic games [27] are zero-sum turn-based stochastic games (SG) with two players, which we call Maximizer and Minimizer. The objective of player Maximizer is to maximize the probability of reaching a given target set of states, while player Minimizer aims at the opposite. The basic decision problem is to determine whether there is a strategy for Maximizer achieving at least a given probability threshold. These games are interesting theoretically: the problem is known to be in NP ∩ co-NP, but whether it belongs to P is a major and long-standing problem. Moreover, several other important game problems such as parity games reduce to it [17]. Besides, they are also practically relevant: they can serve as a tool for synthesis with safety/co-safety objectives in environments with stochastic uncertainty.

Multi-objective stochastic systems have attracted a lot of attention recently, both SG and the special case with only one player (Markov decision processes, MDP [43]). They model and enable optimization with respect to conflicting goals, where a desired trade-off is to be considered. A natural multi-dimensional generalization of the reachability threshold constraint $\mathbb{P}[\Diamond T] \geq t$ is a conjunction $\bigwedge_i \mathbb{P}[\Diamond T_i] \geq t_i$ giving rise to *generalized-reachability* (or *multiple-reachability*) *stochastic games*, similar to e.g. generalized mean-payoff SG [8, 16]. The problem is then to decide whether a given vector of thresholds can be achieved by Maximizer. Note that these games are not determined [23], and in this paper we consider the lower-value (worst-case) problem formulation, i.e. finding a strategy of Maximizer that can guarantee the vector no matter what Minimizer does.

The main results established in the literature are as follows. For MDP, while generalized mean-payoff can be solved in P [10, 15], generalized reachability is PSPACE-hard and can be solved in exponential time [44]. For SG, generalized mean-payoff has been solved for almost-sure conditions only [8, 16] and approximation of the values for generalized mean-payoff as well as generalized reachability are still open. The generalized-reachability SG problem is only known to be decidable for the subclass of *stopping* SGs with a *2-dimensional* objective [13] (an SG is stopping if under any strategies a designated set of sinks is reached almost surely).

LICS '20, July 8–11, 2020, Saarbrücken, Germany
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7104-9/20/07.
https://doi.org/10.1145/3373718.3394761

The main open question for generalized-reachability SG is decidability. There are several important subgoals towards this problem: From the decidability perspective, stopping SG with more than 2-dimensional objectives, or general SG with 2- (or more) dimensional objectives have been open. Moreover, the same holds even for ε-approximability. From the algorithmic perspective, [23] provides a converging sequence of *lower bounds* on the *Pareto frontier*, i.e. the set of achievable vectors that are pointwise-maximal (in other words, vectors which cannot be improved in one dimension without sacrificing another one). It is open whether converging upper bounds can also be computed. Since such bounds would imply ε-approximability, this open question is the most imminent.

Our contribution in this paper is twofold. Firstly, we prove the following theorem:

> **Theorem:** *The set of all achievable vectors in an* arbitrary *(not necessarily stopping) SG with generalized-reachability objective of* any dimension *can be effectively approximated for any given precision ε.*

Secondly, we provide a value-iteration algorithm that approximates the Pareto frontier by giving converging lower *and upper* bounds. Consequently, it becomes an *anytime algorithm*, providing the current approximation and its error at each moment of the computation. Thus our first contribution resolves the ε-approximability open question and our second contribution resolves the algorithmic open question; both results are for arbitrary SG with generalized-reachability objectives of any dimension.

Convergent upper bounds on the value are known to be notoriously difficult to achieve. Until recently, the default engine for analysis in the most used probabilistic model checker PRISM [40] and PRISM-Games [22] used *value iteration*, e.g. [43], which converges to the value from below, but because of the used stopping criteria the results could be arbitrarily wrong [35]. For a solution with a given precision, one could use linear programming instead, which however, does not scale well for MDP, and, more importantly, does not work at all for SG [28]. For MDP, value iteration has been extended [11, 35] so that it provides not only the under-approximating convergent sequence, but also an over-approximating one, calling the technique "bounded value iteration" (due to [41]) or "interval iteration", respectively. Its essence is to collapse *maximal end components* (MECs) of the MDP, thereby not changing the values; on MDP without MECs the over-approximating sequence converges to the actual value of the (collapsed as well as original) MDP. This technique was further extended to MDP with mean-payoff objective [2]. In contrast, for SG one cannot collapse MECs since they account for non-trivial alternating structure, as opposed to MDP, where any desired action exiting the MEC can be taken almost surely. Therefore, a more complex procedure has been proposed for SG [37]: Depending on the current under-approximation, problematic parts of MECs are dynamically identified and their over-approximation is lowered to over-approximations of certain actions exiting the MEC, as exemplified and explained later. We lift this procedure to general dimensions. Note that we do not give any convergence rate for our algorithm, because it is not possible to extend the argument of the single-dimensional case [19] in a straightforward manner. This argument requires the lowest probability occurring in a play to be bounded. However, in the multi-dimensional setting strategies may need infinite memory and hence there is no lower bound on the probability that a strategy assigns to actions [24, Appendix B1, full version]. Giving bounds on the convergence is an interesting direction of future work.

This paper combines and extends several techniques from literature to obtain the corresponding result for the multi-dimensional case:

- Firstly, we use the Bellman operator extended to downward-closed sets (instead of just real values) [23], allowing for value iteration in the multi-dimensional setting.
- Secondly, we exploit the technique of [37], which in the single-dimensional setting repetitively identifies the problematic parts of MECs hindering convergence.
- Thirdly, in order to apply this technique, we reduce the multi-dimensional problem to a continuum of single-dimensional problems, by splitting the Pareto front into directions, similar to [33].
- Fourthly, we group the single-dimensional problems into finitely many *regions*, similar in spirit to regions of timed automata [1] since they are essentially given by orderings of the approximate values of certain actions. Nevertheless, due to the projective geometry of the problem, we need to work slightly more generally with simplicial complexes, see e.g. [36].

The main technical difficulty is to identify (i) the parts of MECs with an unjustified too high upper bound and (ii) the value to which it should be decreased in each step. Both of these depend on the desired trade-off between the targets. As we compute the whole set of achievable vectors, we need to consider all possible trade-offs, which are, moreover, uncountably many.

Related work. Already for a decade, MDP have been extensively studied in the setting of multiple objectives. Multiple objectives have been considered both qualitative, such as reachability and LTL [30], as well as quantitative, such as mean payoff [10, 15], discounted sum [18], or total reward [32]. The expectation has been combined with variance in [12]. Beside expectation queries, conjunctions of percentile (threshold) queries have been considered for various objectives [10, 21, 31, 45]. Further, for general Boolean combinations for Markov chains with total reward, [34] approximates the value, while computability is still open. In contrast, [47] shows that Boolean combinations over mean payoff games become quickly undecidable. For the specifics of the two-dimensional case and the interplay of the two objectives, see [4]. The usage of the multi-dimensional setting is discussed in [5, 6], comparing multiple rewards and quantiles and reporting how they have practically been applied and found useful by domain experts.

More recently, SG have been also analyzed with multiple objectives; [46] provides an overview and implementation of existing algorithms for Pareto frontier computation for multi-objective total reward, reachability, and probabilistic LTL properties as well as mixtures thereof. However, the computation is limited to stopping SGs, i.e. ones without end components

Multiple mean-payoff objective was first examined in [8] and both the qualitative and the quantitative problems are coNP-complete [16]. Although Boolean combinations of mean-payoff are undecidable in general [47], in certain subclasses of SG they can be approximated [9]. Boolean combinations of total-reward objectives were approximated in the case of stopping games [23] and applied to autonomous driving [25], where LTL is reduced to total reward in the case of stopping games and, for dimension two, the problem is shown decidable in [13].

PRISM-Games [38] provides tool support for several multi-player multi-objective settings [39]. Other tools supporting multi-player settings, GAVS+ [26] and GIST [20], are not maintained any more and are limited to single-objective settings.

In many settings, Pareto frontiers can be ε-approximated in polynomial time [42]. Pareto frontiers are constructed for the generalized mean-payoff objective for 2-player (non-stochastic) games in [14], MDPs in [10, 21], and SGs in [9]. For the generalized-reachability, the Pareto frontier is approximated for MDP in [30], but for SG the Pareto frontier is not even known to be given by finitely many points, except for dimension two [13]. In contrast, in the single-dimensional case, the value is known to be a multiple of a denominator that can be calculated from the syntactic description of the game [19].

Structure of the paper After recalling the basic notions in Section 2, we illustrate the problem, the difficulties and our solution on examples in Section 3. The algorithm is described and the correctness intuitively explained in Section 4 and formally proven in Section 5. The proofs of several technical statements are, for the sake of readability, relayed to [3, Appendix]. We conclude in Section 6.

2 Preliminaries

2.1 Stochastic Games

A probability distribution on a finite set X is a mapping $\delta : X \to [0,1]$, such that $\sum_{x\in X}\delta(x) = 1$. The set of all probability distributions on X is denoted by $\mathcal{D}(X)$. Given a dimension $n \in \mathbb{N}$, often implicitly clear from context, and $c \in \mathbb{R}$, we let $\vec{c}$ denote the n-dimensional vector with all components equal to c. For a vector $\vec{v}$, its i-th component is denoted $\vec{v}_i$. We compare vectors component-wise, i.e. $\vec{u} \le \vec{v}$ if $\vec{u}_i \le \vec{v}_i$ for all i. In this paper, we restrict ourselves to non-negative vectors, i.e. elements of $\mathbb{R}^n_{\ge 0}$

Now we define turn-based two-player stochastic games. As opposed to the notation of e.g. [27], we do not have special stochastic nodes, but rather a probabilistic transition function.

Definition 2.1 (SG)**.** A *stochastic game (SG)* is a tuple $(S, S_\square, S_\bigcirc, \mathsf{s}_0, \mathsf{A}, \mathsf{Av}, \delta)$, where S is a finite set of *states* partitioned into the sets $S_\square$ and $S_\bigcirc$ of states of the player *Maximizer* and *Minimizer*, respectively, $\mathsf{s}_0 \in S$ is the *initial* state, A is a finite set of *actions*, $\mathsf{Av} : S \to 2^{\mathsf{A}}$ assigns to every state a set of *available* actions, and $\delta : S \times \mathsf{A} \to \mathcal{D}(S)$ is a *transition function* that given a state s and an action $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$ yields a probability distribution over *successor* states.

A Markov decision process (MDP) is then a special case of SG where $S_\bigcirc = \emptyset$. We assume that SG are non-blocking, so for all states s we have $\mathsf{Av}(\mathsf{s}) \ne \emptyset$.

For a state s and an available action $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$, we denote the set of successors by $\mathsf{Post}(\mathsf{s},\mathsf{a}) := \{\mathsf{s}' \mid \delta(\mathsf{s},\mathsf{a},\mathsf{s}') > 0\}$. We say a state-action pair (s,a) is an *exit* of a set of states T, written (s,a) exits T, if $\exists t \in \mathsf{Post}(\mathsf{s},\mathsf{a}) : t \notin T$, i.e., if with some probability a successor outside of T could be chosen. Further, we use $\mathsf{Exits}(T) = \{(\mathsf{s},\mathsf{a}) \mid \mathsf{s} \in T, \mathsf{a} \in \mathsf{Av}(\mathsf{s}), (\mathsf{s},\mathsf{a}) \text{ exits } T\}$ to denote all exits of a state set $T \subseteq S$. Finally, for any set of states $T \subseteq S$, we use $T_\square$ and $T_\bigcirc$ to denote the states of T that belong to Maximizer and Minimizer, whose states are drawn in the figures as $\square$ and $\bigcirc$, respectively.

The semantics of SG is given in the usual way by means of strategies and the induced Markov chain [7] and its respective probability space, as follows. An *infinite path* ρ is an infinite sequence $\rho = \mathsf{s}_0\mathsf{a}_0\mathsf{s}_1\mathsf{a}_1\cdots \in (S \times \mathsf{A})^\omega$, such that for every $i \in \mathbb{N}$, $\mathsf{a}_i \in \mathsf{Av}(\mathsf{s}_i)$ and $\mathsf{s}_{i+1} \in \mathsf{Post}(\mathsf{s}_i,\mathsf{a}_i)$. *Finite path*s are defined analogously as elements of $(S \times \mathsf{A})^* \times S$. A *strategy* of Maximizer or Minimizer is a function $\sigma : (S \times \mathsf{A})^* \times S_\square \to \mathcal{D}(\mathsf{A})$ or $(S \times \mathsf{A})^* \times S_\bigcirc \to \mathcal{D}(\mathsf{A})$, respectively, such that $\sigma(\rho\mathsf{s}) \in \mathcal{D}(\mathsf{Av}(\mathsf{s}))$ for all s. We call a strategy *deterministic* if it maps to Dirac distributions only; otherwise, it is *randomizing*. A pair (σ,τ) of strategies of Maximizer and Minimizer induces an (infinite state) Markov chain $\mathsf{G}^{\sigma,\tau}$ with finite paths as states, s_0 being initial, and the transition function $\delta(\mathsf{w}\mathsf{s}, \mathsf{w}\mathsf{s}\mathsf{a}\mathsf{s}') = \sigma(\mathsf{w}\mathsf{s})(\mathsf{a}) \cdot \delta(\mathsf{s},\mathsf{a},\mathsf{s}')$ for states of Maximizer and analogously for states of Minimizer, with σ replaced by τ. The Markov chain induces a unique probability distribution $\mathbb{P}^{\sigma,\tau}$ over measurable sets of infinite paths [7, Ch. 10] (the usual index with the initial state is not used since it is fixed already in the game).

2.2 End Components

Now we recall a fundamental tool for analysis of MDP called end components. An end component of a SG is then defined as the end component of the underlying MDP with both players unified.

Definition 2.2 (EC)**.** A non-empty set $T \subseteq S$ of states is an *end component (EC)* if there is a non-empty set $B \subseteq \bigcup_{s\in T} \mathsf{Av}(s)$ of actions such that

1. for each $\mathsf{s} \in T$, $\mathsf{a} \in B \cap \mathsf{Av}(\mathsf{s})$, we have $(\mathsf{s},\mathsf{a}) \notin \mathsf{Exits}(T)$,

2. for each $\mathsf{s}, \mathsf{s}' \in T$ there is a finite path $\mathsf{w} = \mathsf{s}\mathsf{a}_0 \dots \mathsf{a}_n \mathsf{s}' \in (T \times B)^* \times T$, i.e. the path stays inside T and only uses actions in B.

Intuitively, ECs correspond to bottom strongly connected components of the Markov chains induced by possible strategies. Hence for some pair of strategies all possible paths starting in an EC remain there. An EC T is a *maximal end component (MEC)* if there is no other end component T' such that $T \subsetneq T'$. Given an SG G, the set of its MECs is denoted by $\mathsf{MEC}(\mathsf{G})$ and can be computed in polynomial time [29].

2.3 Generalized Reachability

For a set $T \subseteq S$, we write $\Diamond T := \{\mathsf{s}_0\mathsf{a}_0\mathsf{s}_1\mathsf{a}_1 \dots \in (S \times \mathsf{A})^\omega \mid \exists i \in \mathbb{N} : \mathsf{s}_i \in T\}$ to denote the (measurable) set of all paths which eventually reach T. A *generalized-reachability objective* (of dimension n) is an n-tuple $\mathcal{T} = (T_1, \dots, T_n)$ of state sets $T_i \subseteq S$. A vector $\vec{v}$ (of dimension n) is *achievable* if there is a strategy σ of Maximizer such that for all strategies τ of Minimizer

$$\forall i \in \{1, \dots, n\} \quad \mathbb{P}^{\sigma,\tau}(\Diamond T_i) \geq \vec{v}_i$$

Note that, since these games are not determined [23], this corresponds to the lower value, i.e. the worst case analysis.

For a given state s, the set of points achievable *from* s, meaning in a game where the initial state is set to s, is denoted $\mathfrak{A}_{\mathcal{T}}(s)$ or just $\mathfrak{A}(s)$ when $\mathcal{T}$ is clear from context.

2.4 Basic Geometry Notation and Pareto Frontiers

In order to consider convex combinations of sets, we define scaling of a set $X \subseteq \mathbb{R}^n$ by a constant $c \in [0, 1]$ as $c \cdot X = \{c \cdot x \mid x \in X\}$, and the Minkowski sum of sets X and Y as $X + Y = \{x + y \mid x \in X, y \in Y\}$. The convex hull of a set X is denoted by $conv(X) = \{\sum_{i=1}^k a_i x_i \mid k \geq 1, x_i \in X, a_i \geq 0, \sum_{i=1}^k a_i = 1\}$.

A *downward closure* of a set X of vectors is $dwc(X) := \{y \mid \exists x \in X : y \leq x\}$. A set X is *downward closed* if $X = dwc(X)$. The set $\mathfrak{A}$ of achievable points is clearly downward closed.

It will be convenient to use a few basic notions of projective geometry, which we now recall. Intuitively, a *direction* is a ray from the origin $\vec{0}$ into the (n-dimensional) first quadrant. As such, we may represent it with any vector $\vec{v} \neq \vec{0}$ on that ray. Then all vectors $\lambda \cdot \vec{v}$ for any $\lambda \in \mathbb{R}_{>0}$ are equivalent and represent the same direction. For instance, direction $\mathbf{d} = [(1, 0, 0)]$ denotes the x-axis and it is equal to $[(\lambda, 0, 0)]$ for any $\lambda > 0$. Formally, a direction $\mathbf{d} = [\vec{v}]$ is the set $\{\lambda \cdot \vec{v} \mid \lambda \in \mathbb{R}_{>0}\}$. We denote by $\mathrm{D} = \{[\vec{v}] \mid \vec{v} \in dwc(\{\vec{1}\})\}$ the set of all directions (in the first quadrant).

Given a set X of points and a direction $\mathbf{d}$, X *evaluated in direction* $\mathbf{d}$ is the (Euclidean) length of the vector from the origin to the farthermost intersection of X and $\mathbf{d}$, denoted

$$X[\mathbf{d}] := \sup\{||\vec{x}|| \mid \vec{x} \in X, \mathbf{d} = [\vec{x}]\}$$

with the usual $\sup \emptyset = 0$. Fig. 1 illustrates an evaluation of a direction on an achievable set. Intuitively, it describes what is

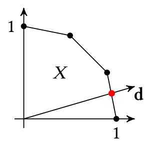


Figure 1. Example showing a Pareto frontier of a set X, a direction $\mathbf{d}$, and the point of intersection of $\mathbf{d}$ with the frontier, depicted as ● in distance $X[\mathbf{d}]$ from the origin.

achievable if we prefer the dimensions in the "ratio" given by $\mathbf{d}$. Another example is the whole set (blue and red) of Fig. 4a: evaluated in $[(1, 1)]$ it yields $\sqrt{2}/2$.

Given a downward closed set X, its *Pareto frontier* is the set of farthermost points in each direction:

$$\mathfrak{P}(X) = \{\vec{x} \mid \mathbf{d} \in \mathrm{D}, \mathbf{d} = [\vec{x}], X[\mathbf{d}] = ||\vec{x}||\}$$

The *Pareto frontier of a state* s is the Pareto frontier of the set achievable in s, i.e. $\mathfrak{P}(\mathsf{s}) := \mathfrak{P}(\mathfrak{A}(\mathsf{s}))$. The *Pareto set of the game* is $\mathfrak{P} := \mathfrak{P}(\mathsf{s}_0)$. Thus by definition, $\mathfrak{P} = \mathfrak{P}(\mathfrak{A}(\mathsf{s}_0))$ and, further, $dwc(\mathfrak{P})$ is (the closure of) $\mathfrak{A}(\mathsf{s}_0)$.[1] Note that it is not known whether $\mathfrak{A}$ is closed, since it is not known whether the suprema of achievable points are also achievable. Our notion of $\mathfrak{P}$ includes these suprema, which is why it is only equal to the closure of $\mathfrak{A}$.

2.5 Problem Formulation

In this paper, we are interested in ε-approximating $\mathfrak{P}$. In terms of under- and over-approximation:

Given an SG, generalized-reachability objective $\mathcal{T}$, and precision $\varepsilon > 0$, the task is to construct sets $\mathcal{L}, \mathcal{U} \subseteq \mathbb{R}^{|\mathcal{T}|}$ such that for each direction $\mathbf{d} \in \mathrm{D}$, $\mathcal{L}[\mathbf{d}]$ and $\mathcal{U}[\mathbf{d}]$ are effectively computable and we have

$$\mathcal{L}[\mathbf{d}] \leq \mathfrak{P}[\mathbf{d}] \leq \mathcal{U}[\mathbf{d}] \quad \text{and} \quad \mathcal{U}[\mathbf{d}] - \mathcal{L}[\mathbf{d}] < \varepsilon\,.$$

2.6 Multi-dimensional and Bounded Value Iteration

In this section we recall two extensions of the standard value iteration: a generalization for *multi-dimensional* objectives and a *"bounded"* one with an over-approximating sequence. Firstly, the multi-dimensional Bellman operator for reachability, e.g. [23],

$$\mathfrak{B} : \left(S \to 2^{[0,1]^n}\right) \to \left(S \to 2^{[0,1]^n}\right)$$

works with *sets* $X(\mathsf{s})$ of points achievable in s rather than single points:

[1] Our notion of Pareto frontier captures the whole surface in the first quadrant. Other definitions such as $\mathfrak{P}_{\mathcal{T}} = \{\vec{v} \mid \vec{v} \text{ is achievable } \land \forall \text{ achievable } \vec{u} : \vec{u} \not> \vec{v}\}$ only capture the Pareto optimal points. For example, if the set of achievable points in the three-dimensional space is the whole unit cube then our definition returns its three sides, while the other definition returns only the singleton with the Pareto optimal point $(1, 1, 1)$.

$$\mathfrak{B}(X)(\mathsf{s}) = \begin{cases} \bigcap_{a\in \mathsf{Av}(\mathsf{s})} X(\mathsf{s},\mathsf{a}) & \text{if } \mathsf{s} \in S_{\bigcirc} \\ conv(\bigcup_{a\in \mathsf{Av}(\mathsf{s})} X(\mathsf{s},\mathsf{a})) & \text{if } \mathsf{s} \in S_{\Box} \end{cases}$$

where we define

$$X(\mathsf{s},\mathsf{a}) = \left(dwc(\{\mathbb{1}_{\mathcal{T}}(\mathsf{s})\}) + \sum_{\mathsf{s}'\in S} \delta(\mathsf{s},\mathsf{a},\mathsf{s}') \cdot X(\mathsf{s}') \right) \cap \mathbf{1}$$

and $\mathbb{1}_{\mathcal{T}}$ is the indicator vector function of target sets, i.e. $\mathbb{1}_{\mathcal{T}}(\mathsf{s})_i$ equals 1 if $\mathsf{s} \in T_i$ and 0 otherwise, and $\mathbf{1} = dwc(\{\vec{1}\})$ is the unit box.

Intuitively, the operator works as follows. Given what can be achieved from s using now an action a, we can compute the value for the minimizing state as the intersection over all actions since these points are achievable no matter what Minimizer does. For maximizing states, if there exists an action achieving a point then Maximizer can achieve it from here; moreover, we compute the convex hull since Maximizer can also randomize and, as opposed to the minimizing case with intersection, union of convex sets need not be convex. Once we have dealt with decision making on the first line, it remains to determine what can be achieved by each decision, on the second line. The achievable values are given by the weighted average of the successors' values, but additionally, the base case of targets must be handled. Namely, whenever a state is in a target set, all values up to 1 in that dimension are achievable (but not greater than 1).

This also gives rise to an algorithm approximating $\mathfrak{A}$, which is the least fixpoint of $\mathfrak{B}$ [23]. We initialize $\mathsf{L} : S \to 2^{[0,1]^n}$ to return $\{\vec{0}\}$ everywhere and iteratively apply the Bellman operator, yielding arbitrarily precise approximations of $\mathfrak{A}$ by $\mathfrak{B}^k(\mathsf{L})$ as $k \to \infty$ [23][2]. Moreover, for every state s it can be checked that the set $\mathfrak{B}^k(\mathsf{L})(s)$ is presented at each step k as a closed downward-closed convex polyhedron, i.e. a *finite* object. Thus we can effectively construct any desired approximation.

However, it is not known how to bound the difference of the actual achievable set $\mathfrak{A}$ and the approximation after k iterations. For that reason, [37] introduced for the single-dimensional case the *bounded value iteration* (named along the tradition of [41]), a way to compute also an over-approximating sequence. If we initialize U to return $dwc(\{\vec{1}\})$ everywhere[3], then $\lim_{k\to\infty} \mathfrak{B}^k(\mathsf{U})$ is a fixpoint, which is generally different from the least one. Hence [37] modifies $\mathfrak{B}$ so that it has a single fixpoint equal to the least one of the original $\mathfrak{B}$. Then both the sequence of lower bounds and of upper bounds converge to $\mathfrak{A}$, the value of the game. The modification is demonstrated in the next section, where we also illustrate the main ideas how to cope with the multi-dimensional case.

[2]Precisely, $\lim_{k\to\infty} \mathfrak{B}^k(\mathsf{L}) \subseteq \mathfrak{A} \subseteq dwc(\mathfrak{P}) = \text{clos}(\lim_{k\to\infty} \mathfrak{B}^k(\mathsf{L}))$ where clos is the standard closure in $\mathbb{R}^n$.

[3]The same holds even if we initialize to 0 all the dimensions i in states from which there is no path to T_i, as is customary in MDP analysis. The solution of [37] is not sensitive to this and does not require this special treatment in the initialization of U.

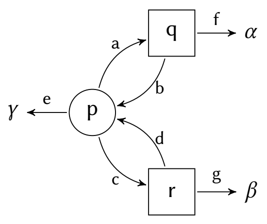


Figure 2. An example demonstrating the complications arising in an end component.

3 Example

In this section, we illustrate the issues preventing convergence of the upper bounds, as well as the solution of [37] and our extension of it. Value iteration converges if the SG is stopping, i.e. if the game reaches a designated sink with probability 1, or equivalently, if there are no end components (ECs). Hence the difficulty in solving reachability SG is rooted in ECs, as it is possible to cycle in its states infinitely long. As a running example, consider the EC in Fig. 2 with states p, q, r and actions a, . . . , g. The symbols α, β and γ are placeholders; in the single dimensional case, they represent a real number; in the multi dimensional case, a Pareto frontier. One can make this game a standard SG in the single dimensional case by, for example, replacing α with a transition that reaches the target with probability α and the sink with probability $1 - \alpha$. The multi-dimensional case is a straightforward extension.

We start by considering the single-reachability objective. The standard Bellman update procedure as described in Section 2.6 reduces to the following equations, where intersections become minima and unions become maxima. We write U_k as short for $\mathfrak{B}^k(\mathsf{U})$.

$$\begin{aligned} \mathsf{U}_{i+1}(\mathsf{p}) &= \min\{\mathsf{U}_i(\mathsf{q}), \mathsf{U}_i(\mathsf{r}), \gamma\} \\ \mathsf{U}_{i+1}(\mathsf{q}) &= \max\{\mathsf{U}_i(\mathsf{p}), \alpha\} \\ \mathsf{U}_{i+1}(\mathsf{r}) &= \max\{\mathsf{U}_i(\mathsf{p}), \beta\} \end{aligned}$$

By replacing U with L, we get the update equations for the lower bound. Recall that we initialize L_0 to return 0 everywhere and U_0 to return 1 everywhere.

3.1 MDP

Firstly, let us briefly mention the solution of [11, 35] for MDP. Suppose that all states belonged to the maximizing player, i.e. p was also maximizing. Then, the initialization $\mathsf{U}_0 = \vec{1}$ is already a fixpoint, although the true value of all three states is $\max\{\alpha, \beta, \gamma\}$. Intuitively, the reason for this is that the equations create a cyclic dependency: the process of finding the value by "asking neighbours" is not well-founded and all states falsely believe that they can achieve the higher value 1. [37] calls such an EC *bloated*, having an unjustifiably large (bloated) upper bound. The solution of [11, 35] is to detect that this is an EC and collapse it into a single state, eliminating the cycle. Only outgoing actions α, β, γ of the EC are kept, and in the next iteration, the Bellman operator correctly sets the value of the collapsed state to $\max\{\alpha, \beta, \gamma\}$, thus converging to the true value. The solution of [37] captures this idea from a different perspective: It does not change the underlying graph, but instead realizes that all three states can reach the *"best exit"* of the EC, i.e. the state with an action exiting the EC and having the highest value. Then the algorithm reduces the upper bounds of the states of the EC to that of the best exit. This is called *deflating*, as the "internal higher pressure" of bloated upper bounds is "relieved", equalizing with the best exit.

3.2 Single-reachability SG

Secondly, for single-reachability *SG*, the EC cannot in general be collapsed, since the values of the states differ, and it is not clear a priori which states share a value. They depend on the *ordering* of the values of the exits, i.e. on the ordering of α, β and γ.

Case 1: If $\gamma < \min(\alpha, \beta)$, then after the first iteration we have $\mathsf{U}_1(\mathsf{p}) = \gamma$, $\mathsf{U}_1(\mathsf{q}) = 1$ and $\mathsf{U}_1(\mathsf{r}) = 1$. After the next iteration, $\mathsf{U}_2(\mathsf{p}) = \gamma$, $\mathsf{U}_2(\mathsf{q}) = \alpha$ and $\mathsf{U}_2(\mathsf{r}) = \beta$. These are the true values, as observable in Figure 2. In this case U_k converges to the value. However, note that the values of the states in the same EC are different.

Case 2: If $\gamma \geq \min(\alpha, \beta)$, and say $\alpha > \beta$, then the values of p and r are β and that of q is α. This is the case, because p will always play action c, not allowing state r to achieve anything but the smallest value β. However, U_k does not converge to these values. In the first iteration, $\mathsf{U}_1(\mathsf{p}) = \gamma$, $\mathsf{U}_1(\mathsf{q}) = 1$ and $\mathsf{U}_1(\mathsf{r}) = 1$. After the next iteration, $\mathsf{U}_2(\mathsf{p}) = \mathsf{U}_2(\mathsf{q}) = \mathsf{U}_2(\mathsf{r}) = \gamma$. After this, the upper bounds do not change any more, because we have the problem of cyclic dependencies as described in Section 3.1. If we fix the strategy of the Minimizer to c as that is the best choice, only $\{\mathsf{p}, \mathsf{r}\}$ forms an EC. The value of both p and r is β, as that is the best exit that the Maximizer can achieve, given that Minimizer does not play the suboptimal action a. Such an EC where all states share the same value is called simple end component (SEC) [37]. It is simple, because after fixing the strategy of Minimizer to be optimal, this player cannot influence the play anymore (as the SG locally becomes an MDP). In the SEC, Maximizer can direct the play to the best exit and almost surely achieve the value of it. Deflating the SEC $\{\mathsf{p}, \mathsf{r}\}$, i.e. setting the upper bound for all states in the SEC to that of the best exit, correctly updates the bounds to β. Afterwards, the upper bound of q is correctly set to α in the next iteration. So one would like to find and deflate all SECs.

However, which states form a SEC depends on the relative ordering of the exits' values and the corresponding choices that Minimizer makes (recall we had to fix the strategy of p to the optimal action c in order to realize which states form the SEC). Indeed, in the case with $\alpha < \beta$, a different SEC ($\{\mathsf{p}, \mathsf{q}\}$) should be deflated and if $\alpha = \beta$ then all three states form a SEC. Since we do not know the values of the exits, the algorithm uses the approximations (L_i) to guess which actions are suboptimal for the Minimizer, and hence which states form a SEC. As the lower approximation converges to the value, the true SECs are eventually detected and correctly deflated. However, when L_i is not yet close enough to the value, the computation of SECs can be wrong, e.g. if $\alpha < \beta$, but for the first few iterations of the algorithm the lower bound on β is smaller than that on α. Then, for these first iterations, the algorithm believes $\{\mathsf{p}, \mathsf{r}\}$ to be the SEC, and only afterwards realizes that it actually is $\{\mathsf{p}, \mathsf{q}\}$. Hence, the operation we perform on the SEC has to be conservative, i.e. sound even if it is given a set of states that actually do not form a SEC. This is why deflating was introduced, as it is sound for any EC, even ones that are not SECs [37, Lemma 3]. In contrast, modifying the underlying graph by collapsing as in [11, 35] would commit to the detected SEC-candidate and thereby possibly make the wrong choice. Note that we never know that we have correctly detected a SEC, we just know that in the limit we will eventually detect it.

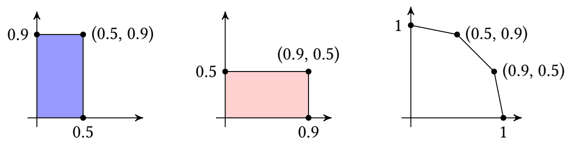


Figure 3. Pareto frontiers of α (left), β (center) and γ (right) in a 2-objective setting. X-axis represents the value along the first objective and Y-axis represents the value along the second objective.

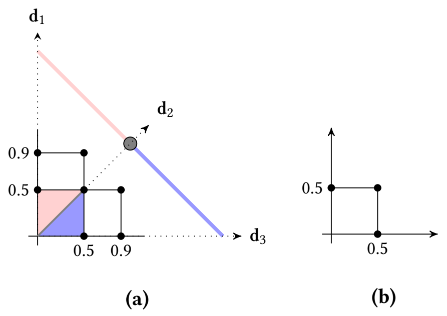


Figure 4. (a) Visualizing the regions for state p; and (b) the result of deflating.

3.3 Generalized-reachability SG

Here we intuitively describe and illustrate the main elements of our solution. The formal definitions of the key concepts only follow in the next section.

Regions. Consider again the example of Fig. 2. In the multi-dimensional case, instead of α, β and γ being reals, they are sets of achievable vectors. Let them be given as in Fig. 3, so e.g. $\alpha = dwc(\{(0.5, 0.9)\})$. Here γ gives the highest values, so it is the best one for Maximizer, and hence Minimizer will not play the corresponding e (as in Case 2 in Section 3.2). α and β, however, cannot be compared. Depending on the trade-off (corresponding to a direction) that Maximizer wants to achieve, α or β might be better than the other. To this end, let $\mathbf{d}$ be the direction in which Maximizer wants to maximize. Depending on $\mathbf{d}$, Minimizer's behaviour changes. If the objective along the x-axis is more important, then Minimizer chooses action a. This way, the value of the more important objective is restricted to 0.5. If on the other hand, the objective along y-axis is more important, then the Minimizer chooses action c. The Minimizer, for each direction $\mathbf{d}$, decides on the action to be chosen by comparing α and β evaluated in that direction; in other words, by computing the minimum of $\alpha[\mathbf{d}]$ and $\beta[\mathbf{d}]$.

Our algorithm identifies finitely many *regions* where the Minimizer has the same preference ordering over actions and then we deflate each region separately. In our example, we can identify three regions, as shown in Fig. 4a. Between the directions $\mathbf{d}_1$ and $\mathbf{d}_2$ (red ● region), Minimizer's best choice is action c; between $\mathbf{d}_2$ and $\mathbf{d}_3$ (blue ● region), Minimizer's best choice is action a; and along $\mathbf{d}_2$ (grey ● line), Minimizer is indifferent.

Deflating regional SECs. Once restricting to a region fixes the preference ordering over Minimizer's actions, we can proceed as in the single-dimensional case: We fix Minimizer's optimal strategy based on the lower bounds, identify SEC-candidates and deflate them. That means we update the Pareto frontier in the region to that of the *best exit* from the SEC. The whole Pareto frontier is constructed piece by piece, region by region.

Returning to our running example, we have already identified the three regions in the Pareto frontier for state p in Figure 4a. The SECs depending on the regions are as follows: In the blue ● region it is {p, q}, in the red ● region it is {p, r}, and along $\mathbf{d}_2$ all three states form a SEC. Deflating the blue ● region, we see that the best exit from the SEC has value α, so between 0° and 45° the value of p is set to the corresponding part of α. Doing the same for the other two regions results in the Pareto frontier depicted in Figure 4b. This result is also intuitively expected, as depending on which direction Maximizer prefers, Minimizer can always restrict the play to the other exit. Note that for the sake of example here we always talked about the true values, while the algorithm does not know these precisely. Therefore, deflating cannot update the values based on the value of α, but only on its approximation. Being on the safe side, the values will be decreased only to its over-approximation.

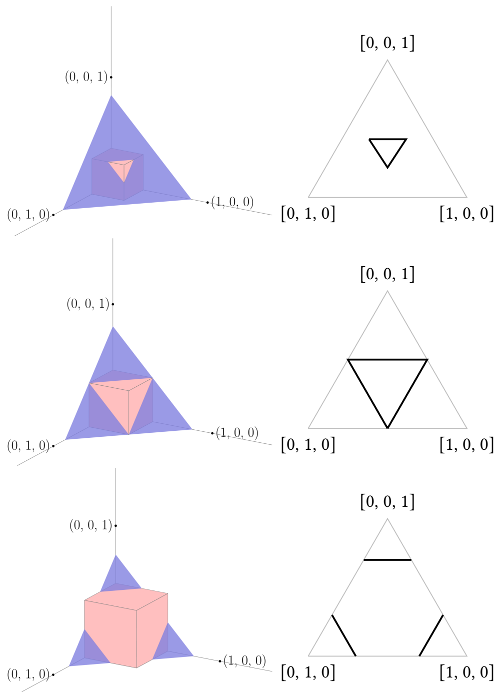


Figure 5. The left column shows Pareto frontiers; there is the blue tetrahedon and a pink box of varying size. The right column shows the projection of the intersection onto the projective hyperplane.

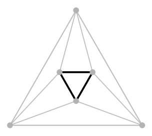
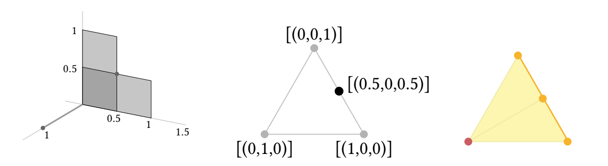


Figure 6. Triangulation of the top right of Fig. 5

Computing and representing regions. As explained above, a region depends on the preference ordering of actions. To compute regions where this ordering is constant, we use geometric methods. In the example of Fig. 4a, the point where the preference ordering changes is $(0.5, 0.5)$, which is where the two Pareto frontiers intersect. So, intuitively, by drawing the Pareto frontiers and finding the points of intersection, we can identify the regions (sets of the corresponding directions) where the preference ordering over actions is constant.

In Figure 5, we give a set of three examples to illustrate the construction of regions. The left picture in each row of the figure shows two Pareto frontiers: One is the blue tetrahedron, generated by Maximizer's free, but exclusive choice between target sets. The other is a red box of different sizes, generated by the possibility to reach a state in all target sets with a given probability. From top to bottom, we increase this



(a) Three sets of achievable points

(b) Projection of the intersections

(c) Convex regions as a simplicial complex

Figure 7. Projections of intersections of Pareto frontiers to the projection plane, which in 3D is the triangle formed by the points $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$. In Fig. 7b, labels represent directions and not individual vectors.

probability, thereby increasing the size of the box, yielding three different examples. We define regions as sets of directions. In order to draw directions, it is useful to consider the so-called *projective hyperplane.* It is the set of all directions and can be drawn (in our case with non-negative vectors only) as a triangle with corners $[1, 0, 0]$, $[0, 1, 0]$, $[0, 0, 1]$, capturing all directions. When a point (vector) $\vec{v}$ is projected into its direction $[\vec{v}]$, it intuitively corresponds to drawing a ray from the origin through the point $\vec{v}$. If we identify the projective hyperplane with the hyperplane passing through the *points* $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$ (or more precisely with this triangle) then the intersection of the ray and the hyperplane, say point $\vec{p}_v$, is the *projection* of $\vec{v}$ to the projective hyperplane. In our example, the right side of the figure shows the projection of the intersection of the Pareto frontiers onto the projective hyperplane. This gives rise to three regions, each with different preference ordering: the inner open triangle, its boundary and the outer triangle with the hole. Minimizer prefers the red action in the outside triangle, the blue one in the inside triangle, and is indifferent on the boundary. As these regions are hard to describe (as well as possibly not even convex and connected), we triangulate the projections to get smaller regions which are convex and generated by finitely many points. The triangulation of the top right of Figure 5 is depicted in Figure 6. Further note that while the preference ordering of actions is constant in each region, the faces of a region represent turning points of the preference ordering; hence these faces need to be separate regions like is customary for timed automata [1]. Hence in order to represent the regions, we thus decompose the triangle (generally, in higher dimensions, a *simplex*) into open triangles, open line segments and points (in general into a *simplicial complex*, i.e. the simplex together with its faces and recursively their faces).

As another example of the projection to the projective hyperplane and the triangulation, consider Figure 7a with three achievable sets: two rectangles – $dwc(\{(1, 0, 0.5)\})$ and $dwc(\{(0.5, 0, 1)\})$; and one line – $dwc(\{(0, 1, 0)\})$. The frontiers of the sets generate only one non-empty intersection[4], namely the point $(0.5, 0, 0.5)$. Its projection is represented by its direction, $[(0.5, 0, 0.5)]$ in Figure 7b. In order to keep the representation of regions effective, we again triangulate regions into finer ones, which are convex and generated by finitely many points, see Fig. 7c. Finally, note that Pareto frontiers of smaller dimensions may induce regions that are faces of the projective hyperplane (triangle). In this example, the vertex at $[(0, 1, 0)]$ is its own region, as it is the only direction where playing the line-action is not optimal for Minimizer. We can also see that in Fig. 4b: the red vertex corresponds to Minimizer choosing one of the "rectangular" actions (as the other action is suboptimal), the orange region to choosing the action yielding $(0, 1, 0)$, and in the yellow Minimizer is indifferent between all actions. Since these cases only arise on faces of the projective hyperplane, the decomposition into the simplicial complex of the projective hyperplane (triangle) caters for these corner cases. Note that for identifying the regions, we considered the point $[(0.5, 0, 0.5)]$, which is the turning point of preference between the two rectangles. As both of them are suboptimal in this direction, this is not necessary to get the coarsest partition. However, it is not a problem to use a finer partition (splitting the orange line and the yellow triangle), as we still have the invariant that in every region the strategy of Minimizer is constant.

4 Algorithm

4.1 Lifting the concepts from the single-dimensional case

Before giving the algorithm, we have to define extensions of the concepts of best exit and simple end component (SEC) introduced in [37] to the multi-objective setting, as intuitively discussed in the previous section. To this end, we also introduce the concept of *regions.*

Best exits. In the single-dimensional case, the best exit of an EC was just the best exiting action for the Maximizer. In the multi-dimensional setting, Maximizer cannot only pick the best exit, but first visits all targets inside the (S)EC and then use any combination of exits to achieve any desired tradeoff. The definition of best exit depends on a parameter f. This function is used to calculate the set of achievable points from an exit. We can instantiate it with $\mathfrak{A}$ to denote the actual set of achievable points, as well as with the over-approximation U; in the algorithm, we do the latter, as we do not know $\mathfrak{A}$.

Thus we define the best exit in the multi-dimensional setting (similarly as $X(\mathsf{s}, \mathsf{a})$ in Section 2.6):

$$\mathsf{BE}^f(T) := \left(dwc(\sum_{\mathsf{s} \in T} \{\mathbb{1}_{\mathcal{T}}(\mathsf{s})\}) + conv(\bigcup_{(\mathsf{s},\mathsf{a}) \in \mathsf{Exits}(T_\Box)} f(\mathsf{s}, \mathsf{a})) \right) \cap \mathbf{1}$$

[4]The neutral element $\{\vec{0}\}$ is not considered a non-empty intersection.

The first part ensures that, if a target is in the EC, all states in the EC have probability 1 to reach it; the second part takes the convex hull of the union of (Pareto sets of) all of Maximizer's exits, corresponding to randomizing over the exiting actions. For general ECs, this may give a strict over-approximation since Minimizer might prevent Maximizer from freely visiting all states and combining all actions. However, for SECs the expression is later shown exact. Note that here we use the convention $\bigcup_{\emptyset}(\cdot) = \{\vec{0}\}$, which is a neutral and minimal element. This solves the corner case of an EC without any exit.

Regions. The extension of SEC works only when partitioning the set of all possible directions into *regions*, and then applying the same ideas as in the single-dimensional case in each region separately.

Definition 4.1 (Region). A *region* is a subset $R \subseteq \mathsf{D}$ of directions.

To keep the presentation simple, we rely on a very general definition of regions at this point. We will see later in Section 4.3 how we can restrict to handling only regions that correspond to a finitely generated cone. In the following, slightly abusing notation, we sometimes view a region R as the set of points it contains, i.e. $\{v \in [0,1]^n \mid \exists \mathbf{d} \in R : [v] = \mathbf{d}\}$.

Simple ECs. In the single-dimensional case, the idea of SEC is the following: If Minimizer fixes their strategy to the optimal strategy (i.e. ignores all suboptimal actions), and in the remaining game there still exists an EC, then this EC is simple. It is the best choice of Minimizer to allow Maximizer to roam around freely in the SEC and pick the best exit. Thus, all states in the SEC have the same value, namely that of the best exit (recall, best for Maximizer).

In the multi-dimensional case, the optimal strategy of Minimizer depends on the tradeoffs between the different goals. This is why, to generalize the concept of SEC, we need to add the restriction that a set of states is a SEC for some region R, as the trade-offs between the goals are resolved in the same way in the whole region, or in other words: where the optimal strategy of Minimizer is the same for all directions in R. Formally:

Definition 4.2 (Regional *SEC*). An EC T is *a regional simple end component for some region R*, if for every direction $\mathbf{d} \in R$ and all states $s \in T$, $\mathfrak{A}(s)[\mathbf{d}] = \mathsf{BE}^{\mathfrak{A}}(T)[\mathbf{d}]$.

Note that from this definition we also know that all states in the regional *SEC* have the same value. Moreover, as we shall see, the definition implies that on R, the optimal strategy of Minimizer should be the same in all directions. Lifting this to a set of regions we have the following property:

Definition 4.3 (Consistent Partition). Let T be an EC and $f : S \to 2^{\mathbb{R}^n}$. A partition of the set D of directions into a set of regions $\mathcal{R}$ is called *consistent* w.r.t. T and f if for all $R \in \mathcal{R}$ and all $\mathbf{d}_1, \mathbf{d}_2 \in R$, $s \in T_{\bigcirc}$ and $a \in \mathsf{Av}(s)$ it holds that

$$f(s,a)[\mathbf{d}_1] = \min_{b \in \mathsf{Av}(s)} f(s,b)[\mathbf{d}_1] \iff f(s,a)[\mathbf{d}_2] = \min_{b \in \mathsf{Av}(s)} f(s,b)[\mathbf{d}_2].$$

In the other direction, we shall see that every possible regional SEC can be defined on regions of an arbitrary consistent partition. Hence, algorithmically, we shall be looking for such partitions first and then for regional SECs.

Algorithm 1 Multi-Objective Bounded Value Iteration

```
Input:
    SG G, generalized-reach. objective $\mathcal{T}$, precision $\varepsilon$
Output:
    $\mathcal{L}, \mathcal{U}$ such that $\forall \mathbf{d} \in \mathsf{D} : \mathcal{L}[\mathbf{d}] \leq \mathfrak{P}[\mathbf{d}] \leq \mathcal{U}[\mathbf{d}]$ and
    $\mathcal{U}[\mathbf{d}] - \mathcal{L}[\mathbf{d}] < \varepsilon$
1:  procedure MO-BVI(G,$\mathcal{T}$,$\varepsilon$)
2:      for each s $\in S$ do                        ▷ Initialization
3:          L(s) ← $\{\vec{0}\}$                         ▷ to the least and
4:          U(s) ← $dwc(\{\vec{1}\})$                     ▷ the greatest values
5:      repeat                               ▷ The new Bellman update $\overline{\mathfrak{B}}$
6:          L ← $\mathfrak{B}$(L)                      ▷ Standard Bellman updates
7:          U ← $\mathfrak{B}$(U)
8:          U ← DEFLATE_SECs(G, L, U)          ▷ New treatment
9:      until $\max_{\mathbf{d} \in \mathcal{D}}$ U(s_0)[d] − L(s_0)[d] < $\varepsilon$  ▷ $\varepsilon$-approximate
10:     return (L(s_0), U(s_0))
```

4.2 Algorithms

We present our overall bounded VI procedure as Algorithm 1. In the following, we provide intuitive explanations of the algorithm and its sub-procedures, as well as the proofs for the lemmata on correctness of the sub-procedures. The correctness of the whole algorithm is proven in Section 5. Section 4.3 gives more details on the effectiveness of the computation in Algorithm 3, as that pseudocode is rather mathematical and it is not trivial to see that it is indeed effectively computable and yields an effective approximation.

Algorithm 1 (MO-BVI). initializes the under- and over-approximations L and U and updates them using the new Bellman update operator $\overline{\mathfrak{B}}$. This operator first performs the standard Bellman updates and then calls the procedure DEFLATE_SECs, which we exemplified in Section 3. The intuition of the whole algorithm is, that as the under-approximation converges, eventually the correct regional SECs are found and deflated. When all regional SECs are deflated, the over-approximation approaches the true set of achievable vectors in the limit. Note that the stopping criterion can be evaluated, as the under- and over-approximation are at all times described by finitely many points, for details see Section 4.3.

Algorithm 2 Deflate candidate SECs

```
Input:
    SG G, functions L and U such that for all states $s$ : $\mathsf{L}(s) \subseteq \mathfrak{A}(s) \subseteq \mathsf{U}(s)$
Output:
    Updated upper bound $\mathsf{U}'$
1: procedure DEFLATE_SECs(G, L, U)
        ▷ In each MEC, we compute relevant regions, find all candidate SECs and decrease their upper bounds
2:      $\mathsf{U}'(s) \leftarrow \{\vec{0}\}$ for all $s \in S$                ▷ Result variable
3:      $\mathcal{M} \leftarrow \mathsf{MEC}(\mathsf{G})$    ▷ MEC decomposition of the game
4:      for each $T \in \mathcal{M}$ do
5:          $\mathcal{R} \leftarrow$ GET_REGIONS($T$, L)
6:          for each $R \in \mathcal{R}$ do
7:              $\mathcal{S} \leftarrow$ FIND_SECs($T$, L, $R$)   ▷ Candidate SECs
8:              for each $s \in T$ do
                    ▷ If in candidate SEC, deflate
9:                  if $s \in C$ for some $C \in \mathcal{S}$ then
10:                     $\mathsf{U}'(s) \leftarrow \mathsf{U}'(s) \cup (\mathsf{U}(s) \cap \mathsf{BE}^{\mathsf{U}}(C) \cap R)$
                    ▷ Otherwise, keep estimate in this region
11:                 else
12:                     $\mathsf{U}'(s) \leftarrow \mathsf{U}'(s) \cup (\mathsf{U}(s) \cap R)$
13:     return $\mathsf{U}'$
```

Algorithm 3 Compute consistent partition into regions

```
Input:
    MEC $T \subseteq S$, function L such that for all states $s$ : $\mathsf{L}(s) \subseteq \mathfrak{A}(s)$
Output:
    Consistent partition $\mathcal{R}$ w.r.t. $T$ and L
1: procedure GET_REGIONS($T$, L)
2:      $\mathcal{R} \leftarrow \{\mathsf{D}\}$                ▷ initialize with trivial partition
3:      for $s \in T_{\bigcirc}$ do
4:          for each $B \subseteq \mathsf{Av}(s)$ do
5:              $R_B \leftarrow \{\mathbf{d} \in \mathsf{D} \mid B = \arg\min_{a \in \mathsf{Av}(s)} \mathsf{L}(s,a)[\mathbf{d}]\}$
6:          $\mathcal{R}' = \{R_B \mid B \subseteq \mathsf{Av}(s), R_B \neq \emptyset\}$
7:          $\mathcal{R} \leftarrow$ common refinement of $\mathcal{R}$ and $\mathcal{R}'$
8:      return $\mathcal{R}$
```

***Algorithm* 2 (**DEFLATE_SECs**).** is the heart of our new algorithm. It implements the correct handling of end components, ensuring convergence of the upper and the lower approximation to the *same* fixpoint. As every SEC is an EC and every EC is a subset of a MEC, the algorithm first computes the MEC-decomposition. Then, for each MEC we compute a consistent partition of the set of directions into regions using Algorithm 3. Finally, Algorithm 2 updates the over-approximation of every state in the considered MECs. It does so piece by piece, region by region; this is why in Lines 10 and 12 we always intersect with R, restricting the update to points in the current region, and take the union with the intermediate result U', adding all the points from the previous iterations of the loop over $\mathcal{R}$. If a state is part of a regional SEC C (as detected by Algorithm 4), the upper bound in the current region is reduced to $\mathsf{BE}^{\mathsf{U}}(C)$, i.e. to the best exit from the regional SEC. If a state is not in a candidate SEC for the current region, its upper bound does not change. Note that the best exit depends on U, our current best over-approximation. The intersection with $\mathsf{U}(s)$ ensures that deflate is monotonic. Formally, we have the following:

Algorithm 4 Find candidate SECs

```
Input: Under-approximation L, EC $T$, region $R$ from a consistent partition w.r.t. $T$ and $L$
Output: Set of Regional SECs for $R$, according to L
1: procedure FIND_SECs(L,$T$,$R$) $T \subseteq S$, L, region $R$
2:      $\mathbf{d} \leftarrow$ arbitrary element of $R$
3:      $\mathsf{Av}' \leftarrow \mathsf{Av}$
4:      for each $s \in T_{\bigcirc}$ do
            ▷ Keep only optimal Minimizer actions
5:          $\mathsf{Av}'(s) \leftarrow \{a \in \mathsf{Av}(s) \mid \mathsf{L}(s,a)[\mathbf{d}] = \min_{b \in \mathsf{Av}(s)} \mathsf{L}(s,b)[\mathbf{d}]\}$
6:      return $\mathsf{MEC}(T|_{\mathsf{Av}'})$    ▷ MEC decomposition on $T$ with actions restricted to Av′
```

Lemma 4.4 (DEFLATE is monotonic and sound)**.** *Given a game* G *with correct upper and lower bounds* U *and* L *(i.e.* $\forall s \in S : \mathsf{L}(s) \subseteq \mathfrak{A}(s) \subseteq \mathsf{U}(s)$*),* $\mathsf{U}' = \mathsf{DEFLATE\_SECs}(\mathsf{G}, \mathsf{L}, \mathsf{U})$ *has the following properties: For all states* $s \in S$*,*

- $\mathsf{U}'(s) \subseteq \mathsf{U}(s)$ *(Monotonicity),*
- $\mathfrak{A}(s) \subseteq \mathsf{U}'(s)$ *(Soundness),*

Proof. For monotonicity notice that due to line 10, $\mathsf{U}'(s)$ is obtained by intersecting $\mathsf{U}(s)$ with $\mathsf{BE}^{\mathsf{U}}(C)$ on each region $R \in \mathcal{R}$, which makes sure that $\mathsf{U}'(s) \subseteq \mathsf{U}(s)$ in the end. For the second item, we have that $\mathfrak{A}(s') \subseteq \mathsf{U}'(s')$ for all states s' by assumption. Recall that $\mathsf{BE}^{\mathsf{U}}(\mathfrak{A})$ is the set of points achievable from $s \in C$ assuming that Maximizer has control over all states in C. Clearly, $\mathfrak{A}(s) \subseteq \mathsf{BE}^{\mathfrak{A}}(C) \subseteq \mathsf{BE}^{\mathsf{U}}(\mathfrak{A})$, which proves soundness (see [3, Appendix A.1] for details). □

***Algorithm* 3 (**GET_REGIONS**).** has to return a consistent partition of the set of directions D, i.e. for all directions in a region, the optimal strategy of Minimizer needs to be the same. To do that, for every state in the given MEC, we partition the set of directions into regions according to the optimal strategy of Minimizer, i.e. which actions are optimal in the region[5]. Then we take the *common refinement* of all these partitions. The common refinement of two partitions $\mathcal{R}_1$ and $\mathcal{R}_2$ is defined as the coarsest partition $\mathcal{R}$ such that for all $R_1 \in \mathcal{R}_1$, $R_2 \in \mathcal{R}_2$

[5]The implementation suggested in Section 4.3 actually computes regions for all orderings of actions. It then describes the regions with the same optimal actions as a union of all regions where these actions are at the top of the ordering.

we have $R_1 \cap R_2 \in \mathcal{R}$. Notice that the common refinement of any number of consistent partitions (w.r.t. the same T and L) is again consistent. Intuitively, in every resulting region the strategy of all Minimizer states in the MEC is constant. Formally, we have the following lemma:

Lemma 4.5 (GET_REGIONS is sound)**.** *For any set of states T and bound function L, the set of regions $\mathcal{R}$ returned by procedure* GET_REGIONS*(T, L) is a consistent partition.*

Proof. We simply consider for every subset $B \subseteq \mathsf{Av}(s)$, $s \in T_{\bigcirc}$, the region R where the actions in B are all optimal. This yields a partition $\mathcal{R}' = \{R_B \mid B \subseteq \mathsf{Av}(s), R_B \neq \emptyset\}$ which is consistent w.r.t. $\{s\}$ and L. We repeat this for all $s \in T_{\bigcirc}$ and take the common refinement of all partitions obtained in this way, yielding a consistent partition for the whole EC T and L. See the next section on how to technically implement these operations effectively. □

Algorithm 4 (*FIND_SECs*). is very similar to the single-dimensional case ([37, Alg. 2]). The difference is that in the multi-objective setting we cannot just fix the strategy of Minimizer and compute the ECs in the resulting SG. We have to pick a direction from the region and consider the strategy of Minimizer w.r.t. that direction. Since we know that the given region is from a consistent partition by assumption on the input (which is true due to Lemma 4.5), Minimizer's optimal strategy is the same for all directions in the input region. Thus the direction can be arbitrarily chosen from that region. We stress that FIND_SECs is called with the current under-approximation and returns only those state sets, which according to the current lower bound form regional SECs; these need not actually be regional SECs according to $\mathfrak{A}$. However, as sketched in the proof of Lemma 4.4, deflation is so conservative that it is sound given any EC. The required property of FIND_SECs is that it eventually finds the correct regional SECs when L converges to $\mathfrak{A}$ close enough, or formally:

Lemma 4.6 (FIND_SECs is sound)**.** *For $T \subseteq S$ and a region R from a consistent partition, it holds that $X \in$ FIND_SECs$(T, \mathfrak{A}, R)$ if and only if X is an inclusion-maximal SEC for region R.*

Proof. Since R is from a consistent partition, we can pick any direction $\mathbf{d} \in R$ and identify Minimizer's optimal actions for the whole region R as in line 5. Let X be a MEC returned by FIND_SECs. Then within this EC, Minimizer only has optimal actions for region R and thus, it does not matter how exactly these choices are resolved – in particular, it does not make a difference if Maximizer takes over control of Minimizer's states as explained earlier. But then, from each $s \in X$, Maximizer can achieve precisely $\mathsf{BE}^{\mathfrak{A}}(X)$. Thus X is an inclusion-maximal SEC for region R. □

4.3 Effectiveness of GET_REGIONS

In this section we describe GET_REGIONS in more detail and argue why the computation is effective. As discussed in Section 3, regions in our context bear some resemblance to regions of timed automata [1]. We first recall some geometric notions from e.g. [36] that are necessary to talk about the representation of the considered objects:

A (k)*-simplex* is a k-dimensional polytope given as the convex hull of $k+1$ affinely independent vertices. Intuitively, a simplex is a point, line segment, triangle, tetrahedron etc. For example, considering Figure 7b, the point $(0.5, 0, 0.5)$ is a 0-simplex, the line between this point and $(1, 0, 0)$ is a 1-simplex, and the whole triangle is a 2-simplex. A *face* of a k-dimensional simplex is the convex hull of a non-empty subset of the $k+1$ points making up the simplex. A *facet* of a k-dimensional simplex is a natural face, i.e. a face that uses exactly k points. For example, for a 2-simplex which is a triangle, the triangle itself, the 3 edges and 3 vertices are all faces. Each face is also a simplex. Only the three edges are facets.

A *simplicial complex (SC)* is a set of simplices closed under taking faces, i.e. every face of a simplex in the SC is also part of the SC. It also satisfies the property that a non-empty intersection of any two simplices in the SC is a face of both the simplices. Using Figure 7b again: Consider the SC containing the two lines (1-simplices) between $(0.5, 0, 0.5)$ and $(1, 0, 0)$ as well as between $(0.5, 0, 0.5)$ and $(0, 0, 1)$. It also has to contain the point (0-simplex) $(0.5, 0, 0.5)$, as that is the intersection of the lines. Additionally, the points $(1, 0, 0)$ and $(0, 0, 1)$ need to be in the SC, as they are the faces of the lines. In order to represent (i) partitions (disjoint decompositions) and (ii) open regions, as discussed already in Section 3.3, we consider the open version: we subtract from each simplex all its facets and, abusing the notation, call them simplices and their union SC.

The invariant of our computation is that all partitions into regions as well as the Pareto frontiers are represented as finite unions of SCs. The partitions decompose (triangulate) D, the part of the projective hyperplane that is in the non-negative orthant (n-dimensional analog of the first quadrant), which can thus itself be seen as $(n-1)$-simplex; the Pareto frontiers are given by linear functions on the areas defined by regions, hence consists of SCs in the non-negative orthant (of course, generally not arranged in a hyperplane). Altogether, since simplices can be stored as the set of their vertices, we can effectively represent these partitions and frontiers by finite sets of finite sets of points.

For the computation of GET_REGIONS on Line 5, we can compute the intersections of all pairs of Pareto frontiers of the available actions as they are piece-wise linear with finitely many pieces, and we obtain a finite partition. The projected intersections then become k-simplices for some $k > 0$ (the intersection of Pareto frontiers can be points, lines, planes and so on as seen in Fig. 7 and 5). Similarly, on Line 7, starting from two finite partitions, their common refinement after the respective triangulation, as e.g. in Figure 6, is also finite and an SC. Recall the base case for the partition is the SC of the projective $n-1$-simplex.

Finally, the resulting approximation of $\mathfrak{A}$ is effective since, given a direction $\mathbf{d}$, we can identify its region and the respective simplex on the Pareto frontiers L and U and their value in the intersection with $\mathbf{d}$. For effectiveness of the stopping criterion on Line 9 of Algorithm 1, we additionally note that we only need to test for each simplex the differences in its generating points (more precisely the limits as the simplex is open) since the difference is a linear function on each of the finitely many pieces of the approximation.

5 Correctness Proof of Algorithm MO-BVI

Our new Bellman operator $\overline{\mathfrak{B}}$ defined as one application of the loop body of Algorithm 1 is a higher order operator transforming pairs of the estimate functions: the two estimate functions $\mathsf{L}, \mathsf{U} \in S \to 2^{[0,1]^n}$ for the under-/over-approximation are transformed into a pair with the modified under- and over-approximation. It can thus be seen as a function of type

$$\overline{\mathfrak{B}} : \left(S \to 2^{[0,1]^n} \times 2^{[0,1]^n}\right) \to \left(S \to 2^{[0,1]^n} \times 2^{[0,1]^n}\right).$$

We fix an SG $\mathsf{G} = (S, S_{\Box}, S_{\bigcirc}, \mathsf{s}_0, \mathsf{A}, \mathsf{Av}, \delta)$ and a generalized-reachability objective $\mathcal{T}$ for the following proofs and implicitly use them as parameters of $\overline{\mathfrak{B}}$. Note that for all states $\mathsf{s} \in S$, $\mathsf{U}_0(\mathsf{s}) = dwc(\{\vec{1}\})$ respectively $\mathsf{L}_0(\mathsf{s}) = \{\vec{0}\}$ are set by the initialization.

We consider the sequence $(\mathsf{L}_i, \mathsf{U}_i) := \overline{\mathfrak{B}}^i(\mathsf{L}_0, \mathsf{U}_0)$, $i \in \mathbb{N}$, output by our algorithm. We also use the notation $\mathsf{L}_\infty := \lim_{i\to\infty} \mathsf{L}_i := \bigcup_{i\geq 0} \mathsf{L}_i$ and $\mathsf{U}_\infty := \lim_{i\to\infty} \mathsf{U}_i := \bigcap_{i\geq 0} \mathsf{U}_i$.

Proposition 5.1. ***Soundness***
Algorithm 1 computes for each state $\mathsf{s} \in S$ a sequence of monotonic over- and under-approximations of $\mathfrak{A}(\mathsf{s})$, i.e. $\forall i \in \mathbb{N}$: $\mathsf{L}_i(\mathsf{s}) \subseteq \mathfrak{A}(\mathsf{s}) \subseteq \mathsf{U}_i(\mathsf{s})$ and for $i < j, \mathsf{L}_i(\mathsf{s}) \subseteq \mathsf{L}_j(\mathsf{s})$ as well as $\mathsf{U}_i(\mathsf{s}) \supseteq \mathsf{U}_j(\mathsf{s})$.

Proposition 5.2. ***Convergence from below***
$\forall$ states $\mathsf{s} \in S$ and all directions $\mathbf{d} \in \mathsf{D}$: $\mathsf{L}_\infty(\mathsf{s})[\mathbf{d}] = \mathfrak{A}(\mathsf{s})[\mathbf{d}]$.

Proposition 5.3. ***Convergence from above***
$\forall$ states $\mathsf{s} \in S$ and all directions $\mathbf{d} \in \mathsf{D}$: $\mathsf{U}_\infty(\mathsf{s})[\mathbf{d}] = \mathfrak{A}(\mathsf{s})[\mathbf{d}]$.

Note that for all directions $\mathbf{d}$ and for all $\mathsf{s} \in S$ by definition $\mathfrak{A}(\mathsf{s})[\mathbf{d}] = \mathfrak{P}(\mathsf{s})[\mathbf{d}]$. Using this and the three propositions, we can prove the main theorem.

Theorem 5.4. *Algorithm 1 computes convergent monotonic over- and under-approximations of $\mathfrak{P}(\mathsf{s})$ for each $\mathsf{s} \in S$. Since it is convergent, for every $\epsilon > 0$ there exists an i, such that for every $\mathsf{s} \in S$ and direction $\mathbf{d} \in \mathsf{D}$: $\mathsf{U}_i(\mathsf{s})[\mathbf{d}] - \mathsf{L}_i(\mathsf{s})[\mathbf{d}] < \epsilon$. So by instantiating s with s_0, we solve the problem posed in Section 2.5.*

Proof of Propositions 5.1 and 5.2. Note that for all $i \in \mathbb{N}$ it holds that $\mathsf{L}_i = \mathfrak{B}^i(\mathsf{L}_0)$, since DEFLATE_SECs does not change the under-approximation. [9, Proposition 8] proves that $\mathfrak{B}$ is order-preserving, i.e. monotonic, and that it converges to the unique least fixpoint $\mathfrak{A}$ when repeatedly applied to the bottom element of a complete partial order. The least possible lower bound assigns $\vec{0}$ to all S, since there is no smaller vector that can be assigned to a state. This is exactly the definition of L_0, which implies that for all $\mathsf{s} \in S$, the closure of $\mathsf{L}_\infty(\mathsf{s})$ equals the closure of $\mathfrak{A}(\mathsf{s})$, which implies Proposition 5.2.

For the soundness of the over-approximation we require that the additional operation, namely DEFLATE_SECs, performed by $\overline{\mathfrak{B}}$ is sound (proven in Lemma 4.4). The monotonicity of the under- and over-approximation follows from the monotonicity of $\mathfrak{B}$ [9, Proposition 8] and of DEFLATE_SECs (Lemma 4.4) Thus we can deduce Proposition 5.1. ◻

It only remains to show Proposition 5.3. As a key ingredient for the proof we will use the following:

Lemma 5.5 (Fixpoint). *$\overline{\mathfrak{B}}(\mathsf{L}_\infty, \mathsf{U}_\infty) = (\mathsf{L}_\infty, \mathsf{U}_\infty)$, i.e. the limit of $\overline{\mathfrak{B}}$ is also a fixpoint.*

Proof idea. We only need to argue about the second component U_∞. If we did not have a fixpoint, then a further application of $\overline{\mathfrak{B}}$ would find a SEC T for some region R and decrease the over-approximation U_∞ for some $\mathbf{d} \in R$ and $s \in T$, i.e. $\mathsf{BE}^{\mathsf{U}_\infty}(T)[\mathbf{d}] < \mathsf{U}_\infty(s)[\mathbf{d}]$. The key idea is that since the lower approximations L_i converge to $\mathfrak{A}$, the SEC T is detected and deflated infinitely many times before convergence. But this means that $\mathsf{BE}^{\mathsf{U}_\infty}(T)[\mathbf{d}] = \mathsf{U}_\infty(s)[\mathbf{d}]$, contradiction. For more details, see [3, Appendix A.2]. ◻

Proof of Proposition 5.3. We will use the fixpoint property from Lemma 5.5 to derive a contradiction. We assume for contradiction that there is a state $\mathsf{s} \in S$ and a direction $\mathbf{d} \in \mathsf{D}$ such that $\mathsf{U}_\infty(\mathsf{s})[\mathbf{d}] \neq \mathfrak{A}(\mathsf{s})[\mathbf{d}]$. Applying the Bellman operator once more to $(\mathsf{L}_\infty, \mathsf{U}_\infty)$ results in a new upper bound U'. We will show that $\mathsf{U}' \subsetneq \mathsf{U}_\infty$. In other words, applying the loop once more decreases the over-approximation. This is a contradiction to U_∞ being a fixpoint and proves our goal.

1. Assume for contradiction, that $\exists t \in S, \mathbf{d} \in \mathsf{D} : \mathsf{U}_\infty(t)[\mathbf{d}] \neq \mathfrak{A}(t)[\mathbf{d}]$ and thus $\exists \mathbf{d}\ \mathsf{U}_\infty(t)[\mathbf{d}] > \mathfrak{A}(t)[\mathbf{d}]$ with Prop. 5.1. We fix this direction $\mathbf{d}$ and t for the rest of the proof.
2. Let $X := \{\mathsf{s} \in S \mid \Delta(\mathsf{s}) = \max_{t\in S} \Delta(t)\}$, where $\Delta(\mathsf{s}) := \mathsf{U}_\infty(\mathsf{s})[\mathbf{d}] - \mathfrak{A}(\mathsf{s})[\mathbf{d}]$ is the difference between over-approximation $\mathsf{U}_\infty(\mathsf{s})$ and achievable set $\mathfrak{A}$ in $\mathbf{d}$.
   a. We also define $\Delta(\mathsf{s}, \mathsf{a}) := \mathsf{U}_\infty(\mathsf{s}, \mathsf{a})[\mathbf{d}] - \mathfrak{A}(\mathsf{s}, \mathsf{a})[\mathbf{d}]$ for an action $\mathsf{a} \in \mathsf{Av}(s)$.
   b. By assumption, $X \neq \emptyset$ and for all $\mathsf{s} \in X : \Delta(\mathsf{s}) > 0$.
   Note: $\Delta(s), \Delta(s, a)$ and X are all defined w.r.t. the fixed direction $\mathbf{d}$ (not indicated in notation to avoid clutter).
3. $\Delta(s) > 0$ implies that $dwc(\mathbb{1}_{\mathcal{T}}(\mathsf{s}))[\mathbf{d}] = 0$, i.e. s is not contained in target sets "aligned" in direction $\mathbf{d}$ because otherwise, $\mathsf{U}_\infty(\mathsf{s})[\mathbf{d}] = \mathfrak{A}(\mathsf{s})[\mathbf{d}] = dwc(\mathbb{1}_{\mathcal{T}}(\mathsf{s}))[\mathbf{d}]$.
4. For all (s, a) exits X it holds that $\Delta(\mathsf{s}, \mathsf{a}) < \Delta(\mathsf{s})$.
   Reason: If (s, a) exits X, then $\exists \mathsf{s}' \in \mathsf{Post}(\mathsf{s}, \mathsf{a}) \setminus X$. Note

that $\Delta(s') < \Delta(s)$ by construction of X

$$\begin{aligned}\Delta(s,a) &= U_\infty(s,a)[\mathbf{d}] - \mathfrak{A}(s,a)[\mathbf{d}] \quad (\text{Definition of } \Delta(s,a))\\ &= \sum_{s'\in \mathrm{Post}(s,a)} \delta(s,a,s')\left(U_\infty(s')[\mathbf{d}] - \mathfrak{A}(s')[\mathbf{d}]\right)\\ &\qquad (\text{Definition of } \mathfrak{A}(s,a) \text{ and Step 3})\\ &= \sum_{s'\in \mathrm{Post}(s,a)} \delta(s,a,s')\Delta(s') \qquad (\text{Definition of } \Delta(s))\\ &< \Delta(s) \qquad (\text{since } t \in \mathrm{Post}(s,a) \text{ and } \Delta(t) < \Delta(s))\end{aligned}$$

5. No state in X depends on a leaving action. Formally:
   (a) $\forall s \in X_{\bigcirc}, (s,a)$ exits $X : \mathfrak{A}(s)[\mathbf{d}] < \mathfrak{A}(s,a)[\mathbf{d}]$, i.e. a leaving action leaving X cannot be optimal for Minimizer in direction $\mathbf{d}$.
   Reason: Since s is a state of Minimizer, $\forall a \in \mathrm{Av}(s)$: $U_\infty(s)[\mathbf{d}] \leq U_\infty(s,a)[\mathbf{d}]$. From this and the inequality from the previous Step 4, we get that:

$$\begin{aligned}&U_\infty(s,a)[\mathbf{d}] - \mathfrak{A}(s,a)[\mathbf{d}]\\ &= \Delta(s,a) && (\text{Definition of } \Delta)\\ &< \Delta(s) && (\text{Step 4})\\ &= U_\infty(s)[\mathbf{d}] - \mathfrak{A}(s)[\mathbf{d}] && (\text{Definition of } \Delta)\\ &\leq U_\infty(s,a)[\mathbf{d}] - \mathfrak{A}(s)[\mathbf{d}] && (s \in X_{\bigcirc})\end{aligned}$$

   Subtracting $U_\infty(s,a)[\mathbf{d}]$ and multiplying by (-1) yields the claim.
   (b) $\forall s \in X_{\square}$, $U_\infty(s)[\mathbf{d}] > \sum_{a\in \mathrm{Av}(s,a)} w_a \cdot U_\infty(s,a)$ if $w_a > 0$ for some action a exiting X. Intuitively, this means that Maximizer cannot assign positive weight to any action leaving X. The proof is similar to part (a).
6. X contains an EC because if not, then $\exists s \in X : \forall a \in \mathrm{Av}(s) : (s,a)$ exits X. But then s necessarily depends on a leaving action in the sense of the previous Step 5, contradiction.
7. Using that X contains an EC, we can show that X even contains a regional *simple* EC $Z \subseteq X$ w.r.t. to the region $\{\mathbf{d}\}$. Applying $\overline{\mathfrak{B}}$ once more to $(\mathfrak{A}, U_\infty)$, the over-approximation decreases.

   Reason: We only give high-level intuition here, as the proof is very technical. The formal details are in [3, Appendix A.3]. We prove by a large case distinction that X contains a regional SEC Z for the region $\{\mathbf{d}\}$. Since $L_\infty = \mathfrak{A}$, by Lemma 4.6 this regional SEC is found and deflated. By construction of Z, then its value is set to "depend on the outside", i.e. it assigns a positive weight on an action leaving X. Then, by Step 5, the over-approximation is reduced and we arrive at a contradiction. □

6 Conclusion

For a given $\varepsilon > 0$ and a generalized-reachability stochastic game, we compute an ε-approximation of its Pareto frontier. Our algorithm can be run as an anytime algorithm, reporting the under- and over-approximations on the frontier, due to an extended version of value iteration. We have suggested the name "bounded value iteration" as it better generalizes to higher dimensions than "interval iteration". We conjecture that this technique can be generalized to other models, such as concurrent games, and more complex objectives, such as total reward. Finally, while decidability remains open, the approximation algorithms are practically more relevant even in the single-dimensional case. Note that approximative value iteration is the default technique for analysis of MDP, although there is an exact and polynomial solution by linear programming. The reason is that the theoretical worst-case complexity of value iteration is practically not too relevant. Consequently, an efficient implementation, possibly exploring only a part of the state space using learning, as e.g. in [11, 37], may be an interesting future direction.

Acknowledgments

Pranav Ashok, Jan Křetínský and Maximilian Weininger were funded in part by TUM IGSSE Grant 10.06 (PARSEC) and the German Research Foundation (DFG) project KR 4890/2-1 "Statistical Unbounded Verification". Krishnendu Chatterjee was supported by the ERC CoG 863818 (ForM-SMArt) and Vienna Science and Technology Fund (WWTF) Project ICT15-003. Tobias Winkler was supported by the RTG 2236 UnRAVeL.

References

[1] Rajeev Alur and David L. Dill. 1994. A Theory of Timed Automata. *Theor. Comput. Sci.* 126, 2 (1994), 183–235.
[2] Pranav Ashok, Krishnendu Chatterjee, Przemyslaw Daca, Jan Kretínský, and Tobias Meggendorfer. 2017. Value Iteration for Long-Run Average Reward in Markov Decision Processes. In *CAV*. 201–221. https://doi.org/10.1007/978-3-319-63387-9_10
[3] Pranav Ashok, Krishnendu Chatterjee, Jan Kretinsky, Maximilian Weininger, and Tobias Winkler. 2020. Approximating Values of Generalized-Reachability Stochastic Games. *CoRR* abs/1908.05106 (2020).
[4] Christel Baier, Marcus Daum, Clemens Dubslaff, Joachim Klein, and Sascha Klüppelholz. 2014. Energy-Utility Quantiles. In *NASA Formal Methods*. 285–299.
[5] Christel Baier, Clemens Dubslaff, and Sascha Klüppelholz. 2014. Trade-off analysis meets probabilistic model checking. In *CSL-LICS*. 1:1–1:10.
[6] Christel Baier, Clemens Dubslaff, Sascha Klüppelholz, Marcus Daum, Joachim Klein, Steffen Märcker, and Sascha Wunderlich. 2014. Probabilistic Model Checking and Non-standard Multi-objective Reasoning. In *FASE*. 1–16.
[7] Christel Baier and Joost-Pieter Katoen. 2008. Principles of Model Checking.
[8] Nicolas Basset, Marta Z. Kwiatkowska, Ufuk Topcu, and Clemens Wiltsche. 2015. Strategy Synthesis for Stochastic Games with Multiple Long-Run Objectives. In *TACAS (Lecture Notes in Computer Science)*, Vol. 9035. Springer, 256–271.
[9] Nicolas Basset, Marta Z. Kwiatkowska, and Clemens Wiltsche. 2018. Compositional strategy synthesis for stochastic games with multiple objectives. *Inf. Comput.* 261, Part (2018), 536–587.

[10] Tomás Brázdil, Václav Brozek, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. 2014. Two Views on Multiple Mean-Payoff Objectives in Markov Decision Processes. *LMCS* 10, 1 (2014). https://doi.org/10.2168/LMCS-10(1:13)2014

[11] Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Kretínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. 2014. Verification of Markov Decision Processes Using Learning Algorithms. In *ATVA (Lecture Notes in Computer Science)*, Vol. 8837. Springer, 98–114.

[12] Tomás Brázdil, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. 2013. Trading Performance for Stability in Markov Decision Processes. In *LICS*. 331–340.

[13] Romain Brenguier and Vojtech Forejt. 2016. Decidability Results for Multi-objective Stochastic Games. In *ATVA (Lecture Notes in Computer Science)*, Vol. 9938. 227–243.

[14] Romain Brenguier and Jean-François Raskin. 2015. Pareto Curves of Multidimensional Mean-Payoff Games. In *CAV (2) (Lecture Notes in Computer Science)*, Vol. 9207. Springer, 251–267.

[15] Krishnendu Chatterjee. 2007. Markov Decision Processes with Multiple Long-Run Average Objectives. In *FSTTCS (Lecture Notes in Computer Science)*, Vol. 4855. Springer, 473–484.

[16] Krishnendu Chatterjee and Laurent Doyen. 2016. Perfect-Information Stochastic Games with Generalized Mean-Payoff Objectives. In *LICS*. ACM, 247–256.

[17] Krishnendu Chatterjee and Nathanaël Fijalkow. 2011. A reduction from parity games to simple stochastic games. In *GandALF*. 74–86. https://doi.org/10.4204/EPTCS.54.6

[18] Krishnendu Chatterjee, Vojtech Forejt, and Dominik Wojtczak. 2013. Multi-objective Discounted Reward Verification in Graphs and MDPs. In *LPAR*. 228–242.

[19] Krishnendu Chatterjee and Thomas A Henzinger. 2008. Value iteration. In *25 Years of Model Checking*. Springer, 107–138.

[20] Krishnendu Chatterjee, Thomas A. Henzinger, Barbara Jobstmann, and Arjun Radhakrishna. 2010. Gist: A Solver for Probabilistic Games. In *CAV*. 665–669. https://doi.org/10.1007/978-3-642-14295-6_57

[21] Krishnendu Chatterjee, Zuzana Kretínská, and Jan Kretínský. 2017. Unifying Two Views on Multiple Mean-Payoff Objectives in Markov Decision Processes. *Logical Methods in Computer Science* 13, 2 (2017).

[22] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, David Parker, and Aistis Simaitis. 2013. PRISM-games: A Model Checker for Stochastic Multi-Player Games. In *TACAS (Lecture Notes in Computer Science)*, Vol. 7795. Springer, 185–191.

[23] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. 2013. On Stochastic Games with Multiple Objectives. In *MFCS (Lecture Notes in Computer Science)*, Vol. 8087. Springer, 266–277.

[24] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. 2013. *On Stochastic Games with Multiple Objectives*. Technical Report. 266–277 pages.

[25] Taolue Chen, Marta Z. Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. 2013. Synthesis for Multi-objective Stochastic Games: An Application to Autonomous Urban Driving. In *QEST*. 322–337. https://doi.org/10.1007/978-3-642-40196-1_28

[26] Chih-Hong Cheng, Alois Knoll, Michael Luttenberger, and Christian Buckl. 2011. GAVS+: An Open Platform for the Research of Algorithmic Game Solving. In *ETAPS*. 258–261. https://doi.org/10.1007/978-3-642-19835-9_22

[27] Anne Condon. 1992. The complexity of stochastic games. *Information and Computation* 96, 2 (1992), 203–224.

[28] Anne Condon. 1993. On Algorithms for Simple Stochastic Games. In *Advances in Computational Complexity Theory, volume 13 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 51–73.

[29] Costas Courcoubetis and Mihalis Yannakakis. 1995. The Complexity of Probabilistic Verification. *J. ACM* 42, 4 (July 1995), 857–907.

[30] Kousha Etessami, Marta Z. Kwiatkowska, Moshe Y. Vardi, and Mihalis Yannakakis. 2008. Multi-Objective Model Checking of Markov Decision Processes. *Logical Methods in Computer Science* 4, 4 (2008).

[31] J.A. Filar, D. Krass, and K.W Ross. 1995. Percentile performance criteria for limiting average Markov decision processes. *Automatic Control, IEEE Transactions on* 40, 1 (Jan 1995), 2–10.

[32] Vojtech Forejt, Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. 2011. Quantitative Multi-objective Verification for Probabilistic Systems. In *TACAS*. 112–127. https://doi.org/10.1007/978-3-642-19835-9_11

[33] Vojtech Forejt, Marta Z. Kwiatkowska, and David Parker. 2012. Pareto Curves for Probabilistic Model Checking. In *ATVA (Lecture Notes in Computer Science)*, Vol. 7561. Springer, 317–332.

[34] Christoph Haase, Stefan Kiefer, and Markus Lohrey. 2017. Computing quantiles in Markov chains with multi-dimensional costs. In *LICS*. 1–12.

[35] Serge Haddad and Benjamin Monmege. 2018. Interval iteration algorithm for MDPs and IMDPs. *Theor. Comput. Sci.* 735 (2018), 111–131.

[36] A. Hatcher. 2002. *Algebraic Topology*. Cambridge University Press. https://books.google.de/books?id=BjKs86kosqgC

[37] Edon Kelmendi, Julia Krämer, Jan Kretínský, and Maximilian Weininger. 2018. Value Iteration for Simple Stochastic Games: Stopping Criterion and Learning Algorithm. In *CAV*. https://doi.org/10.1007/978-3-319-96145-3_36

[38] Marta Kwiatkowska, David Parker, and Clemens Wiltsche. 2016. PRISM-Games 2.0: A Tool for Multi-objective Strategy Synthesis for Stochastic Games. In *TACAS (Lecture Notes in Computer Science)*, Vol. 9636. Springer, 560–566.

[39] Marta Kwiatkowska, David Parker, and Clemens Wiltsche. 2018. PRISM-games: verification and strategy synthesis for stochastic multi-player games with multiple objectives. *STTT* 20, 2 (2018), 195–210.

[40] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *CAV (Lecture Notes in Computer Science)*, Vol. 6806. Springer, 585–591.

[41] H. Brendan Mcmahan, Maxim Likhachev, and Geoffrey J. Gordon. 2005. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML 05*. 569–576.

[42] Christos H. Papadimitriou and Mihalis Yannakakis. 2000. On the Approximability of Trade-offs and Optimal Access of Web Sources. In *FOCS*. IEEE Computer Society, 86–92.

[43] Martin L. Puterman. 2014. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons.

[44] Mickael Randour, Jean-François Raskin, and Ocan Sankur. 2015. Percentile Queries in Multi-dimensional Markov Decision Processes. In *CAV (1) (Lecture Notes in Computer Science)*, Vol. 9206. Springer, 123–139.

[45] Mickael Randour, Jean-François Raskin, and Ocan Sankur. 2017. Percentile queries in multi-dimensional Markov decision processes. *Formal Methods in System Design* 50, 2-3 (2017), 207–248. https://doi.org/10.1007/s10703-016-0262-7

[46] María Svorenová and Marta Kwiatkowska. 2016. Quantitative verification and strategy synthesis for stochastic games. *Eur. J. Control* 30 (2016), 15–30. https://doi.org/10.1016/j.ejcon.2016.04.009

[47] Yaron Velner. 2015. Robust Multidimensional Mean-Payoff Games are Undecidable. In *FoSSaCS*. Springer, 312–327.

G dtControl 2.0: Explainable Strategy Representation via Decision Tree Learning Steered by Experts (TACAS 2021)

This is an exact reprinting of the paper which has been published as a **peer reviewed conference paper**.

Pranav Ashok, Mathias Jackermeier, Jan Kretinsky, Christoph Weinhuber, Maximilian Weininger, and Mayank Yadav. "dtControl 2.0: Explainable Strategy Representation via Decision Tree Learning Steered by Experts". In: TACAS (2). Vol. 12652. Lecture Notes in Computer Science. Springer, 2021, pp. 326–345. doi: 10.1007/978-3-030-72013-1_17

Summary

Recent advances have shown how decision trees are apt data structures for concisely representing strategies (or controllers) satisfying various objectives. Moreover, they also make the strategy more explainable. The recent tool dtControl had provided pipelines with tools supporting strategy synthesis for hybrid systems, such as SCOTS and Uppaal Stratego. We present dtControl 2.0, a new version with several fundamentally novel features. Most importantly, the user can now provide domain knowledge to be exploited in the decision tree learning process and can also interactively steer the process based on the dynamically provided information. To this end, we also provide a graphical user interface. It allows for inspection and re-computation of parts of the result, suggesting as well as receiving advice on predicates, and visual simulation of the decision-making process. Besides, we interface model checkers of probabilistic systems, namely STORM and PRISM and provide dedicated support for categorical enumeration-type state-variables. Consequently, the controllers are more explainable and smaller.

For more details on this publication, we refer to Chapter 6 of the main body.

Contribution

Contribution of Maximilian Weininger	
Development and conceptual design of the research project	40%
Discussion and development of ideas	30%
Composition and revision of the manuscript	75%
Implementation	5%
Experimental evaluation	15%

dtControl 2.0: Explainable Strategy Representation via Decision Tree Learning Steered by Experts [⋆]

Pranav Ashok[1], Mathias Jackermeier[1], Jan Křetínský[1], Christoph Weinhuber[1](✉), Maximilian Weininger[1], and Mayank Yadav[2]

[1] Technical University of Munich, Munich, Germany
`firstname.lastname@tum.de`
[2] Department of Computer Science and Engineering, I.I.T. Delhi, New Delhi, India
`cs1180356@iitd.ac.in`

Abstract. Recent advances have shown how decision trees are apt data structures for concisely representing strategies (or controllers) satisfying various objectives. Moreover, they also make the strategy more explainable. The recent tool `dtControl` had provided pipelines with tools supporting strategy synthesis for hybrid systems, such as `SCOTS` and `Uppaal Stratego`. We present `dtControl 2.0`, a new version with several fundamentally novel features. Most importantly, the user can now provide domain knowledge to be exploited in the decision tree learning process and can also interactively steer the process based on the dynamically provided information. To this end, we also provide a graphical user interface. It allows for inspection and re-computation of parts of the result, suggesting as well as receiving advice on predicates, and visual simulation of the decision-making process. Besides, we interface model checkers of probabilistic systems, namely `STORM` and `PRISM` and provide dedicated support for categorical enumeration-type state variables. Consequently, the controllers are more explainable and smaller.

Keywords: Strategy representation · Controller representation · Decision Tree · Explainable Learning · Hybrid systems · Probabilistic Model Checking · Markov Decision Process

1 Introduction

A *controller* (also known as strategy, policy or scheduler) of a system assigns to each state of the system a set of actions that should be taken in order to achieve a certain goal. For example, one may want to satisfy a given specification of a robot's

[⋆] This work has been partially supported by the German Research Foundation (DFG) project No. 383882557 *SUV* (KR 4890/2-1), No. 427755713 *GOPro* (KR 4890/3-1) and the TUM International Graduate School of Science and Engineering (IGSSE) grant 10.06 *PARSEC*. We thank Tim Quatman for implementing JSON-export of strategies in `STORM` and Pushpak Jagtap for his support with the `SCOTS` models.

behaviour or exhibit a concurrency bug appearing only in some interleaving. It is desirable that the controllers possess several additional properties, besides achieving the goal, in order to be usable in practice. Firstly, controllers should be *explainable*. Only then can they be understood, trusted and implemented by the engineers, certified by the authorities, or used in the debugging process [11]. Secondly, they should be *small* in size and efficient to run. Only then they can be deployed on embedded devices with limited memory of a few kilobytes, while the automatically synthesized ones are orders of magnitude larger [49]. Thirdly, whenever the primary goal, e.g. functional correctness, is accompanied by a secondary criterion, e.g. energy efficiency, they should be *performant* with respect to this criterion.

Automatic controller synthesis is able to provide controllers for a given goal in various domains, such as probabilistic systems [32, 17], hybrid systems [45, 16, 30, 19] or reactive systems [35]. In some cases, even the performance can be reflected [16]. However, despite recent interest in explainability in connection to AI-based controllers [2] and despite typically small memories of embedded devices, automatic techniques for controller synthesis mostly fall short of producing small explainable results. A typical outcome is a controller in the form of a look-up table, listing the actions for each possible state, or a binary decision diagram (BDD) [14] representation thereof. While the latter reduces the size to some extent, none of the two representations is explainable: the former due to its size, the latter due to the bit-level representation with all high-level structure lost. Instead, learning representations in the form of decision trees (DT) [38] has been recently explored to this end [7, 3]. DTs turn out to be usually smaller than BDD but do not drown to the bit level and are generally well known for their interpretability and explainability due to their simple structure. However, despite showing significant potential, the state-of-the-art tool dtControl [4] uses predicates without natural interpretation, and moreover, the best size reductions are achieved using *determinization*, i.e. making the controller less permissive, which negatively affects performance [7].

Example 1 (Motivating example). Consider the cruise control model of [34], where we want to control the speed of our car so that it never crashes into the car in front while, as a secondary performance objective, keeping the distance between the two cars small.

A safe controller for the this model as returned by `Uppaal Stratego`, is a lookup table of size 418 MB with 300,000 lines. The respective BDD has 1,448 nodes with all information bit-blasted. Using adaptations of standard DT-construction algorithms, as implemented in `dtControl`, we can get a DT with 987 nodes, which is still too large to be explained. Using determinization techniques, the controller can be compressed to 3 nodes! However, then the DT allows only to decelerate until the minimum velocity. This is safe, as we cannot crash into the car in front, but it does not even attempt at getting close to the front car, and thus has a very bad performance.

One can find a strategy with optimal performance, retaining the maximal permissiveness, not determinizing at all, which can be represented by a DT with 11 nodes. A picture of this DT as well as reasoning how to derive the predicates from the kinematic equations is in the extended version of this paper [5, Appendix A].

However, exactly because the predicates are based on the *domain knowledge*, namely the kinematic equations, they take the form of *algebraic predicates* and not simply linear predicates, which are the only ones in `dtControl` and commonly in the machine-learning literature on DTs. $\triangle$

This motivating example shows that using domain knowledge and algebraic predicates, available now in `dtControl 2.0`, one can get smaller representation than when using existing heuristics. Further, it improves the performance of the DT, and it is easily explainable, as it is based on domain knowledge. In fact, the discussed controller is so explainable that it allowed us to find a bug in the original model. In general, using `dtControl 2.0` a domain expert can try to compress the controller, thus gain more insight and validate that it is correct. Another example of this has been reported from the use of `dtControl` in the manufacturing domain [31].

While automatic synthesis of good predicates from the domain knowledge may seem as distant as automatic synthesis of program invariants or automatic theorem provers, we adopt the philosophy of those domains and offer *semi-automatic techniques.*

Additionally, if not performance but only safety of a controller is relevant, we can still benefit from determinization without drawbacks. To this end, we also provide a new determinization procedure that generalizes the extremely successful MaxFreq technique of [4] and is as good or better on all our examples.

To incorporate the changes just discussed, namely algebraic predicates, semi-automatic approach, and better determinization, we have also reworked the tool and its interfaces. To begin with, the software architecture of `dtControl 2.0` is now very modular and allows for easy further modifications, as well as adding support for new synthesis tools. In fact, we have already added parsers for the tools `STORM` [17] and `PRISM` [32], and thus we support probabilistic models as well. Since these models also contain categorical (or enumeration-type) variables, e.g. protocol states, we have also added support for categorical predicates. Furthermore, we added a graphical user interface that not only is easier to use than the command-line interface, but also allows to inspect the DT, modify and retrain parts of it, and simulate runs of the model under its control, further increasing the possibilities to explain the DT and validate the controller.

Summing up, the main improvements of `dtControl 2.0` over the previous version [4] are the following:

- Support of algebraic predicates and categorical predicates
- Semi-automatic interface and GUI with several interactive modes
- New determinization procedure
- Interfaces for model checkers PRISM and Storm and experimental evidence of improvements on probabilistic models compared to BDD

The paper is structured as follows. After recalling necessary background in Section 2, we give an overview of the improvements over the previous version of the tool from the global perspective in Section 3. We detail on the algorithmic contribution in Sections 4 (predicate domains), 5 (predicate selection) and 6 (determinization). Section 7 provides experimental evaluation and Section 8 concludes.

Related work. DTs have been suggested for representing controllers of and counterexamples in probabilistic systems in [11], however, the authors only discuss approximate representations. The ideas have been extended to other setting, such as reactive synthesis [12] and hybrid systems [7]. More general linear predicates have been considered in leaves of the trees in [3]. `dtControl 2.0` contains the DT induction algorithms from [7, 3]. The differences to the previous version of the tool `dtControl` [4] are summarized above and schematically depicted in Figure 2.

Besides, DTs have been used to represent and learn strategies for safety objectives in [40] and to learn program invariants in [21]. Further, DTs were used for representing the strategies during the model checking process, namely in strategy iteration [10] or in simulation-based algorithms [42]. Representing controllers exactly using a structure similar to DT (mistakenly claimed to be an algebraic decision diagram) was first suggested by [22], however, no automatic construction algorithm was provided.

The idea of non-linear predicates has been explored in [28]. In that work, however, it is not based on domain knowledge, but rather on projecting the state-space to higher dimensions.

BDDs [14] have been commonly used to represent strategies in planning [15], symbolic model checking [32] as well as to represent hybrid system controllers [45, 30]. While BDD [14] operate only on Boolean variables, they have the advantage of being diagrams and not trees. Moreover, they correspond to Boolean functions that can be implemented on hardware easily. [18] proposes an automatic compression technique for numerical controllers using BDDs. Similar to our work, [49] considers the problem of obtaining concise BDD representation of controllers and presents a technique to obtain smaller BDDs via determinization. However, BDDs are difficult to explain due to variables being bit-blasted and their size is very sensitive to the chosen variable ordering. An extension of BDDs, algebraic or multi-terminal decision diagrams (ADD/MTBDD) [8, 20], have been used in reinforcement learning for strategy synthesis [26, 47]. ADDs extend BDDs with the possibility to have multiple values in the terminal nodes, but the predicates still work only on boolean variables, retaining the disadvantages of BDDs.

2 Decision tree learning for controller representation

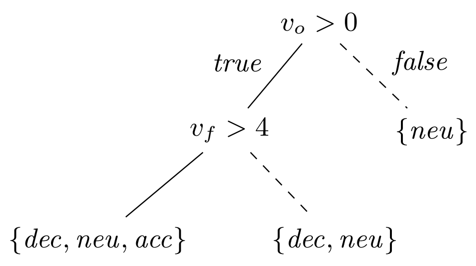
In this section, we briefly describe how controllers can be represented as decision trees as in [4]. We give an exemplified overview of the method, pinpointing the role of our algorithmic contributions.

A (non-deterministic, also called permissive) *controller* is a map $C : S \mapsto 2^A$ from states to non-empty sets of actions. This notion of a controller is fairly general; the only requirement is that it has to be memoryless and non-randomized. These kind of controllers are optimal for many tasks such as expected (discounted) reward, reachability or parity objectives. Moreover, even finite-memory controllers can be written in this form by considering the product of the state space with the finite memory as the domain, for example, like in LTL model checking.

Decision trees (DT), e.g. [38], are trees where every leaf node is labelled with a non-empty set of actions and every inner node is labelled with a *predicate* $\rho : S \mapsto \{true, false\}$.

v_o	v_f	d	actions
0	0	5	$\{neu\}$
2	6	10	$\{dec, neu, acc\}$
2	6	15	$\{dec, neu, acc\}$
4	4	15	$\{dec, neu\}$

(a)



(b)

Fig. 1: An example controller based on the cruise-control model in the form of a lookup table (left), and the corresponding decision tree (right).

Example 2 (Decision tree representation). As an example, consider the controller given in Figure 1a. It is a subset of the real cruise-control case study from the motivating Example 1. A state is a 3-tuple of the variables v_o, v_f and d, which denote the velocity of our car, the front car and the distance between the cars respectively. In each state, our car may be allowed to perform a subset of the following set of actions: decelerate (*dec*), stay in neutral (*neu*) or accelerate (*acc*). A DT representing this lookup table is depicted in Figure 1b.

Given a state, for example $v_o = v_f = 4, d = 10$, the DT is evaluated as follows: We start at the root and, since it is an inner node, we evaluate its predicate $v_o > 0$. As this is true, we follow the true branch and reach the inner node labelled with the predicate $v_f > 4$. This is false, so we follow the false branch and reach the leaf node labelled $\{dec, neu\}$. Hence, we know that all three possibilities of decelerating, staying neutral and accelerating are allowed by the controller. △

To construct a DT representation of a given controller, the following recursive algorithm may be used. Note that it is heuristic since constructing an optimal binary decision tree is an NP-complete problem [27].

Base case: If all states in the the controller agree on their set of actions B (i.e. for all states s we have $C(s) = B$), return a leaf node with label B.

Recursive case: Otherwise, we split the controller. For this, we select a predicate ρ and construct an inner node with label ρ. Then we partition the controller by evaluating the predicate on the state space, and recursively construct one DT for the sub-controller on states $\{s \in S \mid \rho(s)\}$ where the predicate is true, and one for the sub-controller where it is false. These controllers are the children of the inner node with label ρ and we proceed recursively.

For selecting the predicate, we consider two hyper-parameters: The *domain* of the predicates (see Section 4) and the way to *select* predicates (see Section 5). The selection is typically performed by selecting the predicate with the lowest *impurity*; this is a measure for how homogenous (or "pure") the controller is after the split, in other words the degree to which all the states agree on their actions.

We also consider a third hyper-parameter of the algorithm, namely *determinization* by *safe early stopping* (see Section 6). This modifies the base case as follows: if all states in the controller agree on at least one action a (i.e. for all states s we have $a \in C(s)$), then we return a leaf node with label $\{a\}$. This variant of early stopping ensures that, even though the controller is not represented exactly, still for every state a safe action is allowed.

Hence, if the original controller satisfies some property, e.g. that a safe set of states is never left, the DT construction algorithm ensures that this property is retained. This is because our algorithm represents the strategy exactly (or a safe subset, in case of determinization) and does not generalize as DTs typically do in machine learning. DTs are suitable for both tasks, as both rely on the strength of DTs exploiting underlying structure.

Remark 1. Note that for some types of objectives such as reachability, determinization of permissive strategies might lead to a violation of the original guarantees. For example, consider a strategy that allows both a self-looping and a non-self-looping action at a particular state. If the determinizer decides to restrict to the self-looping action, the reachability property may be violated in the determinized strategy. However, this problem can be addressed when synthesizing the strategy by ensuring that every action makes progress towards the target.

3 Tool

`dtControl 2.0` is an easy-to-use open-source tool for representing memoryless symbolic controllers as more compact and more interpretable DTs, while retaining safety guarantees of the original controllers. Our website `dtcontrol.model.in.tum.de` offers hyperlinks to the easy-to-install `pip` package[3], the documentation and the source code. Additionally, the artifact that has passed the TACAS 21 artifact evaluation is available here [6].

The schema in Figure 2 illustrates the workflow of using `dtControl`, highlighting new features in red. Considering `dtControl` as a black box, it shows that given a controller, it returns a DT representing the controller and also offers the possibility to simulate a run of the system under the control of the DT, visualizing

[3] `pip` is a standard package-management system used to install and manage software packages written in Python.

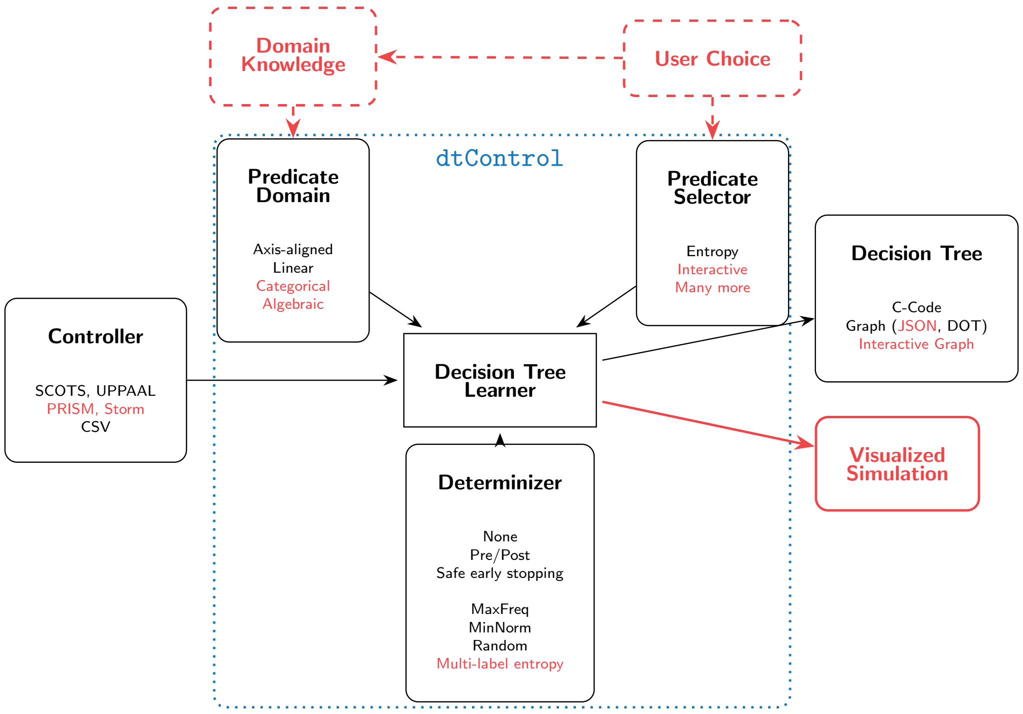
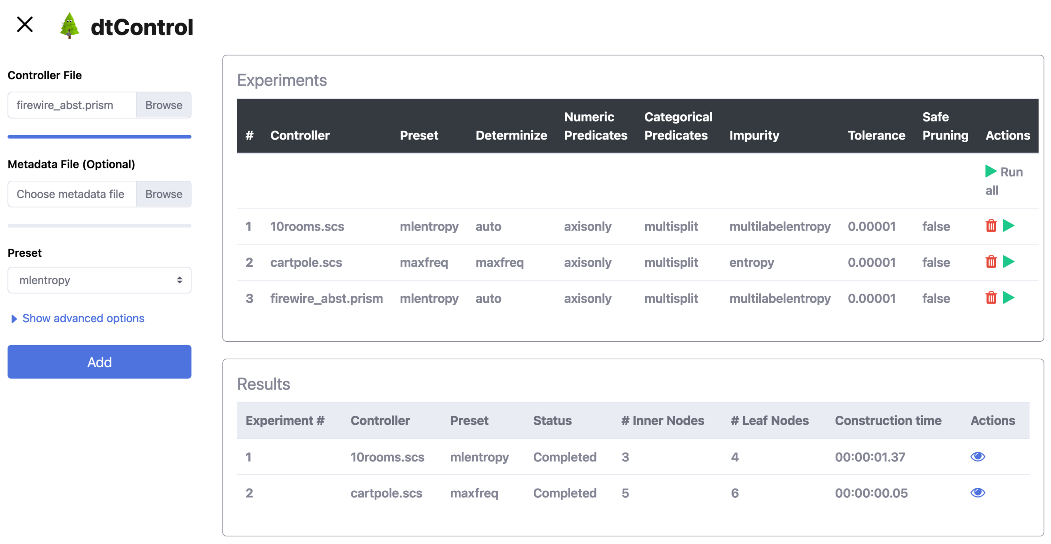


Fig. 2: An overview of the components of `dtControl 2.0`, thereby showing software architecture and workflow. Contributions of this paper are highlighted in red.

the decisions made. The controller can be input in various formats, including the newly supported strategy representations of the well-known probabilistic model checkers `PRISM` [32] and `STORM` [17]. The DT is output in several machine readable formats, and as C-code that can be directly used for executing the controller on embedded devices. Note that this C-code consists only of nested if-else-statements. The new graphical user interface also offers the possibility to inspect the graph in an interactive web user interface, which even allows to edit the DT. This means that parts of the DT can be retrained with a different set of hyper-parameters and directly replaced. This way, one can for example first train a determinized DT and then retrain important parts of it to be more permissive and hence more performant for a secondary criterion. Figure 3 shows a screenshot of the newly integrated graphical user interface.

Looking at the inner workings of `dtControl`, we see the three important hyper-parameters that were already introduced in Section 2: predicate domain, predicate selector, and determinizer. For each of these, `dtControl` offers various choices, some of which were newly added for version 2.0. Most prominently, the user now has the possibility to directly influence both the predicate domain and the predicate selector, by providing domain knowledge and thus also additional predicates, or by directly using the interactive predicate selection. More details on the predicate domain and how domain knowledge is specified can be found in Section 4. The different ways to select predicates, especially the new interactive mode, are the topic of Section 5. Our new insights into determinization are



Controller File

firewire_abst.prism Browse

Metadata File (Optional)

Choose metadata file Browse

Preset

mlentropy

Show advanced options

Add

Experiments

#	Controller	Preset	Determinize	Numeric Predicates	Categorical Predicates	Impurity	Tolerance	Safe Pruning	Actions
									▶ Run all
1	10rooms.scs	mlentropy	auto	axisonly	multisplit	multilabelentropy	0.00001	false	🗑 ▶
2	cartpole.scs	maxfreq	maxfreq	axisonly	multisplit	entropy	0.00001	false	🗑 ▶
3	firewire_abst.prism	mlentropy	auto	axisonly	multisplit	multilabelentropy	0.00001	false	🗑 ▶

Results

Experiment #	Controller	Preset	Status	# Inner Nodes	# Leaf Nodes	Construction time	Actions
1	10rooms.scs	mlentropy	Completed	3	4	00:00:01.37	👁
2	cartpole.scs	maxfreq	Completed	5	6	00:00:00.05	👁

Fig. 3: Screenshot of the new web-based graphical user interface. It offers a sidebar for easy selection of the controller file and hyper-parameters, an experiments table where benchmarks can be queued, and a results table in which some statistics of the run are provided. Moreover, users can click on the 'eye' icon in the results table to inspect the built decision tree.

described in Section 6. To support the user in finding a good set of hyper-parameters, `dtControl` also offers extensive benchmarking functionality, allowing to specify multiple variants and reporting several statistics.

Technical notes. `dtControl 2.0` is written in Python 3 following an architecture closely resembling the schema in Figure 2. The modularity, along with our technical documentation, allows users to easily extend the tool. For example, supporting another input format is only a matter of adding a parser.

`dtControl 2.0` works with Python version 3.7.9 or higher. The core of the tool which runs the learning algorithms requires `numpy` [23], `pandas` [36] and `scikit-learn` [41] and optionally the library for the heuristic `OC1` [39]. The algebraic predicates rely on `SymPy` [37] and `SciPy` [48]. The web user interface is powered by Flask [1] and D3.js [9].

4 Predicate domain

The domain of the predicates that we allow in the inner nodes of the DT is of key importance. As we saw in the motivating Example 1, allowing for more expressive predicates can dramatically reduce the size of the DT.

We assume that our state space is structured, i.e. it is a Cartesian product of the domain of the variables ($S = S_1 \times \ldots \times S_n$). We use s_i to refer to the i-th state-variable of a state $s \in S$. In Example 2, the three state-variables are the velocity of our car, the velocity of the front car, and the distance.

We first give an overview of the predicate domains dtControl 2.0 supports, before discussing the details of the new ones.

Axis-aligned predicates [38] have the form $s_i \leq c$, where c is a rational constant. This is the easiest form of predicates, and they have the advantage that there are only finitely many, as the domain of every state-variable is bounded. However, they are also least expressive.

Linear predicates (also known as oblique [39]) have the form $\sum_i s_i \cdot a_i \leq c$, where a_i are rational coefficients and c is a rational constant. They have the advantage that they are able to combine several state-variables which can lead to saving linearly many splits, cf. [29, Fig. 5.2]. The disadvantage of these predicates is that there are infinitely many choices of coefficients, which is why heuristics were introduced to determine a good set of predicates to try out [39, 4]. However, heuristically determined coefficients and combinations of variables can impede explainability.

Algebraic predicates have the form $f(s) \leq c$, where f is any mathematical function over the state-variables and c is a rational constant. It can use elementary functions such as exponentiation, log, or even trigonometric functions. Example 1 illustrated how this can reduce the size and improve explainability. More discussion of these predicates follows in Section 4.2.

Categorical predicates are special predicates for categorical (enumeration-type) state-variables such as colour or protocol state, and they are discussed in Section 4.1.

4.1 Categorical predicates

Categorical state-variables do not have a numeric domain, but instead are unordered and qualitative. They commonly occur in the models coming from the tools PRISM and STORM.

Example 3. Let one state-variable be 'colour' with the domain $\{\text{red}, \text{blue}, \text{green}\}$. A simple approach is to assign numbers to every value, e.g. $\text{red} = 0, \text{blue} = 1, \text{green} = 2$, and treat this variable as numeric. However, a resulting predicate such as $\text{colour} \leq 2$ is hardly explainable and additionally depends on the assignment of numbers. For example, it would not be possible to single out $\text{colour} \in \{\text{red}, \text{green}\}$ using a single predicate, given the aforementioned numeric assignment. Using linear predicates, for example adding half of the colour to some other state-variable, is even more confusing and dependent on the numeric assignment. △

Instead of treating the categorical variables using their numeric encodings, dtControl 2.0 supports specialized algorithms from literature, see e.g. [43, 44]. They work by labelling an inner node with a categorical variable and performing a (possibly non-binary) split according to the value of the categorical variable. The node can have at most one child for every possible value of the categorical variable, but it can also group together similarly behaving values, see Figure 4 for an example. For the grouping, dtControl 2.0 uses the greedy algorithm from [44, Chapter 7] called attribute-value grouping. It proceeds by first considering to have a branch for every single possible value of the categorical variable, and then merging branches as long as it improves the predicate; see [5, Appendix C] for the full pseudocode of the algorithm.

In our experiments we found that the grouping algorithm sometimes did not merge branches in cases where it would actually have made the DT smaller or more explainable. This is because the resulting impurity, the goodness of a predicate, could be marginally worse due to floating-point inaccuracies. Thus, we introduce *tolerance*, a bias parameter in favour of larger value groups. When checking whether to merge branches, we do not require the impurity to improve, but we allow it to become worse up to our tolerance. Setting tolerance to 0 corresponds exactly to the algorithm from [44], while setting tolerance to ∞ results in merging branches until only two remain, thus producing binary predicates.

To allow `dtControl 2.0` to use categorical predicates, the user has to provide a metadata file, which tells the tool which variables are categorical and which are numeric; see [5, Appendix B.1] for an example.

4.2 Algebraic predicates

It is impossible to try out every mathematical expression over the state-variables, and it would also not necessarily result in an explainable DT. Instead, we allow the user to enter *domain knowledge* to suggest templates of predicates that `dtControl 2.0` should try. See [5, Appendix B.2] for a discussion of the format in which domain knowledge can be entered.

Providing the basic equations that govern the model behaviour can already help in finding a good predicate, and is easy to do for a domain expert. Additionally, `dtControl 2.0` offers several possibilities to further exploit the provided domain knowledge:

Firstly, the given predicates need not be exact, but may contain coefficients. These coefficients can be both completely arbitrary or may come from a finite set suggested by the user. For coefficients with finite domain, `dtControl 2.0` tries all possibilities; for arbitrary coefficients, it uses curve fitting to find a good

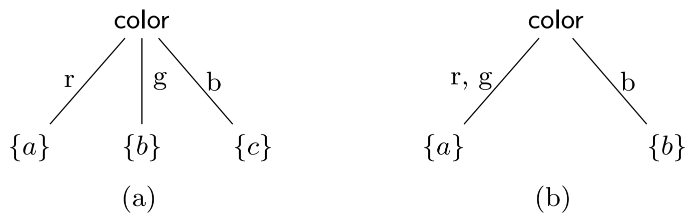


Fig. 4: Two examples of a categorical split. On the left, all possible values of the state-variable `colour` lead to a different child in a non-binary split. On the right, red and green lead to the same child, which is a result of grouping similar values together.

value. For example, the user can specify a predicate such as $d + (v_o - v_f) \cdot c_0 > c_1$ with c_0 being an arbitrary rational number and $c_1 \in \{0, 5, 10\}$.

Secondly, the interactive predicate selection (see Section 5) allows the user to try out various predicates at once and observe their respective impurity in the current node. The user can then choose among them as well as iteratively suggest further predicates, inspired by those where the most promising results were observed.

Thirdly, the decisions given by a DT can be visualized in the simulator, possibly leading to better understanding the controller. Upon gaining any further insight, the user can directly edit any subtree of the result, possibly utilizing the interactive predicate selection again.

5 Predicate selection

The tool offers a range of options to affect the selection of the most appropriate predicate from a given domain.

Impurity measures: As mentioned in Section 2, the predicate selection is typically based on the lowest *impurity* induced. The most commonly used impurity measure (and the only one the first version of `dtControl` supported) is Shannon's entropy [46]. In `dtControl 2.0`, a number of other impurity measures from the literature [43, 13, 25, 39, 3] are available. However, our results indicate that entropy typically performs the best, and therefore it is used as the default option unless the user specifies otherwise. Due to lack of space, we delegate the details and experimental comparison between the impurity measures to [5, Appendix D].

Priorities: `dtControl 2.0` also has the new functionality to assign *priorities* to the predicate generating algorithms. Priorities are rational numbers between 0 and 1. The impurity of every predicate is divided by the priority of the algorithm that generated it. For example, a user can use axis-aligned splits with priority 1 and a linear heuristic with priority $1/2$. Then the more complicated linear predicate is only chosen if it is at least twice as good (in terms of impurity) as the easier-to-understand axis-aligned split. A predicate with priority 0 is only considered after all predicates with non-zero priority have failed to split the data. This allows the user to give just a few predicates from domain knowledge, which are then strictly preferred to the automatically generated ones, but which need not suffice to construct a complete DT for the controller.

Interactive predicate selection: `dtControl 2.0` offers the user the possibility to manually select the predicate in every split. This way, the user can prefer predicates that are explainable over those that optimize the impurity.

The screenshot of the interactive interface in [5, Appendix F] shows the information that `dtControl 2.0` provides. The user is given some statistics and metadata, e.g. minimum, maximum and step size of the state-variables in the current node, a few automatically generated predicates for reference and all predicates generated from domain knowledge. The user can specify new predicates and is immediately informed about their impurity. Upon selecting a predicate, the split is performed and the user continues in the next node.

The user can also first construct a DT using some automatic algorithm and then restart the construction from an arbitrary node using the interactive predicate selection to handcraft an optimized representation, or at any point decide that the rest of the DT should be constructed automatically.

6 New insights about determinization

In our context, *determinization* denotes a procedure that, for some or all states, picks a subset of the allowed actions. Formally, a determinization function δ transforms a controller C into a "more determinized" C', such that for all states $s \in C$ we have $\emptyset \subsetneq C'(s) \subseteq C(s)$. This reduces the permissiveness, but often also reduces the size. Note that, for safety controllers, this always preserves the original guarantees of the controller. For other (non-safety) controllers, see Remark 1.

`dtControl 2.0` supports three different general approaches to determinizing a controller: pre-processing, post-processing and safe early stopping. Pre-processing commits to a single determinization before constructing the DT. Post-processing prunes the DT after its construction, e.g. safe pruning in [7]. The basic idea of safe early stopping is already described in Section 2: if all states agree on at least one action, then instead of continuing to split the controller, stop early and return a leaf node with that common action. Alternatively, to preserve more permissiveness, one can return not only a single common action, but all common actions; formally, return the maximum set B such that for all states s in the node $B \subseteq C(s)$.

The results of [4] show that both pre-processing and post-processing are outperformed by an on-the-fly approach based on safe early stopping. This is because pre-processing discards a lot of information that could have been useful in the DT construction and post-processing can only affect the bottom-most nodes of the resulting DT, but usually not those close to the root.

We now give a new view on safe early stopping approaches for determinizing a controller that allows us to generalize the techniques of [4], reducing the size of the resulting DTs even more.

Example 4. Consider the following controller: $C(s_1) = \{a, b, c\}$, $C(s_2) = \{a, b, d\}$, $C(s_3) = \{x, y\}$. All three states map to different sets of actions, and thus an impurity measure like entropy penalizes grouping s_1 and s_2 the same as grouping s_1 and s_3. However, if determinization is allowed, grouping s_1 and s_2 need not be penalized at all, as these states agree on some actions, namely a and b. Grouping s_1 and s_2 into the same child node thus allows the algorithm to stop early at that point and return a leaf node with $\{a, b\}$, in contrast to grouping s_1 and s_3. $\triangle$

Knowing that we want to determinize by safe early stopping affects the predicate selection process. Intuitively, sets of states are more homogeneous the more actions they share. We want to take this into account when calculating the impurity of predicates. One way to do this would be to calculate the impurity of all possible determinization functions and pick the best one. This, however, is infeasible, hence we propose the heuristic of *multi-label impurity measures*. These impurity measures do not only consider the full set of allowed actions in their calculation, but instead they depend on the individual actions occurring in the set. This allows the DT construction to pick better predicates, namely those whose resulting children are more likely to be determinizable. In [5, Appendix E] we formally derive the multi-label variants of entropy and Gini-index.

To conclude this section, we point out the key difference between the new approach of multi-label impurity measures and the previous idea that was introduced in [4]. The approach from [4] does not evaluate the impurity of all possible determinization functions, but rather picks a smart one – that of maximum frequency (MaxFreq) – and evaluates according to that. MaxFreq determinizes in the following way: for every state, it selects from the allowed actions that action occurring most frequently throughout the whole controller. This way, many states share common actions. This is already better than pre-processing, as it does not determinize the controller a priori, but rather considers a different determinization function at every node. However, in every node we calculate the impurity for several different predicates, and the optimal choice of determinization function depends on the predicate. Thus, choosing a single determinization function for a whole node is still too coarse, as it is fixed independent of the considered predicate. We illustrate the arising problem in the following Example 5.

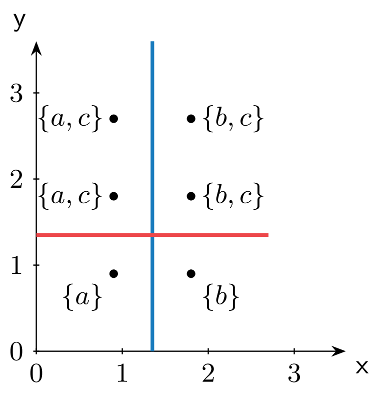


Fig. 5: A simple example of a dataset that is split suboptimally by the MaxFreq approach from [4], but optimally by the new multi-label entropy approach.

Example 5. Figure 5 shows a simple controller with a two-dimensional state space. Every point is labeled with its set of allowed actions.

As c is the most frequent action, MaxFreq determinizes the states $(1, 2)$, $(1, 3)$, $(2, 2)$ and $(2, 3)$ to action c. Hence the red split (predicate $y < 1.5$) is considered optimal, as it groups together all four states that map to c. The blue split (predicate $x < 1.5$) is considered suboptimal, as then the data still looks very heterogeneous. So, using MaxFreq, we need two splits for this controller; one to split of all the c's and one to split the two remaining states.

However, it is better to first choose a predicate and then determine a fitting determinization function. When calculating the impurity of the blue split, we can choose to determinize all states with $x = 1$ to $\{a\}$ and all states with $x = 2$ to $\{b\}$. Thus, in both resulting sub-controllers the impurity is 0 as all states agree on at least one action. This way, one split suffices to get a complete DT. Multi-label impurity measures notice when labels are shared between many (or all) states in a sub-controller, and thus they allow to prefer the optimal blue split. $\triangle$

7 Experiments

Experimental setup. We compare three approaches: BDDs, the first version of `dtControl` from [4] and `dtControl 2.0`. For BDDs[4] the variable ordering is important, so we report the smallest of 20 BDDs that we constructed by starting with a random initial variable ordering and reordering until convergence. To determinize BDDs, we used the pre-processing approach, 10 times with the minimum norm and 10 times with MaxFreq. For the previous version of `dtControl`, we picked the smaller of either a DT with only axis-aligned predicates or a DT with linear predicates using the logistic regression heuristic that was typically best in [4]. Determinization uses safe early stopping with the MaxFreq approach. For `dtControl 2.0`, we use the multi-label entropy based determinization and utilize the categorical predicates for the case studies from probabilistic model checking. We ran all experiments on a server with operating system Ubuntu 19.10, a 2.2GHz Intel(R) Xeon(R) CPU E5-2630 v4 and 250 GB RAM.

Comparing determinization techniques on cyber-physical systems. Table 1 shows the sizes of determinized BDDs and DTs on the permissive controllers of the tools `SCOTS` and `Uppaal Stratego` that were already used in [4]. We see that the new determinization approach is strictly better than the previous one, with only two DTs being of equal size, as the result of the previous method was already optimal. With the exception of the case studies helicopter and truck_trailer where BDDs are comparable or slightly better, both approaches using DTs are orders of magnitude smaller than BDDs or an explicit representation of the state-action mapping.

Case studies from probabilistic model checking. For Table 2, we used case studies from the quantitative verification benchmark set [24], which includes models from the `PRISM` benchmark suite [33]. Note that these case studies contain unordered enumeration-type state-variables for which we utilize the new categorical predicates. To get the controllers, we solved the case study with `STORM` and exported the resulting controller. This export already eliminates unreachable states. The

[4] Our implementation of BDDs is based on the `dd` python library `https://github.com/tulip-control/dd`.

Table 1: Controller sizes of different determinized representations of the controllers from `SCOTS` and `Uppaal Stratego`. "States" is the number of states in the controller, "BDD" the number of nodes of the smallest BDD from 20 tries, `dtControl 1.0` [4] the smallest DT the previous version of `dtControl` could generate and `dtControl 2.0` the smallest DT the new version can construct. "TO" denotes a failure to produce a result in 3 hours. The smallest numbers in each row are highlighted.

Case study	States	BDD	`dtControl 1.0`	`dtControl 2.0`
cartpole	271	127	11	**7**
10rooms	26,244	128	**7**	**7**
helicopter	280,539	870	221	**123**
cruise-latest	295,615	1,448	**3**	**3**
dcdc	593,089	381	9	**5**
truck_trailer	1,386,211	**18,186**	42,561	31,499
traffic_30m	16,639,662	TO	127	**97**

previous version of `dtControl` was not able to handle these case studies, so we only compare `dtControl 2.0` to BDDs.

Table 2 shows that also for case studies from probabilistic model checking, DTs are a good way of representing controllers. The DT is the smallest representation on 13 out of 19 case studies, often reducing the size by an order of magnitude compared to BDDs or the explicit representation. On 3 case studies, BDDs are smallest, and on 2 case studies, both the DT and the BDD fail to reduce the size compared to the explicit representation. This happens if there are many different actions and thus states cannot be grouped together. A worst case example of this is a model where every state has a different action; then, a DT would have as many leaf nodes as there are states, and hence twice as many nodes in total.

Remark 2. Note that the controllers exported by `STORM` are deterministic, so no determinization approach can be utilized in the DT construction. We conjecture that if a permissive strategy was exported, `dtControl 2.0` would benefit from the additional information and be able to reduce the controller size further as for the cyber-physical systems.

8 Conclusion

We have presented a radically new version of the tool `dtControl` for representing controllers by decision trees. The tool now features a graphical user interface, allowing both experts and non-experts to conveniently interact with the decision tree learning process as well as the resulting tree. There is now a range of possibilities on how the user can provide additional information. The algebraic predicates provide the means to capture the (often non-linear) relationships from the domain knowledge. The categorical predicates together with the interface to probabilistic model checkers allow for efficient representation of strategies for Markov decision processes, too. Finally, the more efficient determinization yields

Table 2: Controller sizes of different representations of controllers from the quantitative verification benchmark set [24], i.e. from the tools `STORM` and `PRISM`. “States” is the number of states in the controller, “BDD” the number of nodes of the smallest BDD of 20 tries and `dtControl 2.0` the smallest DT we could construct. The smallest numbers in each row are highlighted.

Case study	States	BDD	`dtControl 2.0`
triangle-tireworld.9	48	51	**23**
pacman.5	232	330	**33**
rectangle-tireworld.11	**241**	498	373
philosophers-mdp.3	344	295	**181**
firewire_abst.3.rounds	610	61	**25**
rabin.3	704	303	**27**
ij.10	1,013	**436**	753
zeroconf.1000.4.true.correct_max	1,068	386	**63**
blocksworld.5	1,124	3,985	**855**
cdrive.10	**1,921**	5,134	2,401
consensus.2.disagree	2,064	138	**67**
beb.3-4.LineSeized	4,173	913	**59**
csma.2-4.some_before	7,472	1,059	**103**
eajs.2.100.5.ExpUtil	12,627	1,315	**153**
elevators.a-11-9	14,742	**6,750**	9,883
exploding-blocksworld.5	76,741	34,447	**1,777**
echoring.MaxOffline1	104,892	43,165	**1,543**
wlan_dl.0.80.deadline	189,641	5,738	**2,563**
pnueli-zuck.5	303,427	**50,128**	150,341

very small (possibly non-performant) controllers, which are particularly useful for debugging the model.

We see at least two major promising future directions. Firstly, synthesis of predicates could be made more automatic using mathematical reasoning on the domain knowledge, such as substituting expressions with a certain unit of measurement into other domain equations in the places with the same unit of measurement, e.g. to plug difference of two velocities into an equation for velocity. Secondly, one could transform the controllers into possibly entirely different controllers (not just less permissive) so that they still preserve optimality (or yield ε-optimality) but are smaller or simpler. Here, a closer interaction loop with the model checkers might lead to efficient heuristics.

References

1. Flask web development: developing web applications with python. `https://pypi.org/project/Flask/`, accessed: 14.10.2020
2. Adadi, A., Berrada, M.: Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). IEEE Access **6**, 52138–52160 (2018)
3. Ashok, P., Brázdil, T., Chatterjee, K., Křetínský, J., Lampert, C.H., Toman, V.: Strategy representation by decision trees with linear classifiers. In: QEST. Lecture Notes in Computer Science, vol. 11785, pp. 109–128. Springer (2019)
4. Ashok, P., Jackermeier, M., Jagtap, P., Křetínský, J., Weininger, M., Zamani, M.: dtcontrol: decision tree learning algorithms for controller representation. In: HSCC. pp. 17:1–17:7. ACM (2020)
5. Ashok, P., Jackermeier, M., Křetínský, J., Weinhuber, C., Weininger, M., Yadav, M.: dtControl 2.0: Explainable strategy representation via decision tree learning steered by experts. CoRR **abs/2101.07202** (2021)
6. Ashok, P., Jackermeier, M., Křetínský, J., Weinhuber, C., Weininger, M., Yadav, M.: dtControl 2.0: Explainable strategy representation via decision tree learning steered by experts (TACAS 21 artifact) (Jan 2021). https://doi.org/10.5281/zenodo.4437169
7. Ashok, P., Křetínský, J., Larsen, K.G., Coënt, A.L., Taankvist, J.H., Weininger, M.: SOS: safe, optimal and small strategies for hybrid Markov decision processes. In: QEST. Lecture Notes in Computer Science, vol. 11785, pp. 147–164. Springer (2019)
8. Bahar, R.I., Frohm, E.A., Gaona, C.M., Hachtel, G.D., Macii, E., Pardo, A., Somenzi, F.: Algebraic decision diagrams and their applications. Formal Methods Syst. Des. **10**(2/3), 171–206 (1997)
9. Bostock, M., Ogievetsky, V., Heer, J.: D^3 data-driven documents. IEEE transactions on visualization and computer graphics **17**(12), 2301–2309 (2011)
10. Boutilier, C., Dearden, R., Goldszmidt, M.: Exploiting structure in policy construction. In: IJCAI. pp. 1104–1113. Morgan Kaufmann (1995)
11. Brázdil, T., Chatterjee, K., Chmelik, M., Fellner, A., Křetínský, J.: Counterexample explanation by learning small strategies in Markov decision processes. In: CAV (1). Lecture Notes in Computer Science, vol. 9206, pp. 158–177. Springer (2015)
12. Brázdil, T., Chatterjee, K., Křetínský, J., Toman, V.: Strategy representation by decision trees in reactive synthesis. In: TACAS (1). Lecture Notes in Computer Science, vol. 10805, pp. 385–407. Springer (2018)
13. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth (1984)
14. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers **35**(8), 677–691 (1986)
15. Cimatti, A., Roveri, M., Traverso, P.: Automatic obdd-based generation of universal plans in non-deterministic domains. In: AAAI/IAAI. pp. 875–881. AAAI Press / The MIT Press (1998)
16. David, A., Jensen, P.G., Larsen, K.G., Mikucionis, M., Taankvist, J.H.: Uppaal stratego. In: TACAS. Lecture Notes in Computer Science, vol. 9035, pp. 206–211. Springer (2015)
17. Dehnert, C., Junges, S., Katoen, J., Volk, M.: A storm is coming: A modern probabilistic model checker. In: CAV (2). Lecture Notes in Computer Science, vol. 10427, pp. 592–600. Springer (2017)
18. Della Penna, G., Intrigila, B., Lauri, N., Magazzeni, D.: Fast and compact encoding of numerical controllers using obdds. In: Cetto, J.A., Ferrier, J.L., Filipe, J. (eds.)

Informatics in Control, Automation and Robotics: Selcted Papers from the International Conference on Informatics in Control, Automation and Robotics 2008, pp. 75–87. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)

19. Frehse, G., Guernic, C.L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: Spaceex: Scalable verification of hybrid systems. In: CAV. Lecture Notes in Computer Science, vol. 6806, pp. 379–395. Springer (2011)
20. Fujita, M., McGeer, P.C., Yang, J.C.: Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. Formal Methods Syst. Des. **10**(2/3), 149–169 (1997)
21. Garg, P., Neider, D., Madhusudan, P., Roth, D.: Learning invariants using decision trees and implication counterexamples. In: POPL. pp. 499–512. ACM (2016)
22. Girard, A.: Low-complexity quantized switching controllers using approximate bisimulation. CoRR **abs/1209.4576** (2012)
23. Harris, C.R., Millman, K.J., van der Walt, S., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del Río, J.F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E.: Array programming with numpy. CoRR **abs/2006.10256** (2020)
24. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The quantitative verification benchmark set. In: TACAS (1). Lecture Notes in Computer Science, vol. 11427, pp. 344–350. Springer (2019)
25. Heath, D.G., Kasif, S., Salzberg, S.: Induction of oblique decision trees. In: Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993. pp. 1002–1007 (1993)
26. Hoey, J., St-Aubin, R., Hu, A.J., Boutilier, C.: SPUDD: stochastic planning using decision diagrams. In: UAI. pp. 279–288. Morgan Kaufmann (1999)
27. Hyafil, L., Rivest, R.L.: Constructing optimal binary decision trees is NP-complete. Inf. Process. Lett. **5**(1), 15–17 (1976)
28. Ittner, A., Schlosser, M.: Non-linear decision trees - NDT. In: ICML. pp. 252–257. Morgan Kaufmann (1996)
29. Jackermeier, M.: dtControl: Decision Tree Learning for Explainable Controller Representation. Bachelor's thesis, Technische Universität München (2020)
30. Jr., M.M., Davitian, A., Tabuada, P.: PESSOA: A tool for embedded controller synthesis. In: CAV. Lecture Notes in Computer Science, vol. 6174, pp. 566–569. Springer (2010)
31. Kiesbye, J.: Private Communication (2020)
32. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. Lecture Notes in Computer Science, vol. 6806, pp. 585–591. Springer (2011)
33. Kwiatkowska, M.Z., Norman, G., Parker, D.: The PRISM benchmark suite. In: QEST. pp. 203–204. IEEE Computer Society (2012)
34. Larsen, K.G., Mikucionis, M., Taankvist, J.H.: Safe and optimal adaptive cruise control. In: Correct System Design. Lecture Notes in Computer Science, vol. 9360, pp. 260–277. Springer (2015)
35. Luttenberger, M., Meyer, P.J., Sickert, S.: Practical synthesis of reactive systems from LTL specifications via parity games. Acta Informatica **57**(1-2), 3–36 (2020)
36. Wes McKinney: Data Structures for Statistical Computing in Python. In: Stéfan van der Walt, Jarrod Millman (eds.) Proceedings of the 9th Python in Science Conference. pp. 56 – 61 (2010). https://doi.org/10.25080/Majora-92bf1922-00a

37. Meurer, A., Smith, C.P., Paprocki, M., Certík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., Singh, S., Rathnayake, T., Vig, S., Granger, B.E., Muller, R.P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M.J., Terrel, A.R., Roucka, S., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., Scopatz, A.M.: Sympy: symbolic computing in python. PeerJ Comput. Sci. **3**, e103 (2017)
38. Mitchell, T.M.: Machine learning. McGraw Hill series in computer science, McGraw-Hill (1997)
39. Murthy, S.K., Kasif, S., Salzberg, S., Beigel, R.: OC1: A randomized induction of oblique decision trees. In: AAAI. pp. 322–327. AAAI Press / The MIT Press (1993)
40. Neider, D., Markgraf, O.: Learning-based synthesis of safety controllers. In: FMCAD. pp. 120–128. IEEE (2019)
41. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
42. Pyeatt, L.D., Howe, A.E., et al.: Decision tree function approximation in reinforcement learning. In: Proceedings of the third international symposium on adaptive systems: evolutionary computation and probabilistic graphical models. vol. 2, pp. 70–77. Cuba (2001)
43. Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986)
44. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
45. Rungger, M., Zamani, M.: SCOTS: A tool for the synthesis of symbolic controllers. In: HSCC. pp. 99–104. ACM (2016)
46. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**(4), 623–656 (1948)
47. St-Aubin, R., Hoey, J., Boutilier, C.: APRICODD: approximate policy construction using decision diagrams. In: NIPS. pp. 1089–1095. MIT Press (2000)
48. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, I., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., SciPy: Scipy 1.0-fundamental algorithms for scientific computing in python. CoRR **abs/1907.10121** (2019)
49. Zapreev, I.S., Verdier, C., Jr., M.M.: Optimal symbolic controllers determinization for BDD storage. In: ADHS 2018. IFAC-PapersOnLine, vol. 51, pp. 1–6. Elsevier (2018). https://doi.org/10.1016/j.ifacol.2018.08.001

H Guaranteed Trade-Offs in Dynamic Information Flow Tracking Games (CDC 2021)

This is an exact reprinting of the accepted version of the following paper which has been published as a **peer reviewed conference paper**.

> Maximilian Weininger, Kush Grover, Shruti Misra, and Jan Kretinsky. "Guaranteed Trade-Offs in Dynamic Information Flow Tracking Games" In: CDC. IEEE, 2021, pp. 3786-3793. doi: 10.1109/CDC45484.2021.9683447

Summary

We consider security risks in the form of advanced persistent threats (APTs) and their detection using dynamic information flow tracking (DIFT). We model the tracking and the detection as a stochastic game between the attacker and the defender. Compared to the state of the art, our approach applies to a wider set of scenarios with arbitrary (not only acyclic) information-flow structure. Moreover, multidimensional rewards allow us to formulate and answer questions related to trade-offs between resource efficiency of the tracking and efficacy of the detection. Finally, our algorithm provides results with probably approximately correct (PAC) guarantees, in contrast to previous (possibly arbitrarily imprecise) learning-based approaches.

For more details on this publication, we refer to Chapter 7 of the main body.

Contribution

Contribution of Maximilian Weininger	
Development and conceptual design of the research project	95%
Discussion and development of ideas	75%
Composition and revision of the manuscript	80%
Proofs	80%
Implementation	20%
Experimental evaluation	10%

CCC RightsLink®

Home | Help | Live Chat | Sign in | Create Account

IEEE Requesting permission to reuse content from an IEEE publication

Guaranteed Trade-Offs in Dynamic Information Flow Tracking Games

Conference Proceedings: 2021 60th IEEE Conference on Decision and Control (CDC)
Author: Maximilian Weininger
Publisher: IEEE
Date: 14 Dec. 2021

Copyright © 2021, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK CLOSE WINDOW

Guaranteed Trade-Offs in Dynamic Information Flow Tracking Games

Maximilian Weininger*, Kush Grover*, Shruti Misra and Jan Křetínský

***Abstract*— We consider security risks in the form of advanced persistent threats (APTs) and their detection using dynamic information flow tracking (DIFT). We model the tracking and the detection as a stochastic game between the attacker and the defender. Compared to the state of the art, our approach applies to a wider set of scenarios with arbitrary (not only acyclic) information-flow structure. Moreover, multidimensional rewards allow us to formulate and answer questions related to trade-offs between resource efficiency of the tracking and efficacy of the detection. Finally, our algorithm provides results with probably approximately correct (PAC) guarantees, in contrast to previous (possibly arbitrarily imprecise) learning-based approaches.**

I. Introduction

In 2013, Target Corporation's network was breached, which resulted in 40 million credit and debit card numbers along with 70 million records of personal information being stolen. Attackers entered the network by compromising a third party vendor and collected sensitive information from Target's network over the course of two weeks, while remaining undetected [22]. Such targeted attacks in which the attacker gains illegitimate access to a system and remains undetected for a long period of time are known as *advanced persistent threats (APTs)*. APTs use prolonged and stealthy incursion techniques, which are customized to specific targets in order to gather confidential data and sabotage critical infrastructure. They are methodically designed to bypass conventional security mechanisms, which makes them difficult to detect. APTs have emerged as a security threat to many organizations and industries, including national defense, manufacturing, and the financial industry [7].

Although detecting APTs is difficult, their interaction with the system during an attack creates information flows such as data-flow and control-flow commands, which are recorded in the system log file. Consequently, the recorded information flows can be analysed, leading to a feasible method to detect the presence of APTs. *Dynamic Information Flow Tracking (DIFT)* is a widely used mechanism to detect APTs in this manner [23]. DIFT operates by "tagging" suspicious input/output data channels, tracing the propagation of the tagged information through the system and then performing security analysis on these flows to detect attacks. However, this introduces memory and performance costs on the system, especially when it involves tracking and analyzing a large number of innocuous flows [13]. Therefore, our aim is to optimally select which system processes to perform the security analysis on, for effective and resource-efficient detection.

This paper extends prior work conducted on modelling a cost effective DIFT-based APT detection mechanism [16]. In [16], the strategic interactions between APT and DIFT are modeled as a *game*. The game is played on the information flow graph (IFG). The position of the information on the IFG along with the actions of the attacker and defender yield a probability distribution on the transitions in the game. A key aspect of this model is that the probability of a successful attack by the APT and an effective defense by DIFT depend on the actions of both the defender and the attacker, making the game *concurrent*. Further, the transition probabilities are determined by the DIFT's rate of false negatives, which varies over the IFG nodes and is unknown in practical scenarios. Therefore, the game is of *incomplete information* with, moreover, asymmetric information structure since DIFT is not capable of distinguishing between malicious and benign information flows. Finally, as the game captures the trade-off between DIFT's resource efficiency and detection effectiveness, the game is *non-zero-sum*. Altogether, the interaction was modelled as a non-zero-sum concurrent stochastic game with unknown transition probabilities. For an acyclic game, the paper [16] proves the existence of Nash equilibria and provides a learning-based approach to find approximate Nash equilibria. This paper retains the characteristics of the game in [16], but extends the work in the following dimensions:

1) Instead of a non-zero-sum game with a single-dimensional reward function [16], we explicitly model the problem with a *multi-dimensional* (zero-sum) reward function. This more interpretable formulation allows us to ask and answer questions such as "What cost does the defender need to pay to achieve the maximum probability of trapping the attacker?", "What is the highest probability to trap the attacker by a defence cheaper than 10" or what the possible trade-offs are. We address this by computing the achievable vectors in a 4-dimensional-reward setting.
2) Our approach is applicable also to *games with cycles*. We observe how to represent the particular concurrent stochastic game of [16] as a turn-based stochastic game, for which we can handle cycles. The work of [6] gives an algorithm to compute the achievable

*These authors contributed equally to this work.

This research was funded in part by the German Research Foundation (DFG) projects 383882557 *Statistical Unbounded Verification (SUV)*, 427755713 *Group-By Objectives in Probabilistic Verification (GOPro)*, and the research training group GRK 2428 *Continuous Verification of Cyber-Physical Systems (ConVeY)*.

K. Grover, J. Křetínský and M. Weininger work at the Department of Informatics, Technical University of Munich, 80333 Munich, Germany. E-mail: `firstname.lastname@tum.de`

S. Misra works at the Department of Electrical and Computer Engineering, University of Washington, Seattle, Washington, United States of America. E-mail: `shrm145@uw.edu`

vectors for turn-based stochastic games with multiple objectives.

3) In contrast to the previous learning-based solution, we provide a *guarantee* on the result in the form of a probably approximately correct (PAC) *best-/worst-case analysis*. Technically, we learn confidence intervals on the transition probabilities. The choice of probabilities within these intervals is assigned to one of the players. If the attacker chooses the probabilities, we obtain the worst-case analysis; if the defender can choose the probabilities, we obtain the best-case analysis. To compute them, we can utilize our solution [24] to bounded-parameter (turn-based) stochastic games.

The paper is organized as follows: In Section II we recall the definitions of the model (concurrent dynamic information flow tracking stochastic game) and of the multi-dimensional objective function. Section III gives the formal problem statement and relation to the one in [16]. In Section IV we describe the algorithm to solve the problem step by step. Section V discusses our implementation and experimental results on several real-world and randomly generated case studies.

A. Related work

The application of stochastic games to model strategic interactions has been well studied in the security context [26], [15]. A dynamic multi-stage Bayesian game framework for APT detection has been provided in [12]. References [20] and [18] provide game-theoretic models for resource efficient detection of APTs, where the trap locations are fixed. However, these game models are not stochastic as they assume that the security analysis performed by DIFT is accurate and do not consider the rate of false-positives and false-negatives generated by the system. A stochastic model of the APT versus DIFT game is proposed in [19], where the notion of conditional branching is considered. Note that in [16]-[20] the game models are concurrent, such that the defender and attacker simultaneously select actions. Moreover, all three game models are non-zero-sum, with a single dimensional reward function, whereas the model proposed in this paper is a turn-based zero-sum game with a multi-dimensional reward function. Concurrent games are generally significantly harder to handle than turn-based ones, e.g. [5], but our special case can be turned into an equivalent turn-based game. For further examples of PAC model checking, see [2], [9], [25].

II. Preliminaries

In this section, we recall the basics of both concurrent and turn-based stochastic games, as well as the specific structure of the Dynamic information flow tracking stochastic game (DIFT-SG). Further, we describe the multi-dimensional objective function.

First, we establish standard notation. $\mathbb{N}$ and $\mathbb{Q}$ are the set of all natural respectively rational numbers. For a set X, X^* denotes the set of all finite sequences consisting of elements of S. $|X|$ denotes the cardinality of the set. A *probability distribution* over a countable set X is a mapping $d : X \to [0,1]$ such that $\sum_{x\in X} d(x) = 1$. $\mathcal{D}(X)$ denotes the set of all probability distributions on X. Given a vector $x \in \mathbb{Q}^n$, we refer to its i-th component as x_i, where $1 \leq i \leq n$.

A. Stochastic games

We briefly summarize the definition of DIFT-SG from [16]. It is a game played between two players which are called Attacker $\mathcal{A}$ and Defender $\mathcal{D}$. The underlying state space is the information flow graph (IFG) of the system, see [16, Section III.A], with the addition of special states to indicate that the Attacker is trapped τ, has dropped out and aborted the attack ϕ or has reached the destination[1] and was successful d. A play of the game moves through the state space. In every state, the Attacker can follow an edge of the IFG to get to a successor state and thereby closer to the destination d. However, the Defender can decide to trap the flow. Then with some probability the trapped-state τ is reached, and with the remaining probability a false negative occurs and the play moves according to Attacker's choice. A DIFT-SG is an instance of a concurrent SG. See Figure 1 for an example and [16, Section III-IV] for more details.

Definition 1 (Concurrent SG): A concurrent stochastic game, e.g. [8], is a tuple $(\mathsf{S}, \mathsf{A}, \mathsf{A}_\mathcal{A}, \mathsf{A}_\mathcal{D}, \delta)$, where S is a finite set of states and A is a finite set of actions. $\mathsf{A}_\mathcal{A}$ and $\mathsf{A}_\mathcal{D}$ are functions $\mathsf{S} \to 2^{\mathsf{A}} \setminus \emptyset$ mapping every state to a non-empty set of available actions. $\delta : \mathsf{S} \times \mathsf{A} \times \mathsf{A} \to \mathcal{D}(\mathsf{S})$ is the transition function which given a state and two available actions[2] returns a probability distribution over successor states.

Definition 2 (DIFT-SG): In a DIFT-SG on an IFG with nodes $s_0, \ldots, s_N$ (where $s_N = d$) the state space is $\mathsf{S} = \{s_0, \ldots, s_{N-1}, d, \phi, \tau\}$. For every state $s \in \{s_0, \ldots, s_{N-1}\}$, we have $\mathsf{A}_\mathcal{A}(s) \subseteq \mathsf{S} \setminus \{\tau\}$, i.e. the Attacker chooses which state to try to reach next; the choice of successor is limited by the information flow graph. The action to drop out is available in every state. $\mathsf{A}_\mathcal{D}(s) = \{0, 1\}$, where 1 denotes trapping the flow and 0 not doing so. The transition function is defined as follows: If the Defender does not trap, we have $\delta(s, 0, s')(s') = 1$. If the Defender traps, we have $\delta(s, 1, s')(\tau) = p(s)$ and $\delta(s, 1, s')(s') = 1 - p(s)$, where $p(s)$ denotes the probability of trapping the Attacker in state s, i.e. 1 minus the false negative rate. In the special states d, ϕ, τ, both players only have a dummy action *loop* that leads to looping in the respective state. The set of all actions is $\mathsf{A} = (\mathsf{S} \setminus \{\tau\}) \cup \{0, 1\} \cup \{loop\}$.

Intuitively, a concurrent SG is played by starting in some initial state s and having both players choose an action from their set of available ones, i.e. $a_\mathcal{D} \in \mathsf{A}_\mathcal{D}(s)$ for Defender and $a_\mathcal{A} \in \mathsf{A}_\mathcal{A}(s)$ for Attacker. Then the game probabilistically transitions to a successor state s', where $\delta(s, a_\mathcal{D}, a_\mathcal{A})(s')$ gives the probability of moving to s'. Choosing actions and sampling the successor state is repeated infinitely often.

The game is concurrent, because in every state both players pick an action and the transitions depends on both their choices. A subclass of concurrent SGs are turn-based

[1]Note that the target is a state s_N of the IFG, but as in previous work we always denote it by d.

[2]Note that δ is not total. $\delta(s, a, b)$ is undefined if $a \notin \mathsf{A}_\mathcal{D}(s)$ or $b \notin \mathsf{A}_\mathcal{A}(s)$.

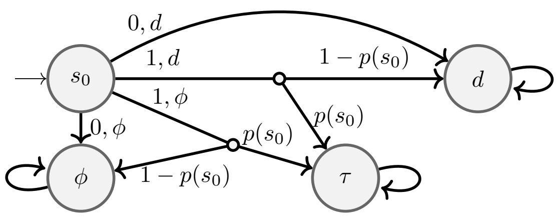


Fig. 1. An example of a small DIFT-SG.

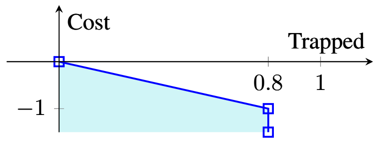


Fig. 2. Example of the set of achievable vectors.

SGs. There, in every state s we have either $|\mathsf{A}_{\mathcal{A}}(s)| = 1$ or $|\mathsf{A}_{\mathcal{D}}(s)| = 1$, i.e. one player has a trivial choice and cannot affect the next transition. Thus, we say it is the other players *turn*. This yields a partitioning[3] of the state space into states of Attacker $\mathsf{S}_{\mathcal{A}} = \{s \in \mathsf{S} \mid |\mathsf{A}_{\mathcal{D}}(s)| = 1\}$ and of the Defender $\mathsf{S}_{\mathcal{D}} = \{s \in \mathsf{S} \mid |\mathsf{A}_{\mathcal{A}}(s)| = 1, |\mathsf{A}_{\mathcal{D}}(s)| > 1\}$. Every turn-based SG can be made concurrent by adding a dummy action for the player to whom the state does not belong.

B. Semantics: Paths and strategies

The semantics of SGs is defined in the usual way by means of paths and strategies. Intuitively, an outcome of an SG depends on the path taken through the game graph. When both players decide what to do in each state, i.e. they fix a strategy, the outcome of the game only depends on the evaluation of the probabilistic transition function. Thus, given two strategies and an SG, we get a probability measure over paths in the game graph.

Formally, an infinite path in an SG is an infinite sequence $\varphi = s_0 a_0 b_0 s_1 a_1 b_1 \ldots$, such that for all $i \in \mathbb{N}$ we have $s_i \in \mathsf{S}$, $a_i \in \mathsf{A}_{\mathcal{D}}(s_i)$, $b_i \in \mathsf{A}_{\mathcal{A}}(s_i)$ and $\delta(s_i, a_i, b_i)(s_{i+1}) > 0$. A finite path is a finite prefix of an infinite path.

A strategy (also called policy, controller or scheduler) is a function $\pi : (\mathsf{S} \times \mathsf{A} \times \mathsf{A})^* \times \mathsf{S} \to \mathcal{D}(\mathsf{A})$, which given a finite path $\varphi = s_0 a_0 b_0 \ldots s_n$ yields a probability distribution on the actions to be taken next. We differentiate between Attacker strategies $\pi_{\mathcal{A}}(\varphi) \in \mathcal{D}(\mathsf{A}_{\mathcal{A}}(s_n))$ and Defender strategies $\pi_{\mathcal{D}}(\varphi) \in \mathcal{D}(\mathsf{A}_{\mathcal{D}}(s_n))$.

By fixing strategies of both players for an SG G with initial state s, we obtain a probability measure $\mathbb{P}^{\pi_{\mathcal{D}}, \pi_{\mathcal{A}}}_{\mathsf{G},s}$ over infinite paths in the usual way [3, Ch. 10].

C. Objectives

An objective assigns a value to every path, and thus provides a way to formalize the goal of the players, as they want to either maximize or minimize this value.

A *total reward objective* augments the game with rewards (also called payoffs) that are assigned to every triple of state and actions of both players. These rewards are accumulated along the path. Formally, a reward structure is a function $\mathsf{r} : \mathsf{S} \times \mathsf{A} \times \mathsf{A} \to \mathbb{Q}$. Then the total reward of an infinite path $\varphi = s_0 a_0 b_0 s_1 a_1 b_1 \ldots$ is given by $\mathsf{TR}(\varphi) = \sum_{j=0}^{\infty} r(s_j, a_j, b_j)$. A special case of total reward objectives are *reachability objectives*. They assign value 1 to all paths that reach a specified target state and 0 otherwise[4].

The *value of an SG* G with initial state s and a total reward objective r is the expected reward that the optimal strategy of the Defender can ensure against all strategies of the Attacker; formally $\mathsf{V}(\mathsf{G}, s, \mathsf{r}) = \sup_{\pi_{\mathcal{D}}} \inf_{\pi_{\mathcal{A}}} \mathbb{E}^{\pi_{\mathcal{D}}, \pi_{\mathcal{A}}}_{\mathsf{G},s}[\mathsf{TR}(\varphi)]$, where $\mathbb{E}^{\pi_{\mathcal{D}}, \pi_{\mathcal{A}}}_{\mathsf{G},s}$ is the expected value of the reward under the probability measure $\mathbb{P}^{\pi_{\mathcal{D}}, \pi_{\mathcal{A}}}_{\mathsf{G},s}$ over paths φ.

In this paper, we are not interested in only a single total reward objective. Instead, we are interested in four separate but interdependent objectives: three reachability objectives, namely the probability to reach the destination node, dropout state or trap state; and the cost that Defender accumulates for trying to trap the Attacker. Hence we need to deal with multiple objectives, see e.g. [6].

For a tuple of total reward objectives $\overline{\mathsf{TR}} = (\mathsf{TR}_1, \ldots, \mathsf{TR}_n)$, the set of achievable vectors is defined as

$$\mathcal{A}(\mathsf{G}, s, \overline{\mathsf{TR}}) = \\ \{x \in \mathbb{Q}^n \mid \exists \pi_{\mathcal{D}} \forall \pi_{\mathcal{A}} \forall 1 \leq i \leq n. \mathbb{E}^{\pi_{\mathcal{D}}, \pi_{\mathcal{A}}}_{\mathsf{G},s}[\mathsf{TR}_i(\varphi)] \geq x_i\}$$

Intuitively, the Defender tries to maximize the value of all total reward objectives while the attacker tries to minimize them. A vector x is achievable if for every dimension i Defender can guarantee that the reward is at least x_i. Note that the objectives might be partially conflicting, and thus two vectors $x_1, x_2 \in \mathcal{A}(\mathsf{G}, s, \overline{\mathsf{TR}})$ can be incomparable. Further, observe that Defender chooses a strategy first in both the definition of the value as well as of the achievable vectors. This is because we are interested in the rewards which the Defender can guarantee against every strategy of the attacker without knowing what to defend against. If we allowed Defender to react to Attacker's strategy, this could increase the set of achievable points; however, assuming to know what the Attacker plans is hardly a practical assumptions. In contrast, computing what Defender can guarantee against every strategy of the attacker gives a worst-case estimate that can definitely be achieved.

Example 1 (Achievable vectors): As the set of achievable vectors is a complex object, we illustrate it on the simple DIFT-SG from Figure 1. We only consider a two-dimensional objective, namely maximizing the probability to trap the Attacker and minimizing the cost. Let the trapping probability be $p(s_0) = 0.8$ and the cost of trapping -1.

The resulting set of achievable vectors is depicted in Figure 2. The corners of the set are $(0, 0)$ and $(0.8, -1)$. The

[3]Note the corner case of states where both players only have one action: they could be assigned to either player, but we choose to let them belong to Attacker.

[4]Formally, to transfer a reachability objective to a total reward objective, one can add a dummy state and give the target state only one transitions with reward 1 that certainly goes to the dummy state.

former vector is achievable by not trapping the Attacker, i.e. not incurring any cost but also having no chance of trapping. The latter point is achievable by trapping, paying 1 cost, but also getting the full 0.8 probability to trap the Attacker. Any vector on the line between the corners can be achieved by randomizing between the two actions. Any vector to the left and below of the line is achievable, because Defender can actually achieve even better vectors.

III. Problem statement

Given a DIFT-SG G with unknown trapping probabilities $p(s)$, we want to approximate the set of achievable vectors $\mathcal{A}(\mathsf{G}, s_0, \overline{\mathsf{TR}})$, where $\overline{\mathsf{TR}}$ consists of the following four total reward objectives: i) Reaching the trapped state τ ii) Reaching the dropout state ϕ iii) Avoiding the destination state d iv) Minimizing the cost. Our definition of achievable vectors requires that Defender is maximizing the reward in all dimensions. Hence some rewards are negative, e.g. instead of minimizing the cost, Defender maximizes the negative cost. Thus, the reward structures for the objectives are the following: for the two reachability objectives, the reward is 1 upon entering the target. For avoiding the destination, the reward is -1 upon entering the destination. For a triple of state, attacker action and defender action $(s, a_{\mathcal{D}}, a_{\mathcal{A}})$, the cost is 0 if $a_{\mathcal{D}} = 0$ and some given negative rational number $C(s)$ if $a_{\mathcal{D}} = 1$.

We comment on some differences to the problem formulation in the previous paper [16] that introduced the analysis of DIFT-SG: On the side of the model, our notion of DIFT-SG allows for cycles in the game graph. On the side of the objective, instead of merging all 4 dimensions of interest (trapping, dropout, destination and cost) into a single utility function, we have separated the goals in order to be able to identify the trade-offs between them. Further, instead of looking for Nash-equilibria, our work identifies the worst-case rewards, i.e. the rewards Defender can guarantee against any strategy of the Attacker. A point on the border of the set of achievable vectors need not be a Nash equilibrium. Still, for any point-wise maximal vector in the set of achievable vectors, the Attacker cannot switch to a strategy that prevents Defender from achieving this vector of rewards.

IV. Algorithm

The problem is hard for two reasons: Firstly, the trapping probabilities are unknown. Secondly, there are no algorithms for approximating the sets of achievable points in multi-objective concurrent SGs[5].

We propose a three step approach to solve these issues: First, we obtain confidence intervals for the unknown trapping probabilities by running simulations and then apply bounded-parameter reasoning in order to obtain guaranteed estimates. Second, we take advantage of the special structure of the DIFT-SG and transform the concurrent SG into a turn-based one. Third, we apply the standard algorithm for computing achievable vectors in turn-based SGs, implemented in PRISM-games [14].

Theorem 1: Given a DIFT-SG G with unknown trapping probabilities, we can probably approximately correctly (PAC) under- and over-approximate the set of achievable vectors $\mathcal{A}(\mathsf{G}, s_0, \overline{\mathsf{TR}})$, i.e. the approximants are correct with probability $1 - \delta$ and differ by at most ε for some $\delta, \epsilon \in (0, 1)$.

In the following, we show how to construct turn-based SGs $\hat{\mathsf{G}}^{-c}$ and $\hat{\mathsf{G}}^{+c}$ such that $\mathcal{A}(\hat{\mathsf{G}}^{-c}, s_0, \overline{\mathsf{TR}}) \subseteq \mathcal{A}(\mathsf{G}, s_0, \overline{\mathsf{TR}}) \subseteq \mathcal{A}(\hat{\mathsf{G}}^{+c}, s_0, \overline{\mathsf{TR}})$. Theorem 1 follows by combining Lemma 1 and Lemma 2 below.

A. Step 1: Estimating unknown trapping probabilities

A DIFT-SG only exhibits probabilistic behaviour when trapping a flow in a certain state. Thus, for every state s we want to estimate the trapping probability $p(s)$. We do so by running simulations of the system, e.g. by simulating an attack and trying to trap the attacker in state s or by analyzing system logs. Then for every state s we get the number of times we tried to trap the flow $\mathit{total}(s)$ and the number of times the trapping succeeded $\mathit{trap}(s)$.

Based on these simulations, we give a probabilistic guarantee with some error tolerance δ. It guarantees that with probability $1 - \delta$, i.e. unless our sampling was unlucky, the following statement holds: by estimating every trapping probability as the empirical average $\frac{\mathit{trap}(s)}{\mathit{total}(s)}$ minus some confidence width c, we ensure that we get an under-approximation of the achievable points. Dually, by adding the confidence width, we get an over-approximation. Note that the confidence width depends on the error tolerance δ. We formalize this claim in Lemma 1 and prove it in Appendix I.

Lemma 1 (Estimating unknown probabilities): Let the confidence width of a state s be $c(s) = \sqrt{\frac{\ln(\delta/(2|\mathsf{S}|))}{-2\ \mathit{total}(s)}}$, where $\delta \in (0, 1)$. For a DIFT-SG G, let G^{-c} be the same SG except that for every state s the trapping probability $p(s)$ is given by $\frac{\mathit{trap}(s)}{\mathit{total}(s)} - c$. Analogously we define G^{+c} with $p(s) = \frac{\mathit{trap}(s)}{\mathit{total}(s)} + c$. Then with probability $1 - \delta$ it holds that for every state s:

$$\mathcal{A}(\mathsf{G}^{-c}, s, \overline{\mathsf{TR}}) \subseteq \mathcal{A}(\mathsf{G}, s, \overline{\mathsf{TR}}) \subseteq \mathcal{A}(\mathsf{G}^{+c}, s, \overline{\mathsf{TR}}).$$

Using Lemma 1, we can estimate the trapping probabilities from the simulations in such a way that solving the obtained SG results in a guaranteed under- or over-approximation of the achievable vectors. Increasing the number of simulations decreases the confidence width and thus improves the quality of the estimates. Since we compute both an under- and over-approximation, we can judge the error introduced by the uncertainty on the trapping probabilities and decide accordingly whether we need more simulations. Thus, we have reduced the problem to solving two concurrent DIFT-SG with known transition probabilities.

B. Step 2: Transformation to turn-based SG

We now give a way to transform a concurrent DIFT-SG into a turn-based one. It crucially relies on the special

[5]While multi-objective concurrent (non-repeated) games are considered already in the works of Blackwell [4] and Shapley [21] together with the notion of approachable sets of vectors, the subsequent research has focused mostly on equilibria rather than on algorithmics for the values, in contrast to the single-dimensional case, cf. [10].

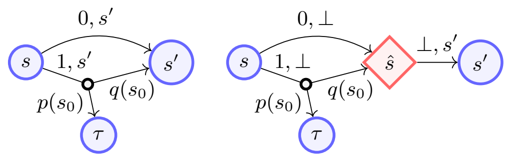


Fig. 3. Example of the transformation from concurrent DIFT-SG to turn-based SG. To avoid clutter, we use $q(s_0) = 1 - p(s_0)$

structure of the DIFT-SG and is not applicable in general. The key idea is depicted in Figure IV-B: for every state s that is not one of the sinks (i.e. not τ, ϕ or d), we create a copy $\hat{s}$ of it. We use this to split the concurrent choice. Defender chooses whether to trap or not in the original state s and, if the trapping was not successful, Attacker chooses an action in the new state $\hat{s}$ afterwards. Replacing the concurrent choice with a consecutive one gives the second player, Attacker, more information, namely that in $\hat{s}$ the trapping was not successful. However, as we will prove, this does not affect the possible outcomes of the game.

Formally, given a DIFT-SG G, we define its turn-based equivalent as $\hat{\mathsf{G}} = (\mathsf{S} \cup \hat{\mathsf{S}}, \mathsf{A} \cup \{\perp\}, \hat{\mathsf{A}}_{\mathcal{A}}, \hat{\mathsf{A}}_{\mathcal{D}}, \hat{\delta})$, where

- $\hat{\mathsf{S}} = \{\hat{s} \mid s \in \mathsf{S} \setminus \{d, \phi, \tau\}\}$ is a copy of all states except the sinks.
- we add a new dummy action $\perp$.
- for all states $s \in \mathsf{S}$, we have $\hat{\mathsf{A}}_{\mathcal{A}}(s) = \{\perp\}$ and $\hat{\mathsf{A}}_{\mathcal{D}}(s) = \mathsf{A}_{\mathcal{D}}(s)$ and dually for all states $\hat{s} \in \hat{\mathsf{S}}$ we have $\hat{\mathsf{A}}_{\mathcal{D}}(\hat{s}) = \{\perp\}$ and $\hat{\mathsf{A}}_{\mathcal{A}}(\hat{s}) = \mathsf{A}_{\mathcal{A}}(s)$.
- for $s \in \mathsf{S} \setminus \{d, \phi, \tau\}$ the transition function is defined as follows[6]: in case of no trapping $\hat{\delta}(s, 0, \perp)(\hat{s}) = 1$; in case of trapping $\hat{\delta}(s, 1, \perp)(\hat{s}) = 1 - p(s)$ and $\hat{\delta}(s, 1, \perp)(\tau) = p(s)$. For the Attacker controlled states $\hat{s} \in \hat{\mathsf{S}}$ and any successor state $s' \in \hat{\mathsf{A}}_{\mathcal{A}}(\hat{s})$ we have $\hat{\delta}(\hat{s}, \perp, s')(s') = 1$. The transition function for the sink states $\{d, \phi, \tau\}$ is not changed, as anyway both players only have dummy actions leading to a self-loop.

Lemma 2 (Transformation to turn-based SG): For some DIFT-SG G, let $\hat{\mathsf{G}}$ be its turn-based version. It holds for all states $s \in \mathsf{S}$ that $\mathcal{A}(\mathsf{G}, s, \overline{\mathsf{TR}}) = \mathcal{A}(\hat{\mathsf{G}}, s, \overline{\mathsf{TR}})$.
The proof is in Appendix II.

C. Step 3: Solving turn-based SG

An algorithm for solving turn-based SGs with a multiple total reward objectives is described in [6] and implemented in the well-known model checker PRISM-games [14]. The algorithm applies a generalization of value iteration, cf. [3, Ch. 10.6.1]. In the k-th iteration, it computes the set of vectors achievable in k steps, see Proposition 2 in the full version of [6]. Consequently, the algorithm converges towards the true set of achievable points.

V. Experiments

Our implementation builds on the code for obtaining probability estimates from [2] and the implementation of the value iteration from [6], both of which are implemented as part of PRISM-games [14]. Our code and the results for all case studies are available at `https://github.com/kushgrover/apt-vs-dift`.

[6]Note that we only specify all non-zero elements of the distribution. For all other states, the transition probability is 0.

TABLE I
Model properties and runtime per case study.

Example	Size	Cyclicity	Time taken (s)
Random	10	No	7.78
	100	No	11.90
Random	10	Yes	8.29
	100	Yes	17.63
ScreenGrab	9	no	7.95
NationState	30	yes	8.40

The considered benchmarks are ScreenGrab [20] and NationState Attack [17], as well as randomly generated graphs. Table I reports for each case study its size, whether it contains cycles and the runtime of all three steps of our algorithm combined, where the error tolerance was $\delta = 0.01$ and the resulting over- and under-approximation never differ by more than $\varepsilon = 0.1$. It shows that our our approach is able to deal with both real world systems as well as larger cyclic models in reasonable time even on a consumer grade computer with 32 GB RAM and a 2.60 GHz CPU.

When performing the experiments, we observed the following interesting facts: The reward in the dropout dimension is always 0. This makes sense, as the dropout action is under the control of Attacker, and thus Defender can never guarantee that Attacker drops out. Intuitively, since we compute the worst-case reward, this includes the case of a very aggressive Attacker with no intention of aborting the attack. Similarly, we observed that the reward for trapping in a state s is never higher than $p(s)$. This is because we cannot guarantee that Attacker will continue to play after the first step, and thus we cannot aggregate more trapping probability. Intuitively, the worst-case also includes the case of a very cautious Attacker that immediately aborts after the first trapping attempt.

We explain the analysis of the set of achievable vectors on the example of NationState. For the other case studies, the results look similar. We omit the dropout dimension, which is always 0 and display the resulting three-dimensional object from multiple viewpoints in Figure 4. The figure shows the achievable vectors for G^{-c} , G and G^{+c}, but we can barely make out a difference between the lines, proving that our approximations are close enough. The plots showing the relation between cost and both destination and trapped look as explained in Example 1. In fact, in this model Attacker has a path of length one to the destination, so Defender only gets the chance to trap once. For the relation between trapped and destination, we see that (by playing trap) Defender can achieve anything with trapping probability less than $p(s_0)$ and probability to reach the destination less than $1 - p(s_0)$. Note that, because of the way we defined the reward structures, when projecting to two dimensions the following holds: every achievable vector implies that any

.

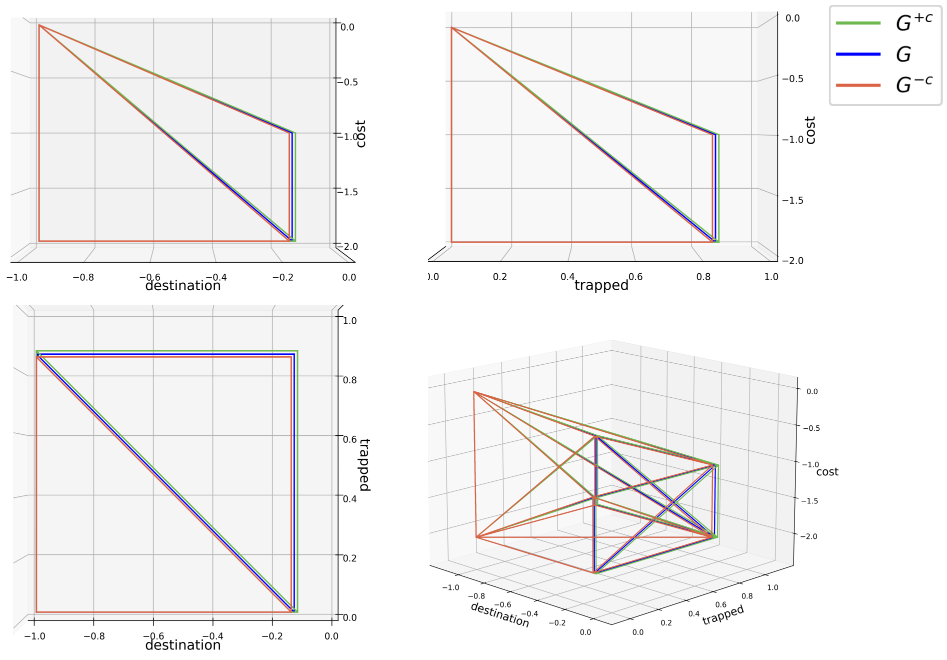


Fig. 4. Achievable vectors of the NationState Attack case study. We give the three dimensional view as well as projections to all 2d planes. We show the frontiers of G^{-c} , G and G^{+c} in red, blue and green respectively.

point below or to the left of it is also achievable.

VI. Conclusion

We have presented a novel solution to DIFT detection of APTs. Compared to the state of the art, our approach applies to a wider set of scenarios, can answer more refined questions on trade-offs between resource efficiency and detection efficacy, and provides results with a guaranteed precision.

Note that arbitrary precision can be achieved since the algorithm of [6] converges in the limit to the set of achievable vectors. Moreover, for SGs without cycles, the difference between the current approximation and the true set can be even bounded by the criterion given in [6, Theorem 4]. If cycles are present, we can still rely on the convergence, but a criterion for the current difference is an interesting point for future work. Recently, a convergent over-approximation technique was introduced [1], giving such a criterion for multi-dimensional reachability objective. We conjecture that it can be extended to total reward.

References

[1] Pranav Ashok, Krishnendu Chatterjee, Jan Kretínský, Maximilian Weininger, and Tobias Winkler. Approximating values of generalized-reachability stochastic games. In *LICS*, pages 102–115. ACM, 2020.

[2] Pranav Ashok, Jan Kretínský, and Maximilian Weininger. PAC statistical model checking for markov decision processes and stochastic games. In *CAV (1)*, volume 11561 of *Lecture Notes in Computer Science*, pages 497–519. Springer, 2019.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[4] D. Blackwell. An analog of the minimax theorem for vector payoffs. *Pac J Math*, 6(1):1–8, 1956.

[5] Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. Strategy improvement for concurrent reachability and turn-based stochastic safety games. *J. Comput. Syst. Sci.*, 79(5):640–657, 2013.

[6] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. On stochastic games with multiple objectives. In *MFCS*, volume 8087 of *Lecture Notes in Computer Science*, pages 266–277. Springer, 2013.

[7] Eric Cole. *Advanced persistent threat: understanding the danger and how to protect your organization*. Newnes, 2012.

[8] Luca de Alfaro, Thomas A. Henzinger, and Orna Kupferman. Concurrent reachability games. In *FOCS*, pages 564–575. IEEE Computer Society, 1998.

[9] Alex Devonport and Murat Arcak. Estimating reachable sets with scenario optimization. In *Learning for dynamics and control*, pages 75–84. PMLR, 2020.

[10] Kristoffer Arnsfelt Hansen, Michal Koucký, Niels Lauritzen, Peter Bro Miltersen, and Elias P. Tsigaridas. Exact algorithms for solving stochastic games: extended abstract. In *STOC*, pages 205–214. ACM, 2011.

[11] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[12] Linan Huang and Quanyan Zhu. Adaptive strategic cyber defense for advanced persistent threats in critical infrastructure networks. *SIGMETRICS Perform. Evaluation Rev.*, 46(2):52–56, 2018.

[13] Yang Ji, Sangho Lee, Evan Downing, Weiren Wang, Mattia Fazzini, Taesoo Kim, Alessandro Orso, and Wenke Lee. RAIN: refinable attack investigation with on-demand inter-process information flow tracking. In *CCS*, pages 377–390. ACM, 2017.

[14] Marta Kwiatkowska, David Parker, and Clemens Wiltsche. Prism-games 2.0: A tool for multi-objective strategy synthesis for stochastic games. In *TACAS*, volume 9636 of *Lecture Notes in Computer Science*, pages 560–566. Springer, 2016.

[15] Kong-wei Lye and Jeannette M. Wing. Game strategies in network security. *Int. J. Inf. Sec.*, 4(1-2):71–86, 2005.

[16] Shruti Misra, Shana Moothedath, Hossein Hosseini, Joey Allen, Linda Bushnell, Wenke Lee, and Radha Poovendran. Learning equilibria in

stochastic information flow tracking games with partial knowledge. In *CDC*, pages 4053–4060. IEEE, 2019.

[17] Shana Moothedath, Dinuka Sahabandu, Joey Allen, Andrew Clark, Linda Bushnell, Wenke Lee, and Radha Poovendran. A game-theoretic approach for dynamic information flow tracking to detect multistage advanced persistent threats. *IEEE Trans. Autom. Control.*, 65(12):5248–5263, 2020.

[18] Shana Moothedath, Dinuka Sahabandu, Andrew Clark, Sangho Lee, Wenke Lee, and Radha Poovendran. Multi-stage dynamic information flow tracking game. In *GameSec*, volume 11199 of *Lecture Notes in Computer Science*, pages 80–101. Springer, 2018.

[19] Dinuka Sahabandu, Shana Moothedath, Linda Bushnell, Radha Poovendran, Joey Allen, Wenke Lee, and Andrew Clark. A game theoretic approach for dynamic information flow tracking with conditional branching. In *ACC*, pages 2289–2296. IEEE, 2019.

[20] Dinuka Sahabandu, Baicen Xiao, Andrew Clark, Sangho Lee, Wenke Lee, and Radha Poovendran. DIFT games: Dynamic information flow tracking games for advanced persistent threats. In *CDC*, pages 1136–1143. IEEE, 2018.

[21] L.S. Shapley. Equilibrium points in games with vector payoffs. *Nav Res Logist Q*, 6:57–61, 1959.

[22] Xiaokui Shu, Ke Tian, Andrew Ciambrone, and Danfeng Yao. Breaking the target: An analysis of target data breach and lessons learned. *CoRR*, abs/1701.04940, 2017.

[23] G Edward Suh, Jae W Lee, David Zhang, and Srinivas Devadas. Secure program execution via dynamic information flow tracking. *ACM Sigplan Notices*, 39(11):85–96, 2004.

[24] Maximilian Weininger, Tobias Meggendorfer, and Jan Kretínský. Satisfiability bounds for ω-regular properties in bounded-parameter markov decision processes. In *CDC*, pages 2284–2291. IEEE, 2019.

[25] Bai Xue, Miaomiao Zhang, Arvind Easwaran, and Qin Li. PAC model checking of black-box continuous-time dynamical systems. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 39(11):3944–3955, 2020.

[26] Quanyan Zhu, Andrew Clark, Radha Poovendran, and Tamer Basar. Deceptive routing games. In *CDC*, pages 2704–2711. IEEE, 2012.

Appendix I
Proof of Lemma 1

We view the simulations of trapping in a state s as a Bernoulli sequence with success probability $p(s)$, number of trials *total*(s) and number of successes *trap*(s). We uniformly distribute the allowed error probability δ over the $|\mathsf{S}|$ many states. Then we use the Hoeffding bound [11, Theorem 1] to obtain the probabilistic guarantee on the confidence intervals around the empirical average as follows:

$$\mathbb{P}\left(|\frac{\mathit{trap}(s)}{\mathit{total}(s)} - p(s)| \geq c\right) \leq 2 \cdot e^{-2c^2 \mathit{total}(s)} \\ \leq \delta / |\mathsf{S}|$$

In words, this application of the Hoeffding bound says that the probability of the empirical average for a state s being more than c away from the true trapping probability $p(s)$ is less than $\delta/|\mathsf{S}|$. Thus, the error probability over all states is δ. Solving the inequation for the confidence width yields $c \geq \sqrt{\frac{\ln(\delta/(2|\mathsf{S}|))}{-2\mathit{total}(s)}}$, and thus our choice of confidence width is correct.

Given the guaranteed confidence interval for the trapping probability, we still have to prove that picking the smallest/greatest values for all states results in an under-/over-approximation of the achievable vectors. Intuitively, we use that the probabilistic transitions are very simple in that they have only two successors. Additionally, one successor is the trap state which is clearly good for the Defender and bad for the Attacker. Thus, to under-approximate we pick the smallest possible trapping probability and to over-approximate the largest one.

Formally, we apply the reasoning for bounded-parameter SGs from [24]. We start by recalling the construction described in their Theorem 1. Given an SG G with intervals on the transition probabilities, their algorithm first computes the basic feasible solution (BFS, cf. [24, Section IV.A]) of the constraints given by the intervals for every action. Mixing these BFS allows to imitate any valid instantiation of the intervals. Then we can construct a new SG G′, where every action leads to a new state; this new state has an action for every BFS, allowing it to mimic any valid distribution. Depending on the player the new states belong to, the resulting modified system either over-approximates or under-approximates the actual value. Note the following differences between the algorithm of [24] and our setting: We consider concurrent SG with multiple expected total rewards, while they consider SG (cf. [24, Remark 5]) with a single omega-regular objective. Concurrency is not a problem, as we can introduce a new state for every pair of actions. The proof of [24, Theorem 1] does not rely on any specific property of omega-regular objectives, but instead only argues about the transition function, so considering multiple total reward objectives works completely analogous.

It remains to show that we can simplify the construction of G' to just replacing all transition probabilities with the extremal values from the intervals. For ease of notation, let $p^- = \frac{\mathit{trap}(s)}{\mathit{total}(s)} - c$ be the minimum value of the confidence interval and analogously define p^+.

First, observe that we know the graph structure of the given DIFT-SG G. Since we know that all transitions where the Defender does not trap have probability 1, we do not need to change anything about them. If the Defender traps, the transition has two possible successors: the trap state τ or some other state s'. As there are only two successors, the basic feasible solutions are very simple: the new state has one action that goes to τ with p^- and to s' with the remaining probability and one action that goes to τ with p^+ and to s' with the remaining probability. By mixing these actions, the state can imitate any trapping probability between p^- and p^+. However, note that, depending on the player the state belongs to, an optimal strategy will always pick only one of the actions. If the state belongs to the Defender, using the action with p^+ is optimal for all objectives. It increases the probability to reach the trap state and thereby decreases the probability to reach the target. As in the trap state no further cost is accumulated, increasing the chances of reaching it also serves to minimize the cost. The dropout dimension is irrelevant, as Defender cannot guarantee any positive reward in this dimension (see Section V for a discussion of this fact). By the dual argumentation, if the state belongs to Attacker, the action with probability p^- is optimal. Thus, we see that instead of surely going to the new state and then always picking the p^- action in the case of an under-approximation (state belongs to Attacker) or the p^+ action in case of an over-approximation, we can just replace the trapping probability on the original action with p^- respectively p^+.

Appendix II
Proof of Lemma 2

In the following, let G be a DIFT-SG and $\hat{\mathsf{G}}$ its turn-based version. To show that the achievable vectors in both games are equal, we use contraposition. That means we show that if a point is not achievable in one game, it is also not achievable in the other. We use the statement "Attacker can spoil a vector" to say that for all Defender strategies there exists an Attacker strategy under which the vector is not achievable.

Our goal is thus reformulated as follows: Attacker can spoil a vector in the concurrent DIFT-SG G if and only if Attacker can spoil it in the turn-based version $\hat{\mathsf{G}}$.

In order to argue about the expected total reward, we argue about the probability space over paths induced by the SGs and the pair of strategies. Note that there exists a bijection between paths[7] in the SGs by adding/omitting the new dummy actions $\perp$ and the new state $\hat{s}$. A path $\varphi = s_0 d_0 a_0 s_1 \dots$ in the concurrent DIFT-SG can be mapped to a path $\hat{\varphi} = s_0 d_0 \perp \hat{s}_0 \perp a_0 s_1$ and vice versa. Observe that the reward of a path and its equivalent are equal, as non-zero rewards are only on the trap action and the special actions in target states of a reachability objective.

The proofs for both directions of the if and only if statement are very similar. We start by proving the simpler direction (concurrent implies turn-based) to introduce the concepts of the proof. For the harder direction (turn-based implies concurrent) we only describe the additional ideas.

Direction $\rightarrow$: We assume that for all Defender strategies $\pi_{\mathcal{D}}$ Attacker has a strategy $\pi_{\mathcal{A}}$ to spoil an arbitrary point x in the concurrent SG G. Using this, we have to show that for all Defender strategies $\hat{\pi}_{\mathcal{D}}$ Attacker also has a strategy $\hat{\pi}_{\mathcal{A}}$ to spoil x in the turn-based version $\hat{\mathsf{G}}$. Intuitively, Attacker has more information in the turn-based SG, so naturally anything spoilable in G is also spoilable in $\hat{\mathsf{G}}$.

Formally, let $\hat{\pi}_{\mathcal{D}}$ be an arbitrary Defender strategy in the turn-based SG. We define the equivalent strategy in the concurrent game as $\pi_{\mathcal{D}}(\varphi) = \hat{\pi}_{\mathcal{D}}(\hat{\varphi})$, i.e. map the path to one in the turn-based SG and apply $\hat{\pi}_{\mathcal{D}}$. Then by assumption there is an Attacker strategy $\pi_{\mathcal{A}}$ so that x is spoiled in the concurrent SG under $\pi_{\mathcal{D}}$ and $\pi_{\mathcal{A}}$. Let $\hat{\pi}_{\mathcal{A}}(\hat{\varphi} d \perp s') = \pi_{\mathcal{A}}(\varphi)$ be the Attacker strategy for the turn-based SG; it maps the path to its concurrent equivalent (dropping the additional information about the Defender's most recent action) and then uses $\pi_{\mathcal{A}}$. It remains to show that the probability space over paths induced by $\hat{\pi}_{\mathcal{D}}$ and $\hat{\pi}_{\mathcal{A}}$ in $\hat{\mathsf{G}}$ is equivalent to the one induced by $\pi_{\mathcal{D}}$ and $\pi_{\mathcal{A}}$ in G, because then we arrive at the conclusion that x is also spoilable in $\hat{\mathsf{G}}$.

Thus we show that, under the strategies just described, the following two probabilities are equal: the probability to go to $s' \in \mathsf{S}$ in one step in the concurrent SG after seeing an arbitrary path φs ending in some state $s \in \mathsf{S}$; and the probability to go to s' in two steps in the turn-based SG after seeing the equivalent path $\hat{\varphi} s$.

First, we argue about the special cases of $s \in \{\tau, \phi, d\}$. As we loop in the sink state, in both SGs the probability to stay is 1. Second, we address the special case of $s' = \tau$, i.e. the Defender succeeding to trap. In the concurrent SG, the probability to go from s to τ in one step is $\pi_{\mathcal{D}}(\varphi s)(1) \cdot p(s)$, i.e. the probability that Defender assigns to the trapping action in state s times the trapping probability. In the turn-based SG, it is $\hat{\pi}_{\mathcal{D}}(\hat{\varphi} s)(1) \cdot p(s)$, which is equal to $\pi_{\mathcal{D}}(\varphi s)(1) \cdot p(s)$ by definition of $\pi_{\mathcal{D}}$. Note that in the second step the play loops in τ surely. Finally, we address the "normal" case of starting in any non-sink state s and going to any state $s' \in \mathsf{S} \setminus \{\tau\}$. We denote by $\diamond^k s'$ the set of all paths that reach s' in k steps. The following chain of equations proves our goal. The first equality unfolds the strategies for one step and makes a case distinction over the choice of Defender. The second equality applies the definitions of $\hat{\pi}_{\mathcal{A}}$ and $\hat{\pi}_{\mathcal{D}}$. The third equality uses the observation that the left side describes exactly all two step paths from s to s' in $\hat{\mathsf{G}}$.

$$
\begin{aligned}
&\mathbb{P}^{\pi_{\mathcal{D}}, \pi_{\mathcal{A}}}_{\mathsf{G}, s}[\diamond^1 s'] \\
&= \pi_{\mathcal{D}}(\varphi s)(0) \cdot \pi_{\mathcal{A}}(\varphi s)(s') \\
&\quad + \pi_{\mathcal{D}}(\varphi s)(1) \cdot (1 - p(s)) \cdot \pi_{\mathcal{A}}(\varphi s)(s') \\
&= \hat{\pi}_{\mathcal{D}}(\hat{\varphi} s)(0) \cdot \hat{\pi}_{\mathcal{A}}(\hat{\varphi} s 0 \perp \hat{s})(s') \\
&\quad + \hat{\pi}_{\mathcal{D}}(\hat{\varphi} s)(1) \cdot (1 - p(s)) \cdot \hat{\pi}_{\mathcal{A}}(\varphi s 1 \perp \hat{s})(s') \\
&= \mathbb{P}^{\hat{\pi}_{\mathcal{D}}, \hat{\pi}_{\mathcal{A}}}_{\hat{\mathsf{G}}, s}[\diamond^2 s']
\end{aligned}
$$

Thus, using the probability measure over paths in G under $\pi_{\mathcal{D}}$ and $\pi_{\mathcal{A}}$ is the same as using the probability measure over paths in $\hat{\mathsf{G}}$ under $\hat{\pi}_{\mathcal{D}}$ and $\hat{\pi}_{\mathcal{A}}$ and then mapping them to their equivalent path in the concurrent SG. Thus, as noted above, we arrive at the conclusion that x is also spoilable in $\hat{\mathsf{G}}$.

Direction $\leftarrow$: Assuming that for all Defender strategies $\hat{\pi}_{\mathcal{D}}$ Attacker has a strategy $\hat{\pi}_{\mathcal{A}}$ to spoil an arbitrary point x in the turn-based SG $\hat{\mathsf{G}}$, we have to show that for all Defender strategies $\pi_{\mathcal{D}}$ Attacker also has a strategy $\pi_{\mathcal{A}}$ to spoil x in the concurrent SG G. The proof is analogous to the previous case, except that obtaining $\pi_{\mathcal{A}}$ from $\hat{\pi}_{\mathcal{A}}$ is more involved, since $\hat{\pi}_{\mathcal{A}}$ uses more information.

Let $\pi_{\mathcal{D}}$ be an arbitrary Defender strategy for the concurrent SG and $\hat{\pi}_{\mathcal{D}}$ its equivalent as in the previous proof. Then there exists a $\hat{\pi}_{\mathcal{A}}$ that spoils x in $\hat{\mathsf{G}}$. We need to find a $\pi_{\mathcal{A}}$ that induces the same probability space over paths in G as using $\hat{\pi}_{\mathcal{A}}$ for $\hat{\mathsf{G}}$ and then mapping the paths to their concurrent equivalent. Formally, we need a $\pi_{\mathcal{A}}$ so that the chain of equations we used in the proof of the other direction holds.

By using the fact that for $i \in \{0,1\} : \pi_{\mathcal{D}}(\varphi s)(i) = \hat{\pi}_{\mathcal{D}}(\hat{\varphi} s)(i)$ and solving the second and third line of the sequence of equations for $\pi_{\mathcal{A}}$, we obtain $\pi_{\mathcal{A}}(\varphi s)(s') = \frac{\pi_{\mathcal{D}}(\varphi s)(0) \cdot \hat{\pi}_{\mathcal{A}}(\hat{\varphi} s 0 \perp \hat{s})(s') \, + \, \pi_{\mathcal{D}}(\varphi s)(1) \cdot (1 - p(s)) \cdot \hat{\pi}_{\mathcal{A}}(\hat{\varphi} s 1 \perp \hat{s})(s')}{\pi_{\mathcal{D}}(\varphi s)(0) + \pi_{\mathcal{D}}(\varphi s)(1) \cdot (1 - p(s))}$. Setting $\pi_{\mathcal{A}}$ like this, using the given probabilities of the Defender strategy and the spoiling Attacker strategy in the turn-based game, we ensure that the chain of equations hold. As a sanity check, observe that $\sum_{s' \in \mathsf{S}} \pi_{\mathcal{A}}(\varphi s)(s') = 1$, so $\pi_{\mathcal{A}}$ is indeed a valid strategy.

Intuitively, we just showed that any way in which Attacker uses the additional information in the turn-based SG can be recreated by using a certain probability distribution in the concurrent SG.

[7] more formally: finite prefixes of paths ending in a state $s \in \mathsf{S}$