Fakultät für Informatik
Technische Universität München

Verification of Discrete-Time Markov Decision Processes

Tobias Meggendorfer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:
Prof. Dr.-Ing. Matthias Althoff

Prüfende der Dissertation:
1. Prof. Dr. Jan Křetínský
2. Prof. Dr. Christel Baier

Die Dissertation wurde am 17.09.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 17.02.2021 angenommen.

Acknowledgements

First, I thank all co-authors of the papers appended to this thesis, in particular my advisor Jan Křetínský, who endured countless discussions and taught me a great deal about math, science, academia, writing, language, life, and many other topics. I thank my parents and friends, who have heard way too much about all the hardships of working with students and probably know all variants of the grid world model by now. Finally, I also thank all the people from Lehrstuhl VII for the relaxed-yet-focused atmosphere, with discussions moving from cookie recipes over applications of verification to kitchen appliances to a concrete research idea in the span of minutes.

Abstract

This thesis deals with the verification of probabilistic systems in the setting of discrete time, in particular *Markov decision processes* (MDP). We present both theoretical as well as practical contributions.

First, we solve a long-open problem related to *mean payoff* queries and present two novel algorithms to efficiently compute such queries. The first algorithm is based on value iteration and additionally is augmented with partial-exploration techniques, focussing computational effort on important parts of the system. The second algorithm is an efficient implementation of strategy iteration, improved by several topological optimizations as well as approximation techniques, yet maintaining precise answers.

Then, we provide efficient practical implementations of *LTL-to-automata* translation algorithms. These play a central role in *probabilistic LTL model checking*.

Moreover, we introduce the novel notion of *cores* in an MDP, which provide a framework to speed up many approximation-based tasks by automatically identifying and removing 'irrelevant' states. Additionally, cores provide us with a well-founded notion of *importance*, aiding in the understandability of systems.

Finally, we present a significant step towards *risk-aware* analysis of probabilistic systems. In particular, we introduce the notion of *conditional value-at-risk* (CVaR) to MDP and show that risk-aware verification lies in the same complexity class as pure expectation maximization in our particular application.

Attached to this thesis are papers published at conferences CAV 2017, ATVA 2017, LICS 2018, CAV 2018, ATVA 2018, and CONCUR 2019.

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Verifikation von probabilistischen Systemen in diskreter Zeit, insbesondere *Markov Entscheidungsprozessen* (MDP). Wir präsentieren sowohl theoretische als auch praktische Fortschritte.

Zuerst lösen wir eine lange offene Frage bezüglich *mean payoff* Problemen. Wir definieren zwei neue Algorithmen, die solche Probleme effizient lösen. Der erste Algorithmus basiert auf der Methodologie der Werte-Iteration und kann durch zusätzliche Teil-Erkundungs-Ansätze den Rechenaufwand auf wichtige Teile des Systems konzentrieren. Der zweite Algorithmus ist eine effiziente Implementierung des Strategie-Iteration Ansatzes. Dieser wird durch mehrere topologische Optimierungen und ein approximatives Verfahren weiter verbessert. Trotz Verwendung approximativer Methoden liefert letzterer Algorithmus präzise Ergebnisse.

Dann präsentieren wir eine effiziente Implementierung von *LTL-zu-Automaten* Übersetzungsalgorithmen. Diese spielen insbesondere im Kontext der probabilistischen Verifikation von LTL Formeln eine zentrale Rolle.

Danach beschreiben wir das neue Konzept des *Kerns* eines MDP. Diese Kerne bieten eine flexible Grundlage um approximative Berechnungen auf MDP zu beschleunigen, indem „unwichtige“ Zustände automatisch entfernt werden. Zusätzlich liefern Kerne eine mathematische Definition für „Wichtigkeit“ von verschiedenen Zuständen. Dies bietet eine systematische Basis für Erklärbarkeit von solchen Systemen.

Zuletzt präsentieren wir einen wichtigen Schritt in Richtung risikobewusster Analyse von probabilistischen Systemen. Wir definieren das Konzept der *conditional value-at-risk* (CVaR) auf MDP und zeigen, dass eine Analyse unter der Berücksichtigung von Risiko in der selben Komplexitätsklasse wie pure Erwartungswert-Maximierung liegt.

Die in dieser Dissertation eingebunden Papiere wurden in den Konferenzen CAV 2017, ATVA 2017, LICS 2018, CAV 2018, ATVA 2018 und CONCUR 2019 veröffentlicht.

Contents

List of Figures

1 Introduction

The analysis of probabilistic systems arises in numerous application contexts, e.g., randomized communication and security protocols, stochastic distributed systems, biological processes, artificial intelligence, speech recognition, and robot planning, to name a few. Such systems comprise both stochastic and non-deterministic elements. The former captures probabilistic influences of the environment, i.e. events where the outcome is beyond our control, but also is not consciously antagonistic to our goals, e.g. a coin toss. The latter represents deliberate decision making, where agents interacting with the environment, e.g., humans or robots, are presented with a set of options and are able to choose among them. An elegant mathematical framework for the analysis of probabilistic systems are *Markov decision processes (MDP)* [How60; FV96; Put94]; see [Whi85; Whi88; Whi93] for an extensive list of applications. MDP are built from three ingredients, namely *states*, *actions*, and *(probabilistic) transitions.* Intuitively, a state fully describes a 'snapshot' of the current configuration of the system. At each state, the agent can choose from a set of actions, corresponding to the *non-determinism* of the system. After choosing an action, the system then transitions into a successor state according to the probability distribution associated with that action.

For an intuitive example, we introduce the classical *grid world.* The grid world is, as the name suggests, a grid of, say, 4x4 cells. The inhabitants of our grid world are a remote controllable robot, some hazards, and a charging station. See Figure 1.1 for a pictorial representation. The state of the corresponding MDP is then given by the position of the robot. At any regular position, the robot can decide to move in any of the four cardinal direction; the hazards damage the robot and no actions are available there. Once the robot decides to move, say, in the northern direction, it transitions to

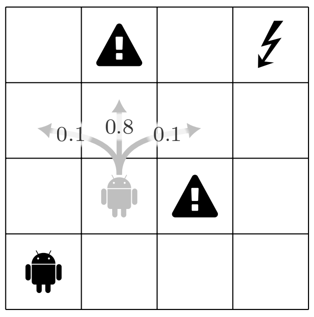


Figure 1.1: A small grid world. A remote controlled robot is in the lower left corner, a charging station in the upper right, and two hazards are located in between. The possible effects of going 'north' at position $(2, 2)$ are sketched in grey.

the cell above it with 80 % probability and with 10 % to the top-left or top-right cell, respectively, since there is a chance of a navigation failure (adapting appropriately at the grid boundary). Now, our goal may be to control the robot such that the probability of reaching the charging station is maximized and hazards are avoided. However, the battery of the robot might be running low, so we instead may be interested in getting to the charging station as fast as possible. Considering the differences between these two goals and their solutions in the example may provide the reader with valuable insights in the dynamics of MDP.

In general, the process of *verification* (or *model checking*) aims at analysing MDP with respect to a given formal description of such a goal, called *objective* or *specification*, and output a *provably correct* result. This kind of analysis is particularly appealing for safety-critical areas such as medicine or aerospace, where even small errors may have grave consequences. Here, rigorous methods give confidence in the correctness of an implementation or, in case a safety requirement is violated, may aid in identifying the root cause. This contrasts best-effort methods such as machine learning approaches, which excel at identifying 'good' solutions but (usually) are not able to certify their quality. As such, studying the formal verification of probabilistic systems is an important research direction with far reaching applications.

Clearly, a central aspect of verification is the formalism used to describe the objective. The first example (reaching the charging station) corresponds to a simple *reachability* query, where we are interested in reaching a particular state of the MDP. The second example (reaching the station as fast as possible) could be modelled as step-bounded reachability or, more generally, cost-bounded reachability. Apart from reachability, a wide variety of different objectives have been introduced and studied in the literature, each coming with its own challenges and widely varying computational complexity, ranging from linear time to undecidability. For example, we could be interested in guiding the robot such that it visits several points of interest or collects some resources while keeping energy consumption below a threshold. Consequently, we want to investigate the decidability and complexity of different objective formalisms in order to identify tractable problem classes.

It often turns out that solution approaches which are theoretically appealing due to their low computational complexity are vastly outperformed in practice by algorithms which are much worse in terms of worst-case. As such, a practical study of verification provides also provides us with many highly relevant questions.

In this thesis, we address several questions of both kinds.

1.1 Summary of Contributions

We present advances in four different directions, briefly summarized in this section.

Mean payoff intuitively describes 'What is the average reward obtained per step in the limit?'. For example, it can be used to describe the average throughput of a flow control protocol or the output of a power plant. Despite existing thorough analysis of its theoretical properties (see, e.g., [Put94, Chapter 8, 9]), there still are open questions. In Chapter 3, we disprove a long standing conjecture, provide an alternative solution, and introduce two different efficient algorithms. We furthermore augment one approach with partial-exploration techniques, achieving further improvements in practice.

Linear Temporal Logic allows to express more complex requirements on paths than pure reachability. For example, we can require *safety* (avoid a certain area), *liveness* (visit an area repeatedly), and combinations thereof. The associated verification problem on MDP has been thoroughly studied, with precise complexity bounds being known for the general problem as well as various sub-classes. Recent improvements of the *automaton theoretic approach* promise a vast increase in practical performance. However, these theoretical improvements lacked a proper implementation. We provide practical algorithms on top of a powerful framework for dealing with linear temporal logic in Chapter 4.

Cores provide a new perspective on probabilistic systems. Inspired by partial exploration approaches which identify a set of states that are relevant to answer a particular query, we consider the set states necessary to answer *any* query up to a given precision. This provides us with several benefits. Firstly, we can transparently combine the idea of cores with existing verification approaches to obtain practical speed-ups. Essentially, cores extend the idea of pruning non-reachable states by also pruning hardly reachable states. Moreover, cores provide a well-founded notion of 'importance'—states which are never relevant for any query naturally can be considered less important. We discuss cores, their application, and further interpretations in Chapter 5. In particular, cores help with the previous two areas.

Risk is an ubiquitous concept in reality and is highly influential for human decision making. Yet, verification hardly dealt explicitly with a quantitative notion of risk. We provide an extensive theoretical treatment of *conditional value-at-risk* applied to MDP in Chapter 6. In particular, we show that single-dimensional risk-aware model checking is possible in polynomial time, paving the way for efficient practical algorithms.

1.2 Summary of Publications

This is a publication-based thesis. We present the following papers in Appendix I.

A Pranav Ashok, Krishnendu Chatterjee, Przemyslaw Daca, Jan Křetínský and Tobias Meggendorfer. 'Value Iteration for Long-Run Average Reward in Markov Decision Processes'. In: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*. ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10426. Lecture Notes in Computer Science. Springer, 2017, pp. 201–221. DOI: `10.1007/978-3-319-63387-9\_10`. URL: `https://doi.org/10.1007/978-3-319-63387-9%5C_10`

B Jan Křetínský and Tobias Meggendorfer. 'Efficient Strategy Iteration for Mean Payoff in Markov Decision Processes'. In: *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings.* Ed. by Deepak D'Souza and K. Narayan Kumar. Vol. 10482. Lecture Notes in Computer Science. Springer, 2017, pp. 380–399. DOI: `10.1007/978-3-319-68167-2\_25`. URL: `https://doi.org/10.1007/978-3-319-68167-2%5C_25`

C Jan Křetínský, Tobias Meggendorfer, Salomon Sickert and Christopher Ziegler. 'Rabinizer 4: From LTL to Your Favourite Deterministic Automaton'. In: *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I.* ed. by Hana Chockler and Georg Weissenbacher. Vol. 10981. Lecture Notes in Computer Science. Springer, 2018, pp. 567–577. DOI: `10.1007/978-3-319-96145-3\_30`. URL: `https://doi.org/10.1007/978-3-319-96145-3%5C_30`

D Jan Křetínský, Tobias Meggendorfer and Salomon Sickert. 'Owl: A Library for ω-Words, Automata, and LTL'. in: *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings.* Ed. by Shuvendu K. Lahiri and Chao Wang. Vol. 11138. Lecture Notes in Computer Science. Springer, 2018, pp. 543–550. DOI: `10.1007/978-3-030-01090-4\_34`. URL: `https://doi.org/10.1007/978-3-030-01090-4%5C_34`

E Jan Křetínský and Tobias Meggendorfer. 'Of Cores: A Partial-Exploration Framework for Markov Decision Processes'. In: *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands.* Ed. by Wan Fokkink and Rob van Glabbeek. Vol. 140. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 5:1–5:17. DOI: `10.4230/LIPIcs.CONCUR.2019.5`. URL: `https://doi.org/10.4230/LIPIcs.CONCUR.2019.5`

F Jan Křetínský and Tobias Meggendorfer. 'Conditional Value-at-Risk for Reachability and Mean Payoff in Markov Decision Processes'. In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018.* Ed. by Anuj Dawar and Erich Grädel. ACM, 2018, pp. 609–618. DOI: `10.1145/3209108.3209176`. URL: `https://doi.org/10.1145/3209108.3209176`

All papers have been published in peer-reviewed conference proceedings and are self-contained. Each paper is prefaced with a brief summary and a list of the thesis author's contributions.

1.3 Outline

Chapter 2 introduces mathematical notation together with the model and objectives of interest. We begin with a definition of Markov decision processes and related concepts in Section 2.1. We proceed with an extensive list of objectives in Section 2.2. Finally, we conclude with classical solution approaches in Section 2.3. Then, Chapters 3 to 6 provide an overview of our contributions towards mean payoff analysis, probabilistic LTL model checking, the notion of cores, and risk-aware verification, in that order. Each chapter introduces the problem at hand, defines concepts specific to the respective problem and explains the state of the art (where applicable), summarizes our contributions, and finally concludes with brief remarks on future work. We only give a semi-formal overview of each paper; for technical treatment we refer the interested reader to the respective papers provided in the subsequent Appendix I. Finally, we provide the permissions to use and publish the papers as attached in Appendix II.

2 Preliminaries

The central object of this thesis are Markov decision processes, a classical model for systems which exhibit both stochastic and non-deterministic behaviour. In this chapter, we introduce these Markov processes as well as systematic approaches common to several of the following chapters. Concepts specific to a single topic are introduced in the respective chapters.

As usual, $\mathbb{N}$ and $\mathbb{R}$ refer to the (positive) natural numbers and real numbers, respectively. We write $[n] \coloneqq \{1, \dots, n\}$ to denote all natural numbers from 1 to n, with $[\infty] \coloneqq \mathbb{N}$. For two real numbers $a, b \in \mathbb{R}$ with $a \leq b$, the interval $[a, b] \subseteq \mathbb{R}$ denotes the set of all real numbers between a and b inclusively. We write $v[i] \in \mathbb{R}$ to denote the i-th component of a vector $v \in \mathbb{R}^n$.

For a set S, $\overline{S}$ denotes its complement, while $S^\star$ and S^ω refers to the set of finite and infinite sequences comprising elements of S, respectively. We use $\chi_S(s) = 1$ if $s \in S$ and 0 otherwise to refer to the *characteristic function* of S.

We assume familiarity with basic notions of probability theory, e.g., *probability spaces*, *probability measures*, and *measurability*; see e.g. [Bil08] for a general introduction. In particular, we omit precise treatment of sigma-fields and proofs of measurability for the sake of readability. We mention in-depth treatment of these issues in the literature at appropriate places. A *probability distribution* over a countable set X is a mapping $d : X \to [0, 1]$, such that $\sum_{x \in X} d(x) = 1$. Its *support* is denoted by $\mathrm{supp}(d) = \{x \in X \mid d(x) > 0\}$. $\mathcal{D}(X)$ denotes the set of all probability distributions on X. Some event happens *almost surely* (a.s.) if it happens with probability 1. For readability, we omit the discussions arising for uncountable domains X.

We direct the interested reader to, e.g., [Put94; BK08; For+11] for further information on the concepts introduced in this section. We primarily refer to these three works throughout the preliminaries in order to minimize notational discrepancies.

2.1 Markov Systems

First, we introduce Markov chains (MCs), which are purely stochastic and can be seen as a special case of MDP.

Definition 1. A *(discrete-time time-homogeneous) Markov chain (MC)* is a tuple $\mathsf{M} = (S, \delta)$, where

- S is a countable set of *states*, and
- $\delta : S \to \mathcal{D}(S)$ is a *transition function* that for each state s yields a probability distribution over successor states.

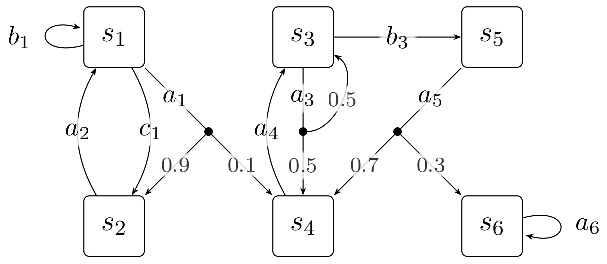


Figure 2.1: Example MDP, used to illustrate various concepts. Each action available in a state is represented by an outgoing edge, labelled by the action and several arrows connecting to the successor states, each labelled with the transition probability. For actions with only one successor, i.e. an action a with $\Delta(s, a)(s') = 1$ for some $s, s' \in S$, we omit the transition probability of 1 in the figure.

Markov chains can also be viewed as a sequence of random variables X_t, where X_{t+1} only depends on X_t [Put94, Appendix A.1]. Note that we do not require the set of states of a Markov chain to be finite. This is mainly due to technical reasons, which become apparent later. Now, we define MDP, which essentially extend Markov chains with non-determinism through actions.

Definition 2. A *(finite discrete-time time-homogeneous countable-state) Markov decision process (MDP)* is a tuple $\mathcal{M} = (S, \mathit{Act}, \mathit{Av}, \Delta)$, where

- S is a finite set of *states*,
- Act is a finite set of *actions*,
- $\mathit{Av} : S \to 2^{\mathit{Act}} \setminus \{\emptyset\}$ assigns a set of *available actions* to each state, and
- $\Delta : S \times \mathit{Act} \to \mathcal{D}(S)$ is a *transition function* that for each state s and (available) action $a \in \mathit{Av}(s)$ yields a probability distribution over successor states.

Note that we require the set of available actions to be non-empty. This means that a run can never get 'stuck' in a degenerate state without successors. See Figure 2.1 for an example MDP, used throughout this chapter to illustrate several concepts.

For ease of notation, we overload functions mapping 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 $\delta(s)(s')$ and $\Delta(s, a)(s')$ we write $\delta(s, s')$ and $\Delta(s, a, s')$, respectively. Furthermore, given a distribution $d \in \mathcal{D}(X)$ and a function $f : X \to \mathbb{R}$ mapping elements of a set X to real numbers, we write $d\langle f\rangle := \sum_{x \in X} d(x) f(x)$ to denote the weighted sum of f with respect to d. For example, $\delta(s)\langle f\rangle$ and $\Delta(s, a)\langle f\rangle$ denote the weighted sum of $f : S \to \mathbb{R}$ over the successors of s in an MC and s with action a in an MDP, respectively.

Finally, we always assume an arbitrary but fixed numbering of the states and identify a state with its respective number. For example, given a vector $v \in \mathbb{R}^{|S|}$ and a state $s \in S$, we may write $v[s]$ to denote the value associated with s by v. In this way, a function $v : S \to \mathbb{R}$ is equivalent to a vector $v \in \mathbb{R}^{|S|}$. In examples, we often number states by s_1, s_2, etc. There, we implicitly identify states s_i with their respective index i.

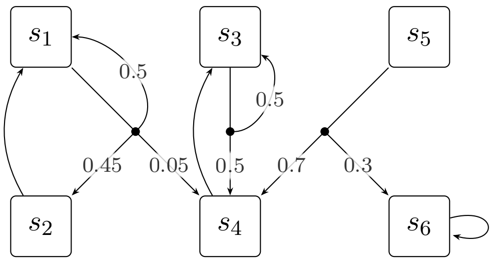


Figure 2.2: Markov chain induced by the example MDP of Figure 2.1 and the strategy π, given by $\pi(s_1) = \{a_1 \mapsto 0.5, b_1 \mapsto 0.5\}$, $\pi(s_3) = \{a_3 \mapsto 1\}$ (omitting states with only one action). We again omit the probability label for edges with probability 1.

2.1.1 Paths & Strategies

An *infinite path* ρ in a Markov chain is an infinite sequence $\rho = s_1 s_2 \cdots \in S^\omega$, such that for every $i \in \mathbb{N}$ we have that $\delta(s_i, s_{i+1}) > 0$. A *finite path* (or *history*) $\varrho = s_1 s_2 \dots s_n \in S^\star$ is a non-empty, finite prefix of an infinite path of length $|\varrho| = n$, ending in some state s_n, denoted by $last(\varrho)$. Additionally, we define $|\rho| = \infty$ for infinite paths ρ. We use $\rho(i)$ and $\varrho(i)$ to refer to the i-th state s_i in a given (in)finite path. A state s *occurs* in an (in)finite path ρ, denoted by $s \in \rho$, if there exists an $i \in [|\rho|]$ such that $s = \rho(i)$. We denote the set of all finite (infinite) paths of an Markov chain M by $\mathsf{FPaths}_\mathsf{M}$ ($\mathsf{Paths}_\mathsf{M}$). Observe that in general $\mathsf{FPaths}_\mathsf{M}$ and $\mathsf{Paths}_\mathsf{M}$ are proper subsets of $S^\star$ and S^ω, respectively, as we imposed additional constraints. Moreover, note that $\mathsf{FPaths}_\mathsf{M}$ is countable in general, whereas $\mathsf{Paths}_\mathsf{M}$ is uncountable.

Similarly, an *infinite path* in an MDP is an infinite sequence $\rho = s_1 a_1 s_2 a_2 \cdots \in (S \times Av)^\omega$ such that for every $i \in \mathbb{N}$ we have $a_i \in Av(s_i)$ and $\Delta(s_i, a_i, s_{i+1}) > 0$, setting the length $|\rho| = \infty$. *Finite paths* ϱ and $last(\varrho)$ are defined analogously as elements of $(S \times Av)^\star \times S$ and the respective last state. Again, $\rho(i)$ and $\varrho(i)$ refer to the i-th state in an (in)finite path with an analogous definition of a state occurring, $|\varrho|$ denotes the length of a finite path, and we refer to the set of (in)finite paths of an MDP $\mathcal{M}$ by $\mathsf{FPaths}_\mathcal{M}$ ($\mathsf{Paths}_\mathcal{M}$). As above, $\mathsf{FPaths}_\mathcal{M}$ is countable and $\mathsf{Paths}_\mathcal{M}$ uncountable.

A Markov chain together with a state $s \in S$ naturally induces a unique probability measure $\mathsf{Pr}_{\mathsf{M},s}$ over infinite paths [BK08, Section 10.1]. For MDP, we first need to eliminate the non-determinism in order to obtain such a probability measure. This is achieved by *strategies* (also called *policy*, *controller*, *adversary*, or *scheduler*).

Definition 3. A strategy on an MDP $\mathcal{M} = (S, Act, Av, \Delta)$ is a function $\pi : \mathsf{FPaths}_\mathcal{M} \to \mathcal{D}(Act)$, such that $\operatorname{supp}(\pi(\varrho)) \subseteq Av(last(\varrho))$ for all $\varrho \in \mathsf{FPaths}_\mathcal{M}$.

Intuitively, a strategy is a 'recipe' describing which step to take in the current state, given the evolution of the system so far. Note that the strategy may yield a distribution on the actions to be taken next.

A strategy π is called *memoryless* (or *stationary*) if it only depends on $last(\varrho)$ for all finite paths ϱ and we identify it with $\pi : S \to \mathcal{D}(Act)$. Similarly, it is called *deterministic*, if it always yields a Dirac distribution, i.e. picks a single action, and we identify it with

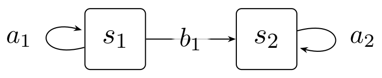


Figure 2.3: Example MDP where no optimal strategy exists.

$\pi : \mathsf{FPaths}_{\mathcal{M}} \to Act$. Together, *memoryless deterministic* strategies can be treated as functions $\pi : S \to Act$ mapping each state to an action. We write $\Pi_{\mathcal{M}}$ to denote the set of all strategies of an MDP $\mathcal{M}$, $\Pi^{\mathsf{M}}_{\mathcal{M}}$ for the memoryless strategies, and $\Pi^{\mathsf{MD}}_{\mathcal{M}}$ for all memoryless deterministic strategies. Observe that there are only finitely many such memoryless deterministic strategies, i.e. $|\Pi^{\mathsf{MD}}_{\mathcal{M}}| < \infty$.

Fixing any strategy π induces a Markov chain $\mathcal{M}^\pi = (\mathsf{FPaths}_{\mathcal{M}}, \delta^\pi)$, where for some state $\varrho = s_1 a_1 \dots s_n \in \mathsf{FPaths}_{\mathcal{M}}$, appropriate action $a_{n+1} \in Av(s_n)$ and successor state $s_{n+1} \in \operatorname{supp}(\Delta(s_n, a_{n+1}))$ the successor distribution is defined as $\delta^\pi(\varrho, \varrho a_{n+1} s_{n+1}) = \pi(\varrho, a_{n+1}) \cdot \Delta(s, a_{n+1}, s_{n+1})$ (see [BK08, Definition 10.92]). In particular, for any MDP $\mathcal{M}$, strategy $\pi \in \Pi_{\mathcal{M}}$, and state s, we obtain a measure over paths[(1)] $\mathsf{Pr}_{\mathcal{M}^\pi, s}$, which we refer to as $\mathsf{Pr}^\pi_{\mathcal{M},s}$. Observe that all these measures operate on the same probability space, namely the set of all infinite paths $\mathsf{Paths}_{\mathcal{M}}$. For a memoryless strategy $\pi \in \Pi^{\mathsf{M}}_{\mathcal{M}}$, we can identify $\mathcal{M}^\pi$ with a Markov chain over the states of $\mathcal{M}$; see Figure 2.2 for an example. We direct the interested reader to [Put94, Section 2.1.6] for further details.

Since we obtain a measure on paths for any strategy π, we can define the maximal probability of a measurable event $A \subseteq \mathsf{Paths}_{\mathcal{M}}$ starting from state $\hat{s}$ by

$$\mathsf{Pr}^{\sup}_{\mathcal{M},\hat{s}}[A] \coloneqq \sup_{\pi \in \Pi_{\mathcal{M}}} \mathsf{Pr}^\pi_{\mathcal{M},\hat{s}}[A]. \tag{2.1}$$

Similarly, given a measurable function $f : \mathsf{Paths}_{\mathcal{M}} \to \mathbb{R}$, we can define the expectation of f under strategy π starting from state $\hat{s}$ by

$$\mathbb{E}^\pi_{\mathcal{M},\hat{s}}[f] \coloneqq \int_{\rho \in \mathsf{Paths}_{\mathcal{M}}} f(\rho) \, d\mathsf{Pr}^\pi_{\mathcal{M},\hat{s}}(\rho)$$

and consequently define the maximal expectation by

$$\mathbb{E}^{\sup}_{\mathcal{M},\hat{s}}[f] \coloneqq \sup_{\pi \in \Pi_{\mathcal{M}}} \mathbb{E}^\pi_{\mathcal{M},\hat{s}}[f]. \tag{2.2}$$

This generalizes the probability measure from Equation (2.1), as $\mathsf{Pr}^{\sup}_{\mathcal{M},\hat{s}}[A] = \mathbb{E}^{\sup}_{\mathcal{M},\hat{s}}[\chi_A]$.

We mention that the supremum indeed is necessary in general—for some functions f, no optimal witness strategy exists. For example, consider the MDP in Figure 2.3 and suppose that f rewards staying in state s_1, but this reward is only collected if the system eventually moves to s_2. Formally, define $f(\rho) = 1 - 1/(|\{i \mid \rho(i) = s_1\}|)$ if there exists an i_0 such that $\rho(i_0) = s_2$ and 0 otherwise. Then, the supremum in Equation (2.2) equals 1, since we can wait arbitrarily long in s_1. However, any strategy eventually has to move to s_2 after a finite number of steps, and any non-zero probability of stopping early gives an overall expectation of strictly less than 1. Note that if f were defined as

(1) Technically, this measure operates on infinite sequences of finite paths, as each state of $\mathcal{M}^\pi$ is a finite path. But, this measure can easily be projected directly on finite paths.

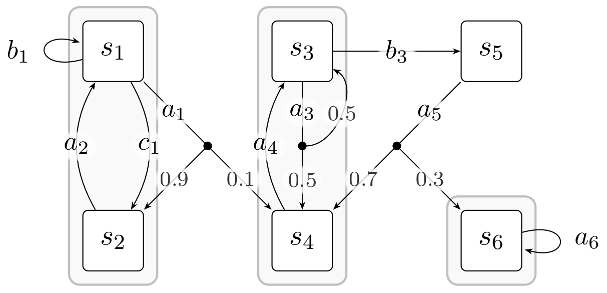


Figure 2.4: Illustration of maximal end components on the MDP from Figure 2.1. Each MEC is shaded. Formally, the set of MECs is given by $(\{s_1, s_2\}, \{b_1, c_1, a_2\})$, $(\{s_3, s_4\}, \{a_3, a_4\})$, and $(\{s_6\}, \{a_6\})$. Note that $(\{s_1\}, \{b_1\})$ is an end component, however it is not inclusion maximal. Moreover, s_5 does not belong to any end component. Intuitively, from s_6 we cannot get back to s_5 and there always is a non-zero chance of moving to s_6 whenever we get to s_5.

$f(\rho) = |\{i \mid \rho(i) = s_1\}|$ there exists an optimal strategy which, for example, remains in state s_1 for 4^n steps with probability 2^{-n}, achieving infinite expected reward.

2.1.2 Strongly Connected Components and End Components

A non-empty set of states $C \subseteq S$ in a Markov chain is *strongly connected* if for every pair $s, s' \in C$ there is a non-empty finite path from s to s'. Such a set C is a *strongly connected component* (SCC) if it is inclusion maximal, i.e. there exists no strongly connected C' with $C \subsetneq C'$. Clearly, SCCs are disjoint and each state belongs to at most one SCC. An SCC is called *bottom strongly connected component* (BSCC) if additionally no path leads out of it, i.e. for all $s \in C, s' \in S \setminus C$ we have $\delta(s, s') = 0$. The set of SCCs and BSCCs in an MC M is denoted by $\mathrm{SCC}(\mathsf{M})$ and $\mathrm{BSCC}(\mathsf{M})$, respectively.

The concept of SCCs is generalized to MDPs by so called *(maximal) end components* [De 97]. Intuitively, an end component is a set of states where the system *can* remain indefinitely, using an appropriate strategy.

Definition 4. Let $\mathcal{M} = (S, Act, Av, \Delta)$ be an MDP. A pair (R, B), where $\emptyset \neq R \subseteq S$ and $\emptyset \neq B \subseteq \bigcup_{s \in R} Av(s)$, is an *end component* of an MDP $\mathcal{M}$ if

1. for all $s \in R, a \in B \cap Av(s)$ we have $\mathrm{supp}(\Delta(s, a)) \subseteq R$, and
2. for all $s, s' \in R$ there is a finite path $\varrho = s a_0 \dots a_n s' \in \mathsf{FPaths}_{\mathcal{M}} \cap (R \times B)^\star \times R$, i.e. the path stays inside R and only uses actions in B.

An end component (R, B) is a *maximal end component (MEC)* if there is no other end component (R', B') such that $R \subseteq R'$ and $B \subseteq B'$.

By abuse of notation, we identify an end component with the respective set of states, e.g., $s \in E = (R, B)$ means $s \in R$. Observe that given two ECs (R_1, B_1) and (R_2, B_2) with $R_1 \cap R_2 \neq \emptyset$, their union $(R_1 \cup R_2, B_1 \cup B_2)$ also is an EC. Consequently, each state belongs to at most one MEC. Again, a MEC is *bottom* if there are no outgoing transitions. The set of ECs of an MDP $\mathcal{M}$ is denoted by $\mathrm{EC}(\mathcal{M})$, the set of MECs by $\mathrm{MEC}(\mathcal{M})$. See Figure 2.4 for an illustration of MECs.

Remark 1. For a Markov chain M, the computation of $\mathrm{SCC}(\mathsf{M})$, $\mathrm{BSCC}(\mathsf{M})$, and a topological ordering of the SCCs can be achieved in linear time w.r.t. the number of states and transitions by, e.g., Tarjan's algorithm [Tar72]. Similarly, the MEC decomposition of an MDP can be computed in polynomial time [CY95]. For improved algorithms on general MDP and various special cases see [CH11; CH12; CH14].

These components fully capture the limit behaviour of any Markov chain and decision process, respectively. Intuitively, both of the following statements show that with probability one a run of such systems eventually forever remains inside one single BSCC or MEC, respectively.[(2)]

Lemma 1. *For any MC M and state s, we have that*

$$\mathsf{Pr}_{\mathsf{M},s}[\{\rho \mid \exists R_i \in \mathrm{BSCC}(\mathsf{M}).\exists n_0 \in \mathbb{N}.\forall n > n_0.\rho(n) \in R_i\}] = 1.$$

Proof. Follows from [BK08, Theorem 10.27]. □

Lemma 2. *For any MDP $\mathcal{M}$, state s, and strategy π, we have that*

$$\mathsf{Pr}^{\pi}_{\mathcal{M},s}[\{\rho \mid \exists (R_i, B_i) \in \mathrm{MEC}(\mathcal{M}).\exists n_0 \in \mathbb{N}.\forall n > n_0.\rho(n) \in R_i\}] = 1.$$

Proof. Follows from [De 97, Theorem 3.2]. □

Moreover, in a BSCC or MEC we (can) visit each state thereof with probability one.

Lemma 3. *For any MC M, BSCC $R \in \mathrm{BSCC}(\mathsf{M})$ and states $s, s' \in R$, we have that* $\mathsf{Pr}_{\mathsf{M},s}[\{\rho \mid s' \in \rho\}] = 1$.

Proof. Follows from [BK08, Corollary 10.34]. □

Lemma 4. *For any MDP $\mathcal{M}$ and MEC $(R, B) \in \mathrm{MEC}(\mathsf{M})$ there exists a strategy $\pi \in \Pi^{\mathsf{M}}_{\mathcal{M}}$ such that for all states $s, s' \in R$, we have* $\mathsf{Pr}^{\pi}_{\mathcal{M},s}[\{\rho \mid s' \in \rho\}] = 1$.

Proof. Follows from [BK08, Lemma 10.119]. □

So, when we consider 'reaching' a particular target, states in the same component are equivalent: Intuitively, if a state s has a particular way of reaching a target, any other state in the same component can first move to s by virtue of Lemma 4 and then follow its strategy. With this insight, we can view components as equivalence classes and apply a *quotient operation.* Defined in [De 97, Algorithm 6.1], the *MEC quotient* intuitively replaces each MEC by a single representative state, eliminating all internal behaviour. Since the formal definition is rather technical, we instead show the MEC quotient of our running example in Figure 2.5.

[(2)] The measurability of the sets in the following lemmas is well known, proofs can be found in, e.g., [BK08, Chapter 10].

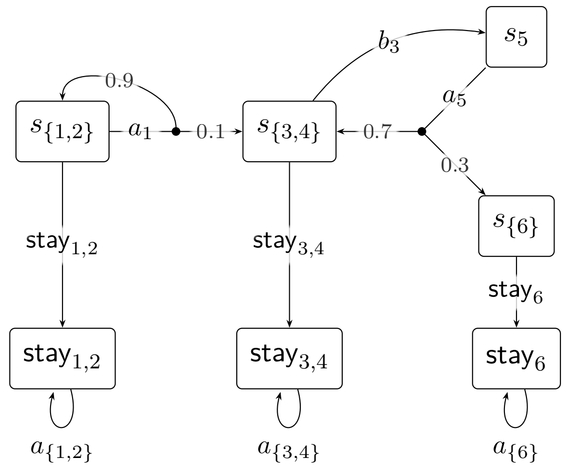


Figure 2.5: Illustration of the MEC quotient of the MDP from Figure 2.1. The internal behaviour of the MECs is removed and a special stay state and action is added, representing that the system remains inside the now collapsed MEC.

2.2 Objectives

We have defined Markov chains and decision processes together with their dynamics. As hinted at in the introduction, we are still missing a way to formalize questions such as 'Is the given system safe?' or, in the case of MDP, 'Can the system be controlled in a safe manner?'. Hence, we need to define a formalism to describe what 'safe' means, which is called *objective*. There are many ways to introduce such objectives. We follow a quite general view, where objectives are essentially defined by three components, namely (i) *path performance*, (ii) *performance aggregation*, and (iii) *optimization*. Intuitively, (i) assigns a performance metric to each path, for example a real number or a vector of reals. For a fixed strategy, this essentially gives us a distribution over the respective value space. Then, (ii) aggregates this distribution into a single value, e.g., by computing an expectation. Finally, (iii) decides which performance values are 'better'. For example, we may be interested in the maximum or minimum possible performance, a particular trade-off, or a sweet-spot. We explain these concepts with several commonly used objectives in the following.

This framework does not cover all possible objectives. In particular, the performance of a single path is considered to be independent of other potential paths of the system. This is not the case with, for example, *Hyperproperties*, where among others information flow or non-interference are considered [Mül20].

We first explain several objectives relevant to this work and then present common solution techniques later in Section 2.3, since the classical approaches to many objectives share the same fundamental ideas. We use the following notational templates:

- $\mathcal{V}_{\mathsf{perf}}(\rho)$: The path performance of a given path ρ.
- $\mathcal{V}^{\pi}_{\mathsf{perf},\mathsf{aggr}}(s)$: The aggregated performance when starting in state s and following

strategy π. For example, with a real-valued performance $\mathcal{V}_{\mathsf{perf}}$ and $\mathsf{aggr} = \mathbb{E}$, we define $\mathcal{V}^{\pi}_{\mathsf{perf},\mathbb{E}}(s) \coloneqq \mathbb{E}^{\pi}_{\mathcal{M},s}[\mathcal{V}_{\mathsf{perf}}]$ as the expected performance under π.

- $\mathcal{V}^{\mathsf{opt}}_{\mathsf{perf},\mathsf{aggr}}(s)$: The optimal aggregated performance when starting in state s. As an example, for a real-valued aggregation function we define $\mathcal{V}^{\max}_{\mathsf{perf},\mathsf{aggr}}(s) \coloneqq \sup_{\pi \in \Pi_{\mathcal{M}}} \mathcal{V}^{\pi}_{\mathsf{perf},\mathsf{aggr}}(s)$ as the supremum over all strategies.

2.2.1 Reachability

One of the simplest, yet surprisingly expressive objectives is given by *reachability*. Essentially, the goal is to reach a given set of states, and typically we are interested in maximizing this probability; minimization corresponds to avoiding the given region and is often called *safety* instead.

Formally, fix an MDP $\mathcal{M} = (S, Act, Av, \Delta)$ and a set of *target states* $T \subseteq S$. Then, *(unbounded) reachability* $\Diamond T = \{\rho \in \mathsf{Paths}_{\mathcal{M}} \mid \exists i \in \mathbb{N}.\ \rho(i) \in T\}$ are all infinite paths which eventually reach the target set T. The sets of paths produced by $\Diamond$ are measurable for any MDP and target set [BK08, Section 10.1.1].(3) Sometimes, *bounded reachability* $\Diamond^{\leq k} T$ is considered, where the goal is to reach T within a given step bound of k steps. However, we mainly focus on unbounded properties in this work and omit precise treatment of the bounded variants.

Note that for a target set T, both $\Diamond \overline{T}$ and $\overline{\Diamond T}$ are well-defined, however they refer to two different concepts. The former denotes the set of all paths reaching a state not in T, whereas the latter is the set of all paths which never reach T.

To fit reachability into our framework, we define the path performance by $\mathcal{V}_{\Diamond T}(\varrho) \coloneqq \chi_{\Diamond T}(\varrho)$, i.e. a path is assigned a value of 1 if any of its states is contained in the target set T and 0 otherwise. Since $\Diamond T$ is measurable, $\mathcal{V}_{\Diamond T}$ is measurable, too, and the expectation of $\mathcal{V}_{\Diamond T}$ is well defined for any given strategy π. Then

$$\mathcal{V}^{\pi}_{\Diamond T,\mathbb{E}}(s) = \mathbb{E}^{\pi}_{\mathcal{M},s}[\mathcal{V}_{\Diamond T}] = \mathsf{Pr}^{\pi}_{\mathcal{M},s}[\Diamond T].$$

This constitutes the performance aggregation—we are interested in expected reachability. In this case, not too many other aggregation schemes make sense, since reachability is binary—a path either reaches the goal or it does not—and thus the expectation fully describes the performance of a strategy. More intricate schemes appear later on.

To now obtain the overall objective, namely the maximal probability of reaching a target set T starting in a given *initial state* $\hat{s}$, we write

$$\mathcal{V}^{\max}_{\Diamond T,\mathbb{E}}(\hat{s}) = \sup\nolimits_{\pi \in \Pi_{\mathcal{M}}} \mathcal{V}^{\pi}_{\Diamond T,\mathbb{E}}(\hat{s}).$$

2.2.2 Linear Temporal Logic

Instead of asking to maximize the probability of reaching a single target set, we may be interested in adding additional constraints, e.g., avoiding a 'bad' region on the way, or add several goal regions. An elegant formalization of such questions is given by *Linear*

(3) Recall that we defined MDP to have finite state and action sets.

Temporal Logic (LTL) [Pnu77]. LTL provides a richer framework to assign 'good' or 'bad' labels to a given run. As such, it only differs from reachability by providing a more complex path performance function, which however still only yields 1 or 0.

We briefly define LTL and direct the interested reader to related work, e.g., [BK08, Chapter 5] or [For+11, Section 7.2]. Let AP be a finite, non-empty set of *atomic propositions.* Such propositions could, for example, describe 'variable x is larger than 5' or 'the robot is damaged'. An LTL formula then is given by the following syntax

$$\phi ::= a \mid \neg\phi \mid \phi \wedge \phi \mid \mathbf{X}\,\phi \mid \phi\,\mathbf{U}\,\phi$$

where $a \in \mathsf{AP}$. We define $\bot = a \wedge \neg a$ for some $a \in \mathsf{AP}$, $\top = \neg\bot$, and $\phi \vee \psi = \neg(\neg\phi \wedge \neg\psi)$. An LTL formula is evaluated over (infinite) sequences of words, i.e. elements of $\mathcal{P}(AP)^\omega$. Thus, each 'letter' of the word is a set of atomic propositions. Intuitively, each such letter describes the set of propositions which hold at the respective time step. The logical connectives of LTL impose restrictions on the 'current' step, i.e. which atomic propositions hold and which do not. The $\mathbf{X}\,\phi$ operator requires that ϕ holds in the next step, while $\phi\,\mathbf{U}\,\psi$ demands that the formula ϕ holds at every step until eventually ψ holds. Two common derivations are $\mathbf{F}\,\phi \coloneqq \top\,\mathbf{U}\,\phi$, requiring that ϕ eventually holds in the **f**uture, and $\mathbf{G}\,\phi \coloneqq \neg\,\mathbf{F}\,\neg\phi$ requires that ϕ holds **g**lobally at every position.

As an example, $\mathtt{send} \wedge (\mathbf{X}\,\mathbf{G}\,\mathtt{error} \vee \mathbf{F}\,\mathtt{recv})$ requires that we `send` in the first step and either enter an `error` state in the next step and remain there forever or eventually `rec`eive an answer. The word $w = \{\mathtt{send}\}\ \emptyset\ \emptyset\ \{\mathtt{recv}\}\ \emptyset\ \cdots$ satisfies this formula, while $w' = \{\mathtt{send}\}\ \{\mathtt{error}\}\ \{\mathtt{error}\}\ \emptyset^\omega$ does not. For an LTL formula ϕ and word w over the appropriate set of atomic propositions, we write $w \models \phi$ if w satisfies the given formula. See for example [BK08, Section 5.1.2] for a formal definition of $\models$. We write $\mathcal{L}(\phi) \coloneqq \{w \in \mathcal{P}(\mathsf{AP})^\omega \mid w \models \phi\}$ for the *language* of ϕ.

In order to connect LTL to MDP, we further need a labelling function $\mathsf{L} : S \to \mathcal{P}(AP)$, assigning to each state of the system a set of atomic propositions which hold in this state.(4) For example, we could assign the atomic proposition `danger` to all states which are close to a hazard in the grid world of Figure 1.1. This mapping can directly be lifted to paths, i.e. given a path ρ we derive the respective word $\mathsf{L}(\rho) \coloneqq \mathsf{L}(\rho(1))\mathsf{L}(\rho(2))\cdots$. Then, we can pose *quantitative LTL queries*, e.g. 'what is the maximal probability of satisfying the given formula?'. For example, we could require that we never are in `danger` for more than one consecutive step by $\mathbf{G}(\neg\mathtt{danger} \vee \mathbf{X}\,\neg\mathtt{danger})$ in the grid world example. Formally, we write $\mathcal{V}_\phi(\rho) = 1$ if $\mathsf{L}(\rho) \models \phi$, carrying over analogous definitions from reachability. See [BK08, Remark 10.57] for a proof of measurability.

We usually are interested in maximizing the probability of satisfying a given formula ϕ starting from a particular state $\hat{s}$, i.e. $\mathcal{V}^{\max}_{\phi,\mathbb{E}}(\hat{s})$. This computation can be reduced to pre-computations followed by a regular reachability analysis on a modified MDP. We explain this process in more detail in Chapter 4.

(4) The labelling can equivalently be assigned to the *actions* of the MDP without changing the underlying problem significantly. We opt for the state-based approach for simplicity and consistency with subsequent definitions.

2.2.3 Weighted Reachability

Both of the previous objectives are binary, i.e. a single path is either good or bad, without any level in between. However, the notion of performance often is more fine-grained. For example, the grid world could offer various treasures, each of a different value. Treasures of higher values might be more difficult to reach, so it may not be immediately clear which way to follow. To this end, we introduce more complex *reward-based* objectives, where the performance is determined using a reward function.

An immediate extension is given by *weighted reachability.* As with reachability, we are given an MDP $\mathcal{M} = (S, Act, Av, \Delta)$ and a set of *target states* $T \subseteq S$. Additionally, we are given a reward function $\mathsf{rew} : T \to \mathbb{R}$. The performance of a path then is defined as the reward assigned to the first visited state of T, i.e.

$$\mathcal{V}_{\mathsf{rew}\Diamond T}(\rho) \coloneqq \mathsf{rew}(\rho(\min\{i \mid \rho(i) \in T\}))$$

or 0 if no such state exists, i.e. $\{i \mid \rho(i) \in T\} = \emptyset$. Again, we now usually are interested in the maximal expected values, i.e. $\mathcal{V}^{\max}_{\mathsf{rew}\Diamond T,\mathbb{E}}$. However, we could, for example, also be interested in the worst-case behaviour, i.e. the smallest reward obtained under the strategy π, denoted $\mathcal{V}^{\pi}_{\mathsf{rew}\Diamond T,wc}(\hat{s})$. We present a novel aggregation scheme interpolating between worst-case and expectation in Chapter 6.

Interestingly, weighted reachability can be reduced to regular reachability by straightforward preprocessing steps, as we explain in Chapter 3. We introduce this particular objective since it is quite useful for the dealing with the more complex mean payoff objective defined in Section 2.2.5.

2.2.4 Total Reward

Another approach to consider rewards is the *k-step total reward* (or *cumulative reward*). Instead of obtaining the first reward we encounter, a path obtains all rewards of the first k steps. Formally, for an MDP $\mathcal{M} = (S, Act, Av, \Delta)$, a reward function $\mathsf{rew} : S \to \mathbb{R}$, and a step bound k, the k-step total reward of a path ρ is defined as

$$\mathcal{V}_{\mathsf{rew}\text{-}k}(\rho) \coloneqq \sum\nolimits_{i=1}^{k} \mathsf{rew}(\rho(i)).$$

In this thesis, we do not treat the total reward objective directly. Instead, we use it as an auxiliary tool when dealing with the mean payoff objective, defined in the next section. Thus, we omit further details here and direct to, e.g., [For+11, Section 5.2].

2.2.5 Mean Payoff

Weighted reachability only offers a 'one shot' reward, and requires us to define a specific goal. As such it is mainly suitable for finite tasks, in the sense that once the goal is achieved the systems' actions are irrelevant. Similarly, total reward only considers a fixed, finite number of steps. However, in long running applications such as task scheduling, flow control, etc., we are instead interested in the long-run behaviour. For example, in a traffic shaping protocol we usually may ask for the average throughput. Here, there is no single goal state to be reached and the system does not stop after a finite number of steps. Instead, we want to investigate the average behaviour 'at infinity'. Mean payoff provides a powerful mechanism to tackle this problem.

We again are provided with an MDP $\mathcal{M} = (S, Act, Av, \Delta)$ and a reward function $\mathsf{rew} : S \to \mathbb{R}$. The *mean payoff* (also known as *long-run average reward*, *limit-average reward* or *gain*) of a run is defined as the limit of the k-step average rewards, i.e.

$$\mathcal{V}_{\mathsf{rew}\text{-}\mathrm{mp}}(\rho) \coloneqq \liminf_{k\to\infty} \frac{1}{k} \sum\nolimits_{i=1}^{k} \mathsf{rew}(\rho(i)) = \liminf_{k\to\infty} \frac{1}{k}\, \mathcal{V}_{\mathsf{rew}\text{-}k}(\rho).$$

The lim inf is necessary, since the limit itself may not be defined for some runs due to oscillations. As before, we usually are interested in maximizing the expected value, i.e. we want to determine $\mathcal{V}^{\max}_{\mathsf{rew}\text{-}\mathrm{mp},\mathbb{E}}$.

Surprisingly, this problem can again be solved by pre-computations and a reduction to weighted reachability, which in turn can be solved by reachability. We elaborate further on this reduction in Chapter 3. See [Put94, Chapter 8, 9] for an extensive treatment of mean payoff on MDP. Note that we defined the reward function on the set of states for consistency reasons. As with LTL, it is straightforward to extend the problem and its solution approaches to rewards assigned to actions.

2.2.6 Multi-dimensional Objectives

So far, we only considered single-valued objectives, where the performance values can be compared directly. Thus, optimizing the achieved value w.r.t. maximization or minimization is straightforward. However, in reality more complex tasks like 'Maximize the power output subject to given safety constraints' appear. A natural way to formalize such requirements is to combine the presented objectives into *multi-dimensional queries*. For example, to formalize the above task, we can define the path performance as a two-dimensional vector: The first component describes the power output via mean payoff and the second component is given by the LTL formula formalizing the safety constraints. We aggregate both dimensions by expectation and define the optimization as maximizing the first component subject to the second component being at least, for example, 95%. Multi-dimensional problems often lead to an explosion in complexity, both in terms of computation and structure of optimal strategies, see, e.g., [CKK17]. As such, we keep treatment thereof to a minimum and focus on single-dimensional queries. We briefly discuss multi-dimensional questions in Chapter 6.

2.3 Solution Techniques

In this section, we briefly discuss classical solution techniques associated with MDP. We explain each of these techniques by the example of reachability and compare their advantages and disadvantages.

As an overarching distinction, we differentiate between *precise* and *approximate*

$$
\begin{array}{ll}
\multicolumn{2}{c}{\text{minimize} \quad \sum_{s \in S} x_s \quad \text{where}} \\
x_s \geq 0 & \text{for } s \in S \\
x_s = 1 & \text{for } s \in T \\
x_s \geq \sum_{s' \in S} \Delta(s, a, s') \cdot x'_s & \text{for } s \notin T \text{ and } a \in Av(s)
\end{array}
$$

Figure 2.6: Linear program to compute the reachability of a target set T. The probability of reaching T from a state s is given by the respective variable x_s.

solutions. Precise approaches provide, as the name suggests, exact answers (assuming an appropriate implementation). On the other hand, approximations yield an answer which is correct up to a given precision $\varepsilon > 0$. Formally, we say that a value v is *ε-optimal* if $|v - \mathcal{V}| < \varepsilon$, where $\mathcal{V}$ is the value of interest.[5] Whenever we consider approximate approaches, we implicitly assume that a precision $\varepsilon > 0$ is fixed. Typically, this precision is chosen somewhere between 10^{-3} and 10^{-9}. Note that such approaches are required to provably bound the approximation error. In particular, 'convergence in the limit' without further guarantees, e.g., a rate of convergence, does not qualify as an approximative solution.

There are several reasons to consider approximative approaches. For example, computing a precise answer may be intractable or even undecidable. Moreover, some appealing approaches may intrinsically only yield approximative answers, for example our algorithm in Section 3.2. Finally, relaxing the problem in this way allows for fundamental optimizations, as we explain in Chapter 5.

2.3.1 Linear Programming

Linear Programming (LP) is an established problem solving technique with a wide spectrum of applications. LP has strong fundamental connections to MDP; most objectives considered in the literature allow for a natural LP formulation. A linear program essentially is characterized by a linear *objective function* $f : \mathbb{R}^n \to \mathbb{R}$ and a set of *linear inequality constraints* on the n variables, i.e. a matrix $A \in \mathbb{R}^{m \times n}$ and vector $b \in \mathbb{R}^m$. The task is to find the maximal value of $f(x)$, subject to the imposed linear constraints. Formally, we want to compute $\max_{x \in \mathbb{R}^n . Ax \leq b} f(x)$. See [Sch99] for details on linear programming in general. LP is very appealing from a theoretical point of view, since it allows for direct mathematical characterizations and can be solved in polynomial time without any imprecision [Kha79; Kar84]. As such, it is a popular tool to prove polynomial complexity of many problems. We show a standard LP for reachability in Figure 2.6. See [BK08, Theorem 10.105] and [For+11, Section 4.2] for further details.

[5] The expression $|v - \mathcal{V}|$ is also called *absolute* error. For $\mathcal{V} \neq 0$, one can also require the *relative* error to be small, i.e. $|v - \mathcal{V}|/|\mathcal{V}| < \varepsilon$. Then, the allowed deviation from the true value is dependent on its magnitude. We omit treatment of relative error for simplicity.

In practice, LP-based solutions appear to be surprisingly inefficient [HM14; Brá+15; Ash+17]. For example, the LP of Figure 2.6 tends to be feasible up to tens of thousands of states with a timeout of one hour. Even comparably abstract real-world models have millions or even billions of states; an analysis by LP seems far out of reach. The approaches presented in the following may still succeed within reasonable time.

Several alternative approaches have been proposed. We discuss two prominent ideas in the following, namely value iteration and strategy iteration. While they both usually have an exponential worst-case runtime, they tend to perform very well on real-world models. In particular they typically outperform LP approaches by a huge margin.

2.3.2 Value Iteration

Value iteration (VI) [Bel66] is a simple yet surprisingly efficient and extendable approach to solve a broad variety of problems. At its heart, VI relies, as the name suggests, on iteratively applying an operation to a value vector. This operation often is called 'Bellman backup' or 'Bellman update'. The shape of this update naturally depends on the domain and objective in question. It usually is derived from a fixed-point characterization thereof, and thus VI often can be viewed as fixed point iteration.

For reachability, the optimal value $\mathcal{V}^{\max}_{\lozenge T,\mathbb{E}}$ is a fixed point of [For+11, Section 4.2]

$$f(s) = \begin{cases} 1 & \text{if } s \in T, \\ \max_{a \in Av(s)} \Delta(s,a)\langle f \rangle & \text{otherwise.} \end{cases} \tag{2.3}$$

Note that $\mathcal{V}^{\max}_{\lozenge T,\mathbb{E}}$ is not the only solution. For example $f(s) = 1$ for all $s \in S$ satisfies the equation, too. However, $\mathcal{V}^{\max}_{\lozenge T,\mathbb{E}}$ is the point-wise *smallest* fixed point [Put94, Theorem 7.2.3][(6)]. Thus, the VI approach starts from an initial value vector $v_0[s] = 0$, certainly less than or equal to the true value, and we apply the iteration

$$v_{k+1}[s] = \begin{cases} 1 & \text{if } s \in T, \\ \max_{a \in Av(s)} \Delta(s,a)\langle v_k \rangle & \text{otherwise.} \end{cases} \tag{2.4}$$

This iteration is monotone and converges to the true value in the limit from below. Formally, we have for all states s that (i) $\lim_{k\to\infty} v_k[s] = \mathcal{V}^{\max}_{\lozenge T,\mathbb{E}}(s)$ and (ii) $v_k[s] \leq v_{k+1}[s] \leq \mathcal{V}^{\max}_{\lozenge T,\mathbb{E}}(s)$ for all iterations k [Put94, Theorem 7.2.12][(6)]. Note that in this case, i.e. (unbounded) reachability, VI intrinsically can only yield approximate solutions. However, this is not always the case for VI. For example, bounded reachability for k steps can be computed precisely by applying the above iteration exactly k times.

There are MDP where convergence up to a given precision takes exponential time [HM14, Theorem 17], but in practice VI often is much faster than methods based on linear programming. Even a naive implementation of VI, requiring a few dozen lines of code, usually yields a result very close to the optimum much faster than LP.

(6) Reachability is a special case of *expected total reward*, obtained by assigning a one-time reward of 1 to each target state.

We are still missing an important ingredient for a practical implementation of VI. So far, we only know that *eventually* the value vector v_k is close to the optimum, but we do not have a concrete (practical) bound. In other words, we need to know when the current values actually are close to the true value and we can stop the iteration. In some applications, a sensible bound on the number of iterations is known a-priori, e.g., bounded reachability. For (unbounded) reachability, this a-priori bound however is exponential. In general, we are interested in finding a so called *stopping criterion*, i.e. a way to determine or estimate how close we are to the true value based on the computation so far. Surprisingly, even for reachability such a stopping criterion was not known until recently; popular model checking implementations resorted to a best-effort solution without any guarantees [HM14, Section 3.1], [Bai+17]. In [Brá+14; HM14], a stopping criterion for reachability was independently discovered by additionally computing converging *upper* bounds. The difference between upper and lower bounds then gives a straightforward stopping criterion—once the difference between upper and lower bound in the initial state is smaller than ε, we can stop the iteration.

A big advantage of VI is its simplicity and extendability. For example, the iteration can be applied *asynchronously*. Here, we do not update the values of all states simultaneously. Instead, we apply the operator of Equation (2.4) to a subset of states. We can thus focus the computational effort on important areas of the system instead of applying the iteration globally while maintaining convergence guarantees. This is a key ingredient of the partial exploration approaches discussed in Chapters 3 and 5. In the context of fixed-point iteration, this is also called *chaotic iteration* [Cou78].

2.3.3 Strategy Iteration

Another elegant approach is described by *strategy iteration* (SI) (or *policy iteration*, *strategy improvement*). Instead of iterating values as in VI, each iteration evaluates and improves a strategy. Abstractly, we (i) fix a strategy, (ii) evaluate it, and (iii) improve the strategy greedily based on the evaluation, repeating until no improvement is possible. An in-depth theoretical discussion of strategy iteration for MDP can be found in, e.g., [Put94]. Here, we provide a brief overview by discussing SI for reachability.

An important observation is that memoryless deterministic strategies are optimal for reachability, i.e. there always exists a strategy $\pi^* \in \Pi_{\mathcal{M}}^{\mathsf{MD}}$ such that $\mathcal{V}_{\Diamond T,\mathbb{E}}^{\max}(s) = \mathcal{V}_{\Diamond T,\mathbb{E}}^{\pi^*}$ [Put94, Theorem 7.1.9][(6)]. In other words, when searching an optimal strategy, it is sufficient to restrict the search to $\Pi_{\mathcal{M}}^{\mathsf{MD}}$, which is finite. This property is not strictly necessary for SI to be applicable, however it significantly eases correctness and convergence arguments as well as complexity analysis.

When applying SI to reachability, we evaluate the performance of the strategy $\pi \in \Pi_{\mathcal{M}}^{\mathsf{MD}}$ by directly computing its performance $\mathcal{V}_{\Diamond T,\mathbb{E}}^{\pi}$. This value also satisfies a fixed point equation very similar to Equation (2.3). In particular, for a strategy $\pi \in \Pi_{\mathcal{M}}^{\mathsf{MD}}$ we have that [BK08, Theorem 10.19]

$$\mathcal{V}_{\Diamond T,\mathbb{E}}^{\pi}(s) = \begin{cases} 1 & \text{if } s \in T, \\ \Delta(s, \pi(s))\langle \mathcal{V}_{\Diamond T,\mathbb{E}}^{\pi} \rangle & \text{otherwise.} \end{cases}$$

Since the non-determinism of the MDP is eliminated by the strategy, we obtain a simple linear equation system. This equation system does not have a unique solution in general, see [BK08, Remark 10.18]. However, this problem is remedied by only considering states which have a non-zero probability of reaching the target set. These states can be identified by a simple graph analysis. Together, we can compute $\mathcal{V}^{\pi}_{\Diamond T,\mathbb{E}}$ using standard approaches. This provides Step (ii) of the strategy iteration template. Now, we need to improve the strategy based on the achieved value $\mathcal{V}^{\pi}_{\Diamond T,\mathbb{E}}$. This often is the crux of applying SI to a particular problem—both correctness and termination of the process clearly depend on the improvement step.

In the case of reachability, it turns out that the improvement step is elegantly simple. Let $\pi \in \Pi^{\mathsf{MD}}_{\mathcal{M}}$ be a strategy. For each state, we simply choose an action which maximizes the value $\mathcal{V}^{\pi}_{\Diamond T,\mathbb{E}}$ over its successors, i.e.

$$\pi'(s) \in \arg\max_{a \in Av(s)} \Delta(s,a)\langle \mathcal{V}^{\pi}_{\Diamond T,\mathbb{E}} \rangle,$$

choosing $\pi'(s) = \pi(s)$ where possible. If $\pi' = \pi$ then clearly π' satisfies the optimality condition Equation (2.3) and thus is optimal.

In contrast to VI, strategy iteration yields exact values while still providing good performance in practice compared to LP. Another interesting by-product of SI is that in each iteration step it trivially yields a witness strategy exactly achieving the current computed value. As we argue in Section 3.3, this can also be considered a weakness—we do not always need precise values in order to improve the current strategy.

3 Efficient Analysis of Mean Payoff

In this chapter, we discuss analysis of mean payoff objectives on Markov decision processes. Informally, we are given an MDP, equipped with a reward function, together with an initial state. Based on this input, the task is to compute (or approximate) the maximal achievable mean payoff.

Problem Statement

Let $\mathcal{M} = (S, Act, Av, \Delta)$ be an MDP, $\hat{s} \in S$ an initial state, and $\mathsf{rew} : S \to \mathbb{R}$ a reward function. Compute the maximal achievable mean payoff

$$\mathcal{V}^{\max}_{\mathsf{rew}\text{-mp},\mathbb{E}}(\hat{s}) = \sup\nolimits_{\pi \in \Pi_{\mathcal{M}}} \mathcal{V}^{\pi}_{\mathsf{rew}\text{-mp},\mathbb{E}}(\hat{s}).$$

We first outline the problem together with classical solution approaches and their limitations in Section 3.1. Then, Section 3.2 presents our value iteration based approach, which we furthermore combine with the partial exploration ideas of [Brá+14]. This allows our VI method to focus on 'important' areas of the system as well as discarding parts which cannot be part of an optimal solution, saving computational resources. In Section 3.3, we show a different approach, based on strategy iteration. In contrast to VI, this offers precise results, however at the expense of additional computational effort. We devise several techniques to speed up the computation, even making it competitive with our VI approach on some models. We conclude with a summary and directions for future work in Section 3.4.

3.1 Background

Mean payoff has already been extensively studied from a theoretical point of view. We direct the interested reader to [Put94, Chapter 9] for further background and provide an overview of the results here.

An important difference of mean payoff to other objectives discussed in this thesis is its 'prefix-independence'—the rewards obtained on any finite prefix of a run are irrelevant for its performance. In other words, mean payoff exclusively depends on the behaviour 'at infinity', completely neglecting the transient behaviour. This is in direct contrast to reachability, where the target is always either reached on a finite prefix or not at all. Nevertheless, memoryless deterministic strategies are optimal for mean payoff, i.e. there always exists a strategy $\pi^* \in \Pi^{\mathsf{MD}}_{\mathcal{M}}$ such that $\mathcal{V}^{\max}_{\mathsf{rew\text{-}mp},\mathbb{E}}(s) = \mathcal{V}^{\pi^*}_{\mathsf{rew\text{-}mp},\mathbb{E}}(s)$ for all states $s \in S$ [Put94, Theorem 9.1.8]. To identify such strategies, we first need to introduce some additional ideas.

3.1.1 Gain & Bias

An important notion for the analysis of mean payoff is the concept of *gain* and *bias*. Intuitively, the gain of a state refers to the optimal mean payoff obtained when starting in this state. The bias denotes the total expected deviation from the gain until the systems' behaviour 'stabilizes' from transient to infinite. For example, suppose we would obtain the rewards 11 7 5 5 5 5 $\cdots$. Then, the gain is 5 and the bias equals 8, as we obtain 6 and 2 more than the gain before stabilizing. This intuition gives rise to the *optimality equations* for the gain g and bias b for all states $s \in S$ [Put94, Section 9.1.1]

$$\begin{aligned} g(s) &= \max\nolimits_{a \in Av(s)} \Delta(s,a)\langle g \rangle \\ b(s) &= \max\nolimits_{a \in Av_g(s)} \Delta(s,a)\langle b \rangle + \mathsf{rew}(s) - g(s), \end{aligned} \tag{3.1}$$

where

$$Av_g(s) \coloneqq \arg\max\nolimits_{a \in Av(s)} \Delta(s,a)\langle g \rangle$$

are the actions of s which maximize the gain, i.e. the witnesses for $g(s)$ in the first line of Equation (3.1). We provide further intuition of gain and bias in the following discussion of strategy iteration for mean payoff. The above equation system has no unique solution in general, however (i) the gain component of any two solutions is equal, (ii) there always exists a solution, and (iii) the gain of this solution corresponds to the optimal mean payoff [Put94, Section 9.1.2, 9.1.3]. Thus, our task now is to solve this equation system. With some effort, we can reformulate this sytem as a linear program for mean payoff [Put94, Section 9.3]. However, we do not discuss this LP in detail here, since it turns out to be impractical for real-world models [Ash+17]. Instead, we focus on dynamic programming approaches.

3.1.2 Strategy Iteration

As with reachability, we can immediately derive an equation system for a fixed strategy by replacing the maximization in Equation (3.1) by the strategy's concrete choices as follows. Let $\pi \in \Pi^{\mathsf{MD}}_{\mathcal{M}}$ be a memoryless deterministic strategy. The mean payoff obtained by π can be computed by solving

$$\begin{aligned} g_\pi(s) &= \Delta(s,\pi(s))\langle g_\pi \rangle \\ b_\pi(s) &= \Delta(s,\pi(s))\langle b_\pi \rangle + \mathsf{rew}(s) - g(s), \end{aligned} \tag{3.2}$$

for all states $s \in S$. The strategy π is optimal if a solution (g_π, b_π) to the above equation satisfies Equation (3.1). Otherwise, we can improve the strategy by optimizing according to the optimality equation as follows. Suppose (g_π, b_π) is a solution to Equation (3.2) which does not satisfy the Equation (3.1). Then, in each state s where $\pi(s) \notin Av_{g_\pi}(s)$,

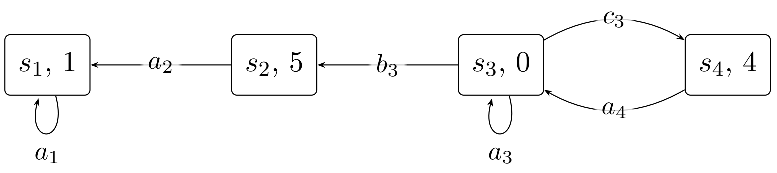


Figure 3.1: Example MDP to illustrate the strategy improvement algorithm and the concepts of gain and bias of mean payoff. The reward assigned to each state by the reward function rew is written next to each state.

i.e. there is an action in state s which provides strictly better gain, we update $\pi(s)$ to any gain-optimal action from $Av_{g_\pi}(s)$. If no gain improvement is possible in any state, but we still do not satisfy the optimality condition, we instead need to improve the bias. As expected, we improve the strategy by changing it to any bias-optimal action, i.e. any action in $\arg\max_{a \in Av_{g_\pi}(s)} \Delta(s, \pi(s))\langle b_\pi \rangle + \mathsf{rew}(s)$.[(1)] This yields a correct strategy improvement algorithm [Put94, Section 9.2], which we illustrate in the following and augment further in Section 3.3.

Consider the example MDP in Figure 3.1. We explain a sample execution of SI on this MDP. This simultaneously provides some insight in the notion of gain and bias. In the example, the only state which allows for a choice is s_3. We use π_a, π_b, and π_c to denote the memoryless deterministic strategy with choice a_3, b_3, and c_3 in s_3, respectively. The optimal strategy is π_c, achieving a mean payoff of 1 for states s_1 and s_2, and a mean payoff of 2 in s_3 and s_4, while π_a and π_b only achieve 0 and 1 in s_4, respectively. Suppose the algorithm starts with π_a. Solving Equation (3.2) gives us $g_{\pi_a} = (1, 1, 0, 0)$ and we can directly improve the strategy based on gain by switching to π_b. Now, we obtain $g_{\pi_b} = (1, 1, 1, 1)$, satisfying the gain optimality condition. However, π_b is not optimal, and indeed we can improve it based on the bias. We get $b_{\pi_b} = (0, 4, 3, 6)$. While both actions b_3 and c_3 offer the same gain, c_3 yields a higher bias, namely 6 instead of 4. By switching from b_3 to c_3, we are not decreasing our gain (since both actions are gain-optimal) and obtain a higher reward at least once, which, intuitively, cannot hurt. And indeed, π_c even obtains a higher gain, namely $g_{\pi_c} = (1, 1, 2, 2)$.

Observe that $b_{\pi_c} = (0, 4, -2, 2)$. In particular, we have $b_{\pi_c}(s_2) > b_{\pi_c}(s_4)$, however s_2 is not gain optimal, i.e. $s_2 \notin Av_{g_{\pi_c}}(s_2)$. Hence, the second part of the optimality equation is satisfied and the SI algorithm does not switch back to π_b.

3.1.3 Value Iteration

While the derivation of a linear program and a strategy iteration approach is rather standard based on the optimality equations, Equation (3.1) does not immediately provide us with a value iteration approach. A different perspective allows for a first step towards a VI solution. Recall that by definition $\mathcal{V}_{\mathsf{rew}\text{-mp}}(\rho) = \liminf_{k \to \infty} \frac{1}{k} \mathcal{V}_{\mathsf{rew}\text{-}k}(\rho)$,

[(1)]Note that the bias-optimal action is selected only among the gain-optimal actions $Av_{g_\pi}(s)$. The SI algorithm in [Put94, Section 9.2.1] incorrectly chooses among all available actions in Step 3b., as we report in [KM17]. The corresponding proof of correctness in [Put94] uses the correct formulation.

i.e. the mean payoff of a single path is the limit of the averaged total reward. Moreover, [Put94, Theorem 9.4.1] shows that the average total reward converges to the mean payoff, i.e.

$$\mathcal{V}^{\max}_{\mathsf{rew\text{-}mp},\mathbb{E}}(s) = \lim_{k\to\infty} \frac{1}{k}\, \mathcal{V}^{\max}_{\mathsf{rew\text{-}}k,\mathbb{E}}(s).$$

We can use VI to compute the k-step total reward by setting $v_0[s] = 0$ for all states $s \in S$ and iterating

$$v_{k+1}[s] = \mathsf{rew}(s) + \max_{a \in Av(s)} \Delta(s,a)\langle v_k\rangle.$$

We have $v_k[s] = \mathcal{V}^{\max}_{\mathsf{rew\text{-}}k,\mathbb{E}}(s)$ [For+11, Section 5.2] and immediately get $\mathcal{V}^{\max}_{\mathsf{rew\text{-}mp},\mathbb{E}}(s) = \lim_{k\to\infty} \frac{1}{k} v_k[s]$ [Put94, Section 9.4.1]. However, this does not yet provide us with a practical algorithm. As mentioned in Section 2.3.2, another important part of the VI puzzle is the stopping criterion. This criterion proves to be much more involved than for reachability. In [Put94, Corollary 9.4.6], a stopping criterion is shown for the special case where all states have the same optimal mean payoff. A potential candidate for the general case is conjectured in [Put94, Section 9.4.2], which however turns out to be wrong, as we show in [Ash+17]. Instead, a topological argument actually allows us to extend the former stopping criterion, yielding the first VI solution for mean payoff applicable to general MDP, which we present in the next section. For coherence, the discussion of the stopping criteria of [Put94] is moved to the next section, too.

3.2 A Partial Exploration Approach

In this section, we present our novel approach to solving the mean payoff optimization problem via approximation, presented in [Ash+17] (see Paper A). The main contributions are threefold, namely

1. We disprove a previously conjectured stopping criterion.
2. We derive *local value iteration*, the first VI approach to solve mean payoff on general MDP.
3. We combine our approach with the partial exploration approach of [Brá+14] to obtain *on-demand value iteration*, focussing the computational effort on 'important' areas of the system.

We first explain the stopping criteria of [Put94] and disprove the one conjectured for the general case on a simple counterexample. Based on the gained insights, we derive a correct VI algorithm. Finally, we outline our novel partial exploration approach.

3.2.1 The Stopping Criterion

As explained in Section 3.1.3, we can directly compute the k-step total reward using VI. Throughout this section, we use $v_k[s] \coloneqq \mathcal{V}^{\max}_{\mathsf{rew\text{-}}k,\mathbb{E}}(s)$ to denote this k-step total reward. By averaging the k-step total reward with $\frac{1}{k}$, we get a value converging to the mean

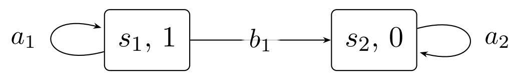


Figure 3.2: Example MDP disproving the general applicability of the stopping criterion for mean payoff given in [Put94].

payoff for $k \to \infty$. The only missing piece is a stopping criterion, estimating how close the current average is to the true mean payoff.

To this end, we again take a different perspective. Intuitively, for $k \to \infty$, the reward obtained in the k-th step, i.e. the difference between $v_k[s]$ and $v_{k+1}[s]$, should correspond to the mean payoff. Indeed, w.l.o.g. we can prove that $\mathcal{V}^{\max}_{\textsf{rew-mp}}(s) = \lim_{k\to\infty} v_{k+1}[s] - v_k[s]$ [Put94, Theorem 9.4.5][(2)]. Intuitively, once this difference 'stabilizes' for all states, it should give us the mean payoff. In order to exploit this intuition for a stopping criterion, we define the *span semi-norm* of a vector $v \in \mathbb{R}^n$ by

$$\operatorname{span}(v) \coloneqq \max_{i\in[n]} v[i] - \min_{i\in[n]} v[i].$$

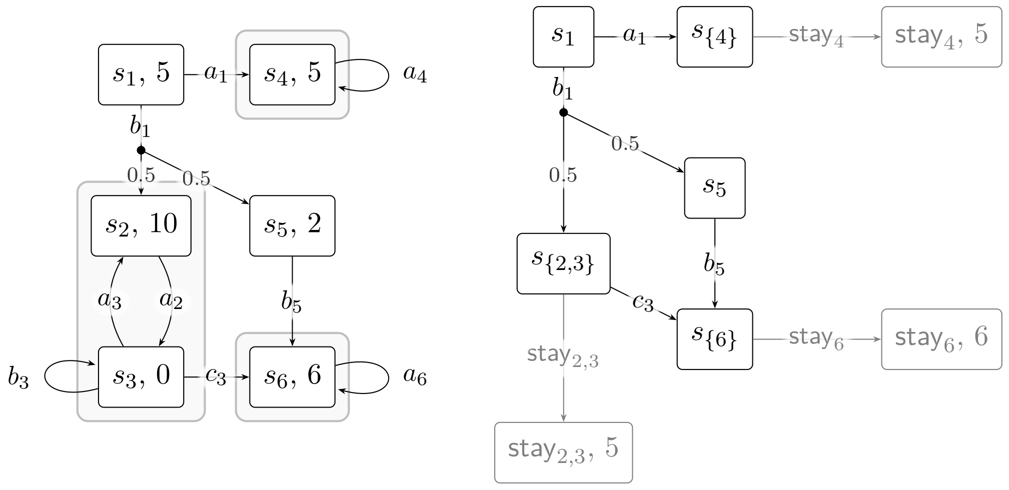
Essentially, the span semi-norm describes how much the entries of a vector differ from each other. For example, $\operatorname{span}((1,2,3)) = 2$ and $\operatorname{span}((2,2,2)) = 0$. In particular, $\operatorname{span}(v) = 0$ iff all entries of v have the same value. Now, define $t_k \coloneqq v_{k+1} - v_k$ the reward obtained in the k-th step. The span semi-norm of t_k is small if all of its entries have similar values, i.e. in step k all states obtain a similar reward.

Rather surprisingly, this directly yields a stopping criterion. In particular, [Put94, Theorem 8.5.6] shows that if $\operatorname{span}(t_k) < \varepsilon$ then $|t_k[s] - \mathcal{V}^{\max}_{\textsf{rew-mp},\mathbb{E}}(s)| < \varepsilon$ for all states $s \in S$. Note that if $\operatorname{span}(t_k) = 0$ then t_k also satisfies the gain equations of (3.1). Interestingly, we do not need the notion of bias at all; it is implicitly taken care of by the underlying total reward computation.

Under the assumption that all states have the same optimal mean payoff [Put94, Corollary 9.4.6] also shows that $\lim_{k\to\infty} \operatorname{span}(t_k) = 0$, directly yielding a VI algorithm for this special case. Unfortunately, this convergence fundamentally relies on that assumption, as we show on an example. Consider the MDP in Figure 3.2. Here, we have that $v_k = (k, 0)$ and thus $t_k = (1, 0)$. Consequently, $\operatorname{span}(t_k) = 1$ for all k and the stopping criterion is never satisfied, even though t_k equals the correct mean payoff.

As an extension to the general case, [Put94, Section 9.4.2] conjectures $\operatorname{span}(t_{k-1}) - \operatorname{span}(t_k) < \varepsilon$. This would be applicable to the example of Figure 3.2, however it is wrong in general, as we show in [Ash+17]. As [Put94, Section 9.4.2] already points out, the issue of the original stopping criterion in the general case is that different states may have different optimal mean payoff, as is the case in Figure 3.2. We extend on this observation in our general VI approach, presented in the next section.

[(2)]The proof assumes that the MDP is 'aperiodic', which can easily be obtained in general by adding a self-loop under every action with a small probability and scaling the mean payoff accordingly, see [Put94, Section 8.5.4] or [Ash+17].



(a) The input MDP. **(b)** The weighted MEC quotient.

Figure 3.3: Example execution of the local value iteration algorithm for mean payoff. The left figure shows the input MDP with its MECs shaded. The right figure then shows the computed weighted MEC quotient. The added states are shaded and self loops on them are omitted. Computing the optimal weighted reachability on the weighted MEC quotient equals the optimal mean payoff of the original MDP.

3.2.2 Local Value Iteration

In order to obtain a VI approach applicable to general MDP, we provide two central observations in [Ash+17].

Firstly, states *in the same MEC* always have the same optimal mean payoff. For an intuitive proof sketch, pick any two states s and s' of the same MEC. By Lemma 4, we can follow a strategy to go from s to s' with probability one. Once at s', we can follow the optimal strategy for s' to obtain the same mean payoff. Thus, when restricting the computation to a single MEC, we can use the already established stopping criterion to approximate the optimal value of this MEC, i.e. the best mean payoff we can achieve assuming that we remain inside this MEC.

Secondly, once we computed the optimal value of each MEC, we can reduce the problem to weighted reachability, as follows. Intuitively, when we currently are in some MEC, we can decide to either stay in the MEC and obtain the optimal value or try to move to a different MEC and obtain the rewards there. In [Ash+17], we thus replace each MEC with a single representative state and add a 'stay' action, leading to a special reward state which yields the MEC's optimal value.

Together, our *local value iteration* works as follows:

1. Compute the set of MECs $\mathsf{MEC}(\mathcal{M})$.

2. For each MEC, approximate the optimal value up to precision $\frac{\varepsilon}{2}$.

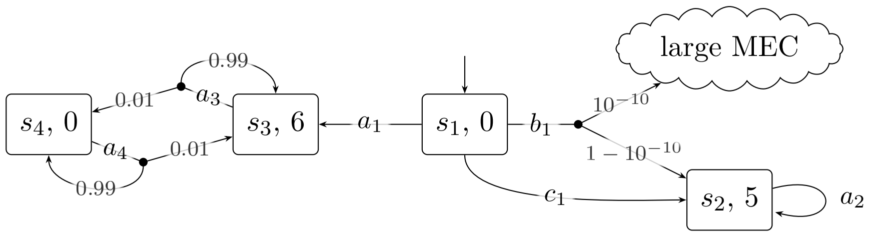


Figure 3.4: Example MDP to illustrate several optimization possibilities of local value iteration.

3. Construct the *weighted MEC quotient*[(3)] with these optimal values.

4. Approximate the optimal weighted reachability value up to precision $\frac{\varepsilon}{2}$.

This yields an ε-optimal value, as we show in [Ash+17, Theorem 3]. We omit a formal discussion here and instead provide an illustrative example in Figure 3.3. The optimal mean payoff in this MDP is 6 for all states except s_4, obtained by moving towards s_6. We highlight the MEC $(\{s_2, s_3\}, \{a_2, a_3, b_3\})$ and its representative state $\{s_2, s_3\}$. Inside this MEC, we could obtain a mean payoff of, for example, 0 or 5. However, for our purposes, only the optimal mean payoff of this MEC is relevant—once we decide to stay in a MEC there is no point in not achieving the optimum. In the MEC quotient, all 'internal' details of the MEC are eliminated, and we are only left with the decision whether to stay or transition to a different MEC. By staying, we obtain the optimal mean payoff of this MEC, namely 5.

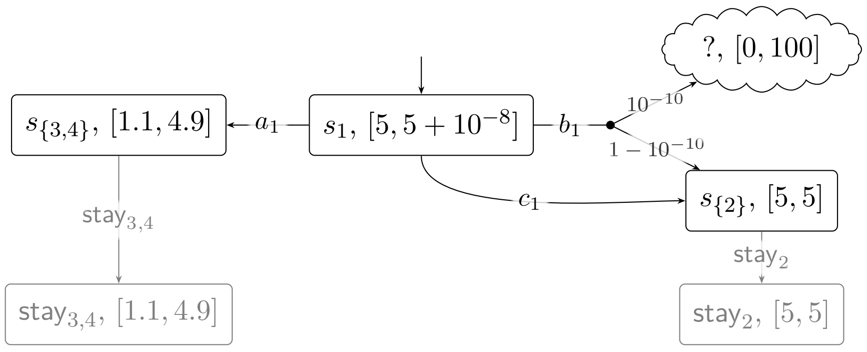
This decomposition allows us to further augment our approach with the partial exploration techniques first presented in [Brá+14], as we explain in the following section.

3.2.3 On-Demand Value Iteration

In the local value iteration approach, we approximate the value of each MEC up to the maximal required precision of $\frac{\varepsilon}{2}$. However, this may be unnecessary due to several reasons. In particular, our previous approach did not consider the initial state $\hat{s}$ of the query until the final result was computed. Consider the MDP in Figure 3.4, where we fix the initial state $\hat{s} = s_1$. Firstly, we observe that the left MEC comprising s_3 and s_4 obtains a mean payoff of 3. Once we know that this mean payoff is less than 5, we can stop further computation, as we can obtain a mean payoff of 5 by choosing action c_1 in the initial state. Secondly, observe that the value obtained in the 'large MEC' is largely irrelevant for approximation—even if we could obtain a mean payoff of 100 there, it only changes the optimal value of s_1 by 10^{-8}. Thus, assuming that we have sufficiently tight upper and lower bounds of this value, we can avoid any computation in 'large MEC'. In particular, we may be able to avoid constructing this MEC altogether.

By combining the ideas of [Brá+14] together with our observations of the previous section, we obtain our *on-demand value iteration* algorithm, capable of dynamically

[(3)]See Section 2.1.2 for a brief outline of classical MEC quotients and [Ash+17, Definition 4] for a formal definition of the weighted variant. Note that their definition already includes the reduction of weighted reachability to regular reachability.



?, $[0, 100]$
10^{-10}
$s_{\{3,4\}}$, $[1.1, 4.9]$
a_1
s_1, $[5, 5 + 10^{-8}]$
b_1
$1 - 10^{-10}$
c_1
$s_{\{2\}}$, $[5, 5]$
$\mathsf{stay}_{3,4}$
stay_2
$\mathsf{stay}_{3,4}$, $[1.1, 4.9]$
stay_2, $[5, 5]$

Figure 3.5: Snapshot of on-demand value iteration applied to the MDP from Figure 3.4. We assume a lower bound of 0 and upper bound of 100 on the reward function.

identifying both of these optimization possibilities. In essence, the algorithm keeps upper and lower bounds on the optimal mean payoff in each state. These bounds are initialized to sane values and dynamically updated until the bounds of the initial state have converged close enough.

We explain the principle ideas in several steps. First, fix some end component (R, B) of the MDP. Assuming that we remain inside this EC, all states can achieve the same optimal mean payoff, denoted v_R. As explained before, we can compute the k-step total rewards v_k restricted to the EC and the differences t_k eventually converge to the optimum v_R. By the stopping criterion of [Put94], we have that $\mathrm{span}(t_k) < \varepsilon$ implies $|t_k[s] - v_R| < \varepsilon$. In other words, we have that $|t_k[s] - v_R| \leq \mathrm{span}(t_k)$. Inserting the definition of the span semi-norm gives us that $\min_{s \in R} t_k[s] \leq v_R \leq \max_{s \in R} t_k[s]$. Thus, the vector t_k directly gives us bounds on v_R. In particular, we can refine bounds on the optimal value in any EC *on demand* by continuing the iteration.

Now, assume that we construct the complete input MDP and compute all MECs. Then, we again construct the weighted MEC quotient. However, instead of a single value, the stay states are assigned upper and lower bounds on the value obtained in this MEC. Using the above intuition, we can refine the bounds of any stay state. All other states of the quotient can be updated by simply propagating the upper and lower bounds according to the computation of weighted reachability.

If we apply all these updates synchronously, we effectively obtain our previous local value iteration approach. However, by dynamically identifying which states to update, we can focus our computation on important areas of the system. For example, if the expected upper bound of one action is strictly less than the lower bound of another action, we can safely omit the former action. In [Ash+17], we explain a guided sampling-based approach for selecting which states to update. Furthermore, we augment this approach with lazy construction and *on-the-fly detection* of end components, allowing us to dynamically build the weighted MEC quotient of the relevant parts of the MDP.

See Figure 3.5 for a sample execution of on-demand value iteration on the MDP from Figure 3.4. The algorithm can approximate the mean payoff of s_1 without investing too much computational effort in the MEC $\{s_3, s_4\}$ or constructing the ‘large MEC’ at all.

3.3 Precise Solutions with Strategy Iteration

While the previous value iteration algorithms inherently only provide approximation solutions in general, we sometimes require precise values. Recall that linear programming yields such precise results, however it does not scale to practical applications. Instead, we investigated the classical strategy iteration approach. Similar to LP, a standard SI algorithm already exists for mean payoff problems, see Section 3.1.2 for an overview. Alas, this algorithm also quickly reaches its limits in practice. In [KM17] (see Paper B), we identified the two following issues:

1. The equation system of Equation (3.2) has size $O(|S|^2)$ and typically $\approx 2k \cdot |S|$ non-zero entries, where k is the average number of successors under an action. This quickly becomes intractably large already for moderately sized models ($|S| \approx 10^6$). Moreover, the equation system is under-determined, ruling out efficient approaches. Uniqueness can be obtained by adding additional rows. However, then the system is not square any more, again ruling out standard approaches. Finally, the *condition number* of this equation system often is rather large (we observed $\kappa \gg 10^9$ already on moderately sized models), quickly resulting in numerical instabilities.

2. In order to determine the gain of a strategy using the equation system (3.2), the bias also has to be determined. However, often only the gain is needed for an improvement step. Similarly, the equation system is solved precisely in each step, while a quick approximation could already yield a possible improvement.

To address these problems, we propose two classes of optimizations to SI, which drastically increase its performance on real-world problems. Firstly, we consider topological optimizations, allowing for a divide-and-conquer approach, addressing the first problem. Secondly, we make use of our observations in Section 3.2. We propose a mechanism alternating between value iteration and strategy iteration in a non-trivial way. This approach uses value iteration to approximate the performance in each state and refines the approximated values in potentially optimal states, tackling the second issue. Moreover, we present a combination of the two approaches, yielding further improvements.

3.3.1 Topological Optimizations

In this section, we briefly outline several topological observations of [KM17]. Recall that for an MDP $\mathcal{M} = (S, Act, Av, \Delta)$ and memoryless deterministic strategy $\pi \in \Pi_{\mathcal{M}}^{\mathsf{MD}}$, we obtain a Markov chain $\mathcal{M}^\pi = (S, \delta^\pi)$ where $\delta^\pi(s, s') = \Delta(s, \pi(s), s')$. Moreover, the gain obtained by solving Equation (3.2) exactly describes the mean payoff obtained in the Markov chain $\mathcal{M}^\pi$.

MEC decomposition Using the ideas presented in Section 3.2, we can compute the overall mean payoff by first computing the optimal value of each MEC separately and then solve the associated weighted reachability problem, again using SI.

BSCC compression Given a strategy $\pi \in \Pi_{\mathcal{M}}^{\mathsf{MD}}$, all states which are in the same BSCC of $\mathcal{M}^\pi$ have the same gain. As such, we can 'compress' the equation system by replacing all gain variables corresponding to states in the same BSCC by a single variable, representing the gain of the respective BSCC. Such an analysis is common to linear equation solving in general. However, this compression approach interestingly always yields a square equation system with a unique solution, allowing us to use more efficient solution approaches.

SCC decomposition By extending the above idea, we can decompose the large equation system into several smaller problems. Observe that the analysis of a BSCC is completely independent of all other parts of the system. Thus, we can solve Equation (3.2) for each BSCC separately. Then, we observe that a state which is not in a BSCC does not 'earn' any gain on its own. Instead, its gain only depends on the probability of reaching particular BSCCs. This observation allows us to further decompose the problem into several smaller equations, all with square matrices and unique solutions.

3.3.2 Strategy Iteration by Approximations

In this section we demonstrate how we can use approximation approaches in the context of strategy iteration. Intuitively, as long as we are working with a 'bad' strategy, we are not interested in knowing how bad it is exactly, we just want to know how to improve it. In [KM17], we assume that we are given an *approximation oracle*, yielding upper and lower bounds on the gain, i.e. values $L^\pi(s), U^\pi(s)$ with $\mathcal{V}^\pi_{\mathsf{rew\text{-}mp},\mathbb{E}}(s) \in [L^\pi(s), U^\pi(s)]$. These approximations can be obtained by, for example, using the value iteration ideas discussed in the previous Section 3.2. Then, we try to improve the strategy based on these approximations. In particular, if for any state s there exists an action $a \in Av(s)$ such that $U^\pi(s) < \Delta(s,a)\langle L^\pi \rangle$ we can safely switch to action a. When no such improvement is possible, we employ one round of strategy iteration. If this round does not yield any improvements, we already are converged, otherwise we continue the process.

We deliberately do not improve the strategy based on bias approximations, due to reasons outlined in [KM17]. Firstly, it proves to be rather difficult to approximate the bias. Secondly, recall that an improvement w.r.t. bias is only allowed among actions yielding the exact same gain, which we cannot determine using approximations.

3.3.3 Combining both Approaches

In order to combine our two approaches (topological optimization and approximation improvements), we use the following observation. Recall that each state in a MEC has the same optimal mean payoff. In particular, the optimal gain of a MEC certainly is at least as large as the lower bound on the gain of any strategy achieved in any state of the MEC. Thus, let $(R, B) \in \mathrm{MEC}(\mathcal{M})$ be a MEC, $\pi \in \Pi^{\mathsf{MD}}_{\mathcal{M}}$ the current strategy, and $L^\pi(s)$ the lower bound approximation. Then, let $L^\pi(R) = \max_{s \in R} L^\pi(s)$ the maximal lower bound of all states in this MEC. In particular, the states may belong to different BSCCs in the induced Markov chain $\mathcal{M}^\pi$. Now, any state in (R, B) with an upper bound smaller than $L^\pi(R)$ clearly performs suboptimally and we can easily adapt the strategy such that all those states 'point towards' the better region by virtue of Lemma 4.

3.4 Conclusion

We presented two different approaches to solve mean payoff queries in practice. Our value iteration approach is able to approximate the result up to an arbitrary precision and focusses computational effort on relevant parts of the system. Strategy iteration on the other hand provides us with precise solutions at the expense computational effort.

As an interesting extension, we seek for several ways to combine the two approaches. In particular, strategy iteration is good at solving MDP where VI tends to be slow. So, we could augment the approach of Section 3.2 with an interleaved strategy iteration to speed up convergence. Dually, the SI method of Section 3.1.2 currently solves the equation system on the whole MDP. By building on the ideas of Section 3.3.3, we might be able to identify regions of interest and restrict computation to those.

For the practical part, we only provided a prototype implementation of our approaches in [Ash+17; KM17]. A unified, efficient implementation in popular model checkers such as PRISM [KNP11] or Storm [Deh+17] may provide us with further insights which structural properties of a system favour one approach over the other.

4 Probabilistic LTL Model Checking

In this chapter, we discuss the analysis of objectives given in linear temporal logic (LTL) as introduced in Section 2.2.2. Similar to Chapter 3, we are given an MDP and an initial state. However, instead of a reward function, we are provided with a labelling of the MDP's states and an LTL formula to be satisfied. Now, the task is to compute the maximal probability of satisfying the given formula.

Problem Statement

Let $\mathcal{M} = (S, Act, Av, \Delta)$ be an MDP, $\hat{s} \in S$ an initial state, AP a finite set of atomic propositions, $\mathsf{L} : S \to 2^{\mathsf{AP}}$ a labelling function, and ϕ an LTL formula over AP. Compute the maximal probability of satisfying the formula ϕ

$$\mathcal{V}^{\max}_{\phi,\mathbb{E}}(\hat{s}) = \sup\nolimits_{\pi \in \Pi_{\mathcal{M}}} \mathcal{V}^{\pi}_{\phi,\mathbb{E}}(\hat{s}).$$

Again, we first outline the classical solution approaches in Section 4.1. Then, we explain our contributions in Section 4.2, combining [Kře+18] (see Paper C) and [KMS18] (see Paper D). Both provide several practical optimizations of the LTL-to-automaton translation explained in Section 4.1, thus speeding up the subsequent model checking task in practice. Brief concluding remarks are given in Section 4.3.

4.1 The Classical Solution Approach: Automata

Before we explain our improvements towards this problem, we first outline the classical solution approach. Further information can be found in [For+11, Section 7] and [BK08, Section 10.3, 10.6.4]. The so called *automata-theoretic approach* [VW86] applied to MDP roughly proceeds as follows.

1. Translate the LTL formula into a *(deterministic) ω-automaton* (introduced in the next section).
2. Compute the *product* of the MDP with the obtained automaton.
3. Identify *winning MECs* in the product.
4. Compute the maximal probability of reaching the winning MECs.

Steps 3 and 4 are executed on the product MDP, obtained in Step 2. Naturally, the size of the product strongly correlates with the size of the automaton from Step 1. Our improvements presented in Section 4.2 are focussed on minimizing the size of this automaton, simplifying the subsequent steps. In Section 5.3, we briefly discuss an approach aiming to similarly improve the performance by reducing the size of the MDP.

4.1.1 Automata

We briefly introduce the concept of ω-automata.

Definition 5 (Deterministic ω-Automata)**.** A *deterministic ω-automaton $\mathcal{A}$* over the set of atomic propositions AP is a tuple $(Q, \mathsf{AP}, \delta, \hat{q}, \alpha)$ where

- Q is a finite set of states,
- $\delta : Q \times \mathcal{P}(\mathsf{AP}) \to Q$ is a *transition function*[(1)],
- $\hat{q} \in Q$ is the *initial state*, and
- α is an *acceptance condition* (described later).

A word $w \in \mathcal{P}(\mathsf{AP})^\omega$ naturally induces an *infinite run* $\rho \in (Q \times \mathcal{P}(\mathsf{AP}))^\omega$ on the automaton starting in the initial state and following the letters of w. We write $\mathcal{A}(w)$ to denote the unique run of $\mathcal{A}$ on w. A transition $t \in Q \times \mathcal{P}(\mathsf{AP})$ *occurs* in a run ρ if there is some i with $\rho_i = t$. We use $\mathrm{Inf}(\rho)$ to denote the set of all transitions occurring infinitely often in ρ.

A wide variety of acceptance conditions have been studied in the literature. In essence, such an acceptance condition decides whether a given word w is *accepted* or *rejected* by the automaton based on $\mathrm{Inf}(\mathcal{A}(w))$, i.e. the set of transitions which are visited infinitely often. In other words, α maps a set of transitions to either 'accept' or 'reject'. The language of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is the set of words accepted by $\mathcal{A}$. An automaton *recognizes* a language $\mathcal{L}$ if $\mathcal{L}(\mathcal{A}) = \mathcal{L}$.

For simplicity, we introduce only two relevant acceptance conditions, namely Rabin acceptance [Rab69] and its generalized variant. An instance of Rabin acceptance is given by a set of *Rabin pairs*. Each Rabin pair is of the form (F_i, I_i) where both $F_i, I_i \subseteq Q \times \mathcal{P}(\mathsf{AP})$ are sets of transitions. These F_i and I_i are called the *prohibited set* (or ***F**inite set*) and the *required set* (or ***I**nfinite set*), respectively. A word w is accepted if there exists a Rabin pair (F_i, I_i) such that some transition of I_i is visited infinitely often and every transition of F_i is visited only finitely often. See Figure 4.1 for an example automaton using Rabin acceptance. Recently, *generalized Rabin* acceptance, has been proposed [CGK13]. Here, each Rabin pair comprises a set of required sets, i.e. $(F_i, \{I_i^j\}_{j=1}^{k_i})$. Now, a word is accepted by a generalized Rabin pair with index i if for each $j \in [k_i]$ some transition of I_i^j is visited infinitely often and again every transition of F_i is visited only finitely often. This generalizes the Rabin condition, where each $k_i = 1$. Every generalized Rabin automaton can be de-generalized into an equivalent Rabin automaton, which however may incur an exponential blow-up [KE12].

[(1)]For simplicity, we assume that δ is a total function, i.e. there is a transition for every input.

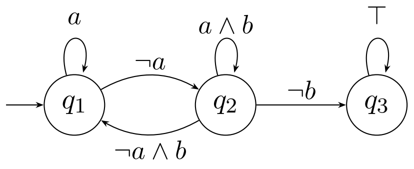


Figure 4.1: Example automaton over the atomic propositions $\mathsf{AP} = \{a, b\}$. The initial state $\hat{q}$ is marked with an arrow, in this case q_1. When drawing transitions or denoting the acceptance condition, we write logical formulae over the atomic propositions instead of listing each transition separately. For example (q_2, b) denotes both the transition $(q_2, \{b\})$ and $(q_2, \{b, a\})$. The Rabin acceptance is given by $F_1 = \{(q_2, a \wedge b), (q_3, \neg a)\}$ and $I_1 = \{(q_1, b)\}$.

Remark 2. Traditionally, acceptance for ω-automata is defined *state based*, i.e. the acceptance condition is formulated in terms of infinitely or finitely often visited states instead of transitions. However, in line with recent works, e.g., [CGK13; Dur+16; Kře+18], we instead use transition based acceptance, which is both theoretically and practically more concise.

We are interested in automata, since they can be used to represent the words satisfying any given LTL formula, corresponding to Step 1. Formally, we have that for any LTL formula ϕ, there exists a Rabin automaton $\mathcal{A}_\phi$ such that $\mathcal{L}(\phi) = \mathcal{L}(\mathcal{A}_\phi)$ [EKS16].(2) There are many different ways to obtain such an automaton. We discuss some of them in Section 4.2. While the worst-case blow-up is known to be doubly exponential in the size of the formula (see, e.g., [AT04]), the automata obtained in practice are often reasonably sized. We first explain how we can use automata for model checking, i.e. the central ideas behind Steps 2 and 3.

4.1.2 Probabilistic LTL Model Checking

In this section, we outline how we use ω-automata to determine the maximal probability of satisfying a given LTL formula in a (labelled) MDP. We first define the notion of product MDP. Intuitively, this product tracks both the evolution of the MDP together with the automaton. The automaton progresses according to the labelling of the MDP.

Definition 6. Fix a set of atomic propositions AP, an MDP $\mathcal{M} = (S, \mathit{Act}, \mathit{Av}, \Delta)$ with labelling function $\mathsf{L} : S \to \mathcal{P}(\mathsf{AP})$, and an automaton $\mathcal{A} = (Q, \mathsf{AP}, \delta, \hat{q}, \alpha)$. We define the *product MDP* $\mathcal{M} \times \mathcal{A} := (S \times Q, \mathit{Act} \times Q, \mathit{Av}^\mathcal{A}, \Delta^\mathcal{A})$, where

- $\mathit{Av}^\mathcal{A}((s, q)) := \{(a, q) \mid a \in \mathit{Av}(s)\}$, and
- $\Delta^\mathcal{A}((s, q), (a, q), (s', q')) := \Delta(s, a, s')$ if $q' = \delta(q, \mathsf{L}(s))$ and 0 otherwise.

(2) We note that the inverse statement does not hold, i.e. there are automata for which no corresponding LTL formula exists, see, e.g., [DG08].

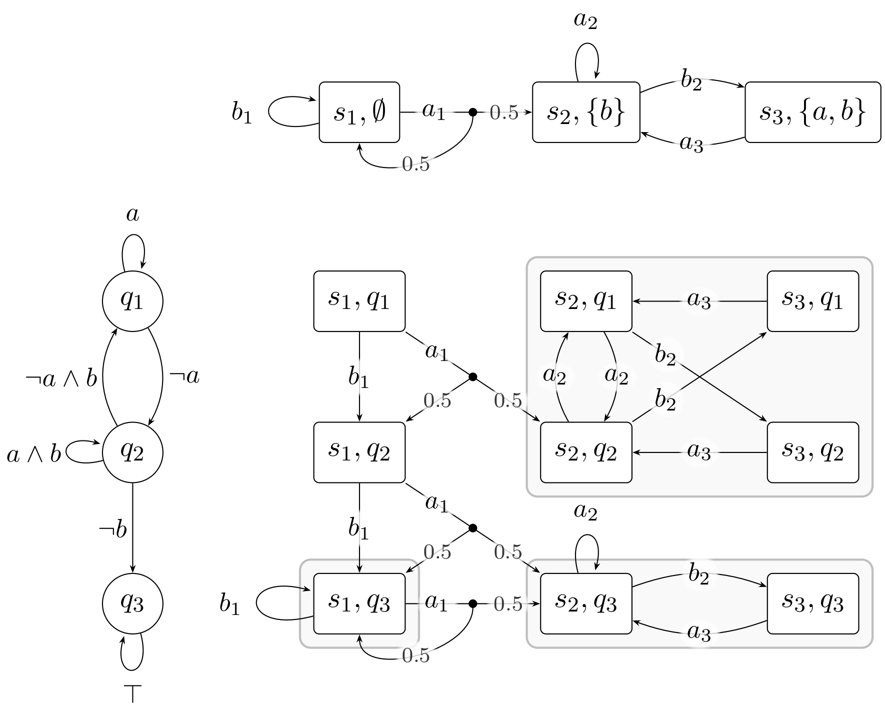


Figure 4.2: Example product MDP of the automaton from Figure 4.1 (left) and an MDP (top). The labels assigned to the MDP's states by the labelling function L are written next to the states. For readability, we write the product MDP actions as elements of *Act* instead of $Act \times Q$.

The paths of the product MDP directly correspond to the paths of the original MDP. However, we additionally know which transition in the automaton the current path is taking based on the current state. This allows for graph-based analysis.

Recall that acceptance of a path in the MDP $\mathcal{M}$ is based on which transitions of the automaton $\mathcal{A}$ are taken infinitely often. Furthermore, Lemma 2 (in Section 2.1) shows that almost all runs eventually remain in a single MEC. Together, we observe that the set of infinitely often seen transitions under almost all runs belong to a single MEC of the product MDP. Consequently, we can analyse each such MEC separately.

It turns out that product MECs are either 'winning' or 'losing', i.e. there either exists a strategy such that almost all runs in this MEC satisfy the LTL formula or for all strategies the probability of satisfying the formula in the MEC is zero. In other words, the value of each product MEC either is 0 or 1. We can determine this value as follows.

Given a MEC in the product MDP, our goal is to check whether there exists a Rabin pair such that F_i is visited finitely often and I_i infinitely often. We thus can check each Rabin pair separately; let (F_i, I_i) be a Rabin pair. We remove all product states which belong to a transition in F_i, i.e. all states (s, q) where $(q, \mathsf{L}(s)) \in F_i$. Then, we check whether in the remains of this MEC we can find an end component which contains an edge of I_i. In that case, we can obtain a winning strategy by first moving towards this

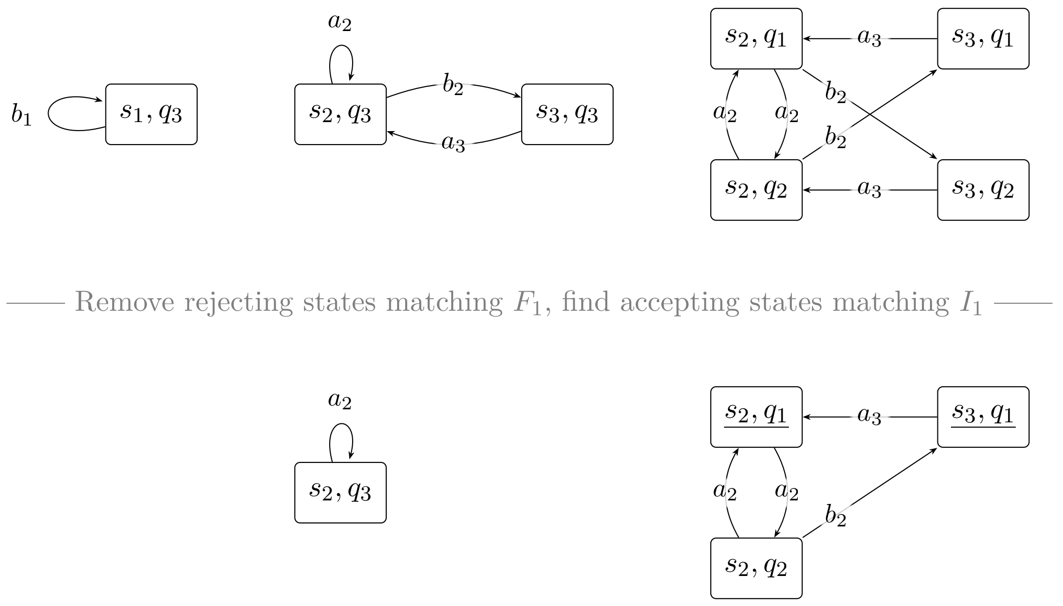


Figure 4.3: Illustration of the LTL model checking steps on the product MDP of Figure 4.2. Recall that the labelling is given by $\mathsf{L} = \{s_1 \mapsto \emptyset, s_2 \mapsto \{b\}, s_3 \mapsto \{a, b\}\}$ and the Rabin acceptance by $F_1 = \{(q_2, a \wedge b), (q_3, \neg a)\}$ and $I_1 = \{(q_1, b)\}$.

sub-region of the product MEC and then cycling inside it. By construction, the end component does not contain any prohibited edge F_i and visits some transition of I_i infinitely often (see Lemma 4). Moreover, the converse holds, too: If we cannot find such an end component, there is no strategy which satisfies this Rabin pair in the considered MEC with non-zero probability. After identifying the set of winning MECs, we obtain the overall probability of satisfying the given LTL query by a simple reachability query, maximizing the probability of reaching such a winning MEC starting from the initial state $(\hat{s}, \hat{q})$. See, e.g., [For+11, Section 7.3] for further details.

Based on the MDP of Figure 4.2, we illustrate this process in Figure 4.3. For each MEC, we remove all product states which match F_1 and then identify states accepted by I_1 (underlined). For the leftmost MEC, all states match F_1 and thus it is removed completely. In the middle MEC, only (s_2, q_3) remains, which does not satisfy I_1 and thus is losing, too. Finally, the result of the rightmost MEC contains an EC with winning states and thus the *whole* MEC is winning. In particular, observe that (s_3, q_2) is winning, too, since it can move to a winning state (s_2, q_2) via a_3. All in all, we obtain that only the top-right MEC of Figure 4.2 is winning, and thus the maximal probability of a run starting in s_1 being accepted by the automaton is 0.5.

The procedure of identifying winning MECs for generalized Rabin automata shares the same essential ideas. Generalized Rabin automata are appealing because they often are significantly smaller than corresponding Rabin automata, while computation of winning MECs is not much more complicated [CGK13].

4.2 Practical Improvements for LTL-to-automata Translations

As we explained at the beginning of this chapter, our contribution towards LTL model checking is focussed on providing smaller automata in Step 1 of the overall approach. We provided several practical improvements in [Kře+18] (see Paper C) and [KMS18] (see Paper D). Most of the contributions are purely focussed on the implementation, thus we omit an in-depth treatment and only report major points.

Owl, presented in [KMS18], is a library providing an efficient framework for LTL and ω-automata related constructions. Among others, Owl provides the following features:

- A versatile yet efficient representation of LTL and automata, with tight integration of BDDs used to represent transitions.
- Efficient implementation of numerous general purpose algorithms for LTL and ω-automata, such as rewriting, SCC decomposition, acceptance optimization, etc.
- Flexible ‘pipeline-style’ interface, allowing for rapid prototyping and comparison of different approaches.
- Fully automated testing framework for quickly detecting bugs in newly developed constructions, using the excellent tool `ltlcross` of Spot [Dur+16].

As of now, Owl serves as basis for several constructions, namely our implementation of Rabinizer 4 [Kře+18] and IAR [Kře+17] as well as Delag [MS17], Strix [MSL18], and many more. Moreover, Owl is actively used in several student projects.

In [Kře+18], we present *Rabinizer 4*, a complete rewrite of *Rabinizer 3.1* [KK14], including several fundamental extensions. At its heart, Rabinizer is a collection of constructions centred around translations from LTL to (generalized) Rabin automata. The constructions of Rabinizer 4 are implemented in Owl and are part of its standard distribution. In particular, this includes a complete reimplementation of Rabinizer 3.1, called **ltl2dgra**, with following additional features:

- Support for the full syntax of LTL, e.g. the (weak) release operator,
- BDD-based rewriting and successor computation,
- suspension analysis and detection of several special cases, and
- several generic optimizations such as SCC decomposition approaches.

Furthermore, **ltl2dgra** heavily profits from the general purpose optimizations provided by Owl. Together, this yields much smaller automata in practice. A practical comparison of these effects can be found in [Kře+18, Section 3].

Owl still is under active development and the practical performance of its tools continues to increase. By providing even smaller automata to the LTL model checking framework, the size of the product MDP decreases and analysis is much faster.

4.3 Conclusion

We provided practical improvements to the translation of LTL formulae to automata, in particular (generalized) Rabin automata, which are well-suited for probabilistic model checking. One important remaining step is a tighter integration with model checkers such as PRISM [KNP11]. As of now, the 'exchange' of the constructed automaton from Owl to PRISM happens via serialization to a textual representation in the *Hanoi Omega-Automaton Format (HOAF)* [Bab+15]. As such, constructing the automaton on demand or directly combining the BDD representation of the automaton's transition structure with PRISM's BDD representation of the MDP is impossible.

The translations offered by Rabinizer yield a 'semantic' labelling of the automaton states, i.e. each state is equipped with meta-information describing the language obtained when starting from that state. In [KMM19], we exploit this labelling to solve parity games, however the underlying idea should be equally applicable to probabilistic model checking, too. In particular, the sampling-based methods of [Brá+14] can be guided by a sophisticated heuristic. When applied to the product MDP, such a guidance could be derived from the state labels of the MDP.

5 The Notion of Cores

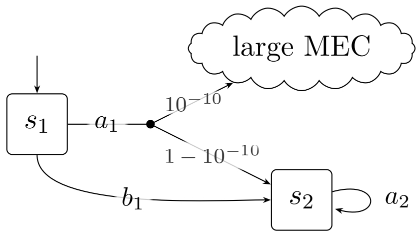


Figure 5.1: Example MDP to motivate the notion of cores.

In this chapter, we discuss the novel notion of *cores*, introduced in [KM19] (see Paper E). As a motivating example, we again consider the MDP of Figure 3.4. It is depicted again in a simplified manner in Figure 5.1. While discussing this MDP in Section 3.2, we note that we do not need to consider the 'large MEC' in order to approximate the optimal mean payoff of state s_1. The key insight motivating this section is that, more generally, we do not need to consider this part of the system for an approximation of *any* target set when starting in s_1. While [Brá+14; Ash+17] use this observation to specifically adapt solution methods of reachability and mean payoff, respectively, we instead aim to derive a generic approach. We want to identify *irrelevant* states, such as the 'large MEC'. Then, we can run existing solution methods on the restricted model, excluding these irrelevant states. Since we reduce the number of states, we speed up the subsequent computations. Abstractly, this extends the straightforward idea of only considering reachable states by instead considering *sufficiently* reachable states. See [KM19, Section 3] for further motivation.

As the notion of cores is fundamentally new, there is no formal problem statement, instead we are interested in discovering the implications of this idea. To this end, we first present the definition and basic properties of cores in Section 5.1, outline our approach to identifying cores in Section 5.2, and discuss some applications and extensions in Section 5.3. Finally, we conclude in Section 5.4.

5.1 Definition of Cores

In this section, we discuss the notion of cores and prove some fundamental properties. First, we define the notion of *ε-core*. Intuitively, an ε-core is a set of states which can only be exited with probability less than ε. More formally, given an MDP $\mathcal{M}$, initial state $\hat{s}$, and precision $\varepsilon > 0$, a set of states S_ε is an *ε-core* if

$$\mathsf{Pr}^{\max}_{\mathcal{M},\hat{s}}[\Diamond \overline{S_\varepsilon}] < \varepsilon,$$

i.e. the probability of ever exiting the core S_ε under any strategy is smaller than ε. Note that the core property also depends on the initial state $\hat{s}$. However, for notational convenience, we omit explicitly denoting this dependency. When ε is clear from the context, we refer to an ε-core by 'core'.

With this simple definition, we already can prove the key statement motivating our interest in cores, namely that they are both sufficient and, in a sense, required to approximate reachability queries up to precision ε. In particular, a set S_ε is an ε-core of $\mathcal{M}$ iff for every reachability query $T \subseteq S$ we have that

$$0 \leq \mathcal{V}^{\max}_{\Diamond T,\mathbb{E}}(\hat{s}) - \sup_{\pi \in \Pi_{\mathcal{M}}} \mathsf{Pr}^{\pi}_{\mathcal{M},\hat{s}}[\Diamond (T \cap S_\varepsilon) \cap (S_\varepsilon \times \mathit{Act})^\omega] < \varepsilon.$$

In other words, for any target set T, we can determine $\mathcal{V}^{\max}_{\Diamond T,\mathbb{E}}(\hat{s})$ up to precision ε by determining the reachability of T on the sub-model induced by an ε-core, i.e. by only considering runs which remain inside S_ε. Conversely, if we consider a set of states not satisfying the core property we may not be able to answer a given reachability property ε-precisely. Note that this statement is weaker than [KM19, Theorem 7]. In an upcoming journal version thereof [KM], we show that the original statement is too strong and prove the above variant.

Nevertheless, this statement motivates us to find cores. We highlight that many typical objectives can be decomposed into pre-processing steps (usually restricted to MECs) and a subsequent reachability analysis. In particular, all 'infinite horizon' objectives introduced in Section 2.2, namely (weighted) reachability, mean payoff, and LTL, allow for such a decomposition. Hence, we can extend the above statement to those objectives and our initial motivation to use cores for practical speed-ups also is applicable for these other objectives.

The whole set of (reachable) states trivially gives an ε-core for any ε. More generally, this set intuitively is a '0-core'. However, this does not yield any computational advantage. In a sense, this is what traditional explicit methods are doing; all reachable states are considered in the computation. We naturally are interested in finding small cores to speed up computation as much as possible. However, it turns out that finding inclusion-minimal cores is NP-complete [KM19, Theorem 6]. Thus, we resort to a best-effort approach, presented in the next section.

5.2 Finding Cores on MDP

While finding minimal cores is infeasible, identifying any sufficiently small core already leads to significant speed-ups. We present a sampling-based approach able to identify reasonably small cores in practice. It is inspired by the partial exploration ideas of [Brá+14] and [Ash+17] (discussed in Section 3.1.3). However, instead of trying to answer a given reachability or mean payoff query, we aim to identify a property of the given system, *independent* of any query.

Intuitively, the approach works as follows. We separate all states into two categories, *explored* and *unexplored*. First, only the initial state $\hat{s}$ is considered explored. For each state, we furthermore keep an upper bound on the probability of reaching an unexplored state. Initially, this upper bound clearly is 1 for every state. We repeatedly explore states and conservatively update these upper bounds until the upper bound of the initial state is smaller than ε. The set of explored states then clearly is a core.

This leaves us with two questions, namely 'Which states should be explored?' and 'How are the upper bounds updated?'. To answer the first question, we employ guided sampling. Intuitively, when we are in some state s, we want to prioritize exploring (i) unexplored states which are (ii) 'easy' to reach, since these contribute the most to the upper bound of s. For example, recalling the example from Figure 5.1, in $\hat{s}$ we want to prioritize exploring s_2 over any state of 'large MEC', since the latter is hardly reachable. To this end, we essentially sample a successor proportional to their upper bounds (addressing (i)) times the transition probability (addressing (ii)). Whenever we sample an unexplored state, we add it to the set of explored states. This approach is inspired by heuristics identified in [Brá+14; Ash+17], but, as we show in [KM19], other choices are possible, too.

For the second question, recall that we are essentially dealing with bounds on a reachability query, namely 'the probability of reaching an unexplored state'. However, the query dynamically changes during the analysis, since the set of unexplored states is repeatedly modified. Nevertheless, it turns out that we can use the classical Bellman equations for reachability presented in Equation (2.4). By further augmenting these ideas with an *on-the-fly* detection of end components, this approach allows us to identify small cores on several models very efficiently, significantly increasing performance of subsequent analyses.

5.3 Extensions

In [KM19] we also discuss several extensions and applications of cores.

Firstly, cores provide a step towards *understandability* of probabilistic systems. Since cores essentially comprise all 'important' states, we can focus on such states when trying to understand or debug such a system. In particular, during the design phase of a system, we can quickly generate an imprecise core to get a rough overview of the general system structure, potentially identifying design faults at an early stage.

We also discuss the notion of k-step cores, i.e. the bounded equivalent to cores. Intuitively, a k-step ε-core is a set of states which can be left with probability at most ε within k steps. As such, they can be directly applied to improve the performance of approximating k-step queries. However, as we argue in [KM19, Section 4.4], we can use this variant for several other applications. For example, we can use them to quickly approximate the systems long-term behaviour, i.e. the K-step behaviour for $K \gg k$. While there are no formal guarantees for the precision of this approximation, we found that on realistic models this tends to be surprisingly precise and, due to the smaller size of k-step cores, much faster than direct analysis. Moreover, we use this insight to derive the idea of *stability*. In essence, we look at how the precision of this approximation changes with increasing K. As such, we get an idea how 'stable' the k-step core is. Now, for example, a sudden change of precision after 100 steps may indicate that a significant event happens around these 100 steps. This provides another perspective on the understandability of systems.

We can also transfer the notion of cores as well as the discussed extensions to different models, such as stochastic games, or richer formalisms, such as mean payoff or LTL. Both extensions to other models as well as richer formalisms turn out to be straightforward. In particular, most objectives can be decomposed into pre-computations and reachability analysis. For example, consider probabilistic model checking of LTL as discussed in Chapter 4. It is straightforward to show that the product MDP obtained by multiplying a core with the respective automaton still allows for ε-approximations.

Finally, a promising general extension are *cost-bounded* cores. These denote a set of state where we remain with high probability given that we do not exceed a cost bound. This generalizes both infinite cores (0 cost everywhere) and finite cores (1 cost everywhere). Thorough analysis may provide a unified algorithm to solve both cases and shed further insight on the underlying idea. However, this remains future work.

5.4 Conclusion

We presented the novel notion of cores, a framework for approximate verification of MDP via partial exploration. Our sampling-based approach identifies small cores efficiently for both Markov chains and Markov decision processes. The principle idea of cores, identifying relevant parts of the system's state space, can be applied to numerous other formalisms, such as stochastic games or probabilistic programs. Such extensions could provide us with valuable understanding of the respective models. In particular, extending finite cores to continuous time systems poses a non-trivial challenge.

6 Taming Risk in Probabilistic Systems

Decisions in real life are often shaped by analysis of the associated risk. Thus, many fields such as psychology or finance invested great effort in both understanding the human notion of risk and finding ways of formally quantifying it. Yet, despite risk being an essential notion in decision making—especially when it comes to safety critical decisions—the verification community hardly dealt with this problem at all. In this chapter, we motivate the treatment of risk in verification and present a first step towards risk-aware analysis of probabilistic systems, as presented in [Kře+18] (see Paper F). In particular, we show that the computational complexity of synthesizing risk aware strategies for our particular application is equivalent to that of regular approaches only aiming for expectation maximization. This motivates further research into the understanding of risk and practical algorithms.

We first present our notion of risk in Section 6.1, analyse the associated model checking problem in Section 6.2, and conclude in Section 6.3.

6.1 A Measure of Risk

In the context of probabilistic systems, random outcomes are typically aggregated using expectation, which is completely oblivious to risk. As a small example, consider an abstract model of a power plant. Naturally, we want to control this plant in a way such that the average power-output is maximized. When operating normally, the plant produces 100 MW. Now, assume that we can overcharge this plant to produce 110 MW. However, overcharging puts the plant into an unstable state, resulting in a 5% chance of damage to the plant, leading to a maintenance period and 0 MW output. Nevertheless, overcharging the plant in this manner is preferred by a purely expectation maximizing agent, while this behaviour might not be desirable in reality. We emphasize that this highly depends on the context and modelling. For example, if we operate several thousand plants in parallel, we might be much more inclined to accept this risk compared to a situation where only a few are operating.

One might think that this problem can be tackled by adding penalties to such incidents. However, there are several problems associated with such penalties. Firstly, we lose interpretability of the obtained result. When we, for example, associate '-100 MW' with a plant failure, this has no clear real-world equivalent and an optimal value of '50 MW' might mean '100 MW' output in 75% of the cases and failure in 25%. Secondly, it is unclear how much such a failure should be penalized, especially if there are several kinds thereof. If penalties are too large, it might be the case that the optimal strategy is not to act at all. In our example, this means that the optimal value is 0 MW and we do not even begin to produce power, since just starting up the plant incurs a small risk

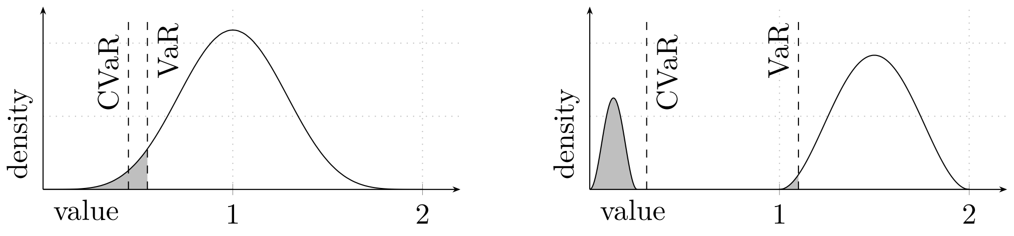


Figure 6.1: Illustration of VaR and CVaR for two random variables, taken from [KM18].

of failure. Yet, in reality, decisions are made despite the slim chance of catastrophic outcomes. As such, we are interested in approaches which are *aware* of risk and act to control it, not eliminate it completely, since this might be overly costly.

To tackle this problem, we employ the following idea. Instead of maximizing expectation over all possible strategies, we first prune 'risky' strategies (according to some notion of risk). Then, we search for the expectation-optimal strategy among those with a controlled level of risk. Now, the question is how to quantify the risk associated with a particular strategy. Recall that an MDP together with a strategy yields a distribution over paths, and a performance function $\mathcal{V}_{\mathsf{perf}}$ as defined in Section 2.2 is a random variable over paths. For now, let us assume that the performance is measured as a real number. Together, we obtain a distribution over real values for every strategy. For example, if we consider the above 'overcharging' strategy, we obtain a distribution where $\{0 \mapsto 5\%, 110 \mapsto 95\%\}$. In order to quantify risk, we define risk based on such a distribution, and then restrict the set of allowed strategies to those with acceptable risk.

To this end, we use the notion of *conditional value-at-risk* (CVaR). CVaR is a well-studied and established measure of risk in the area of finance and operations research, many desirable properties have been proven for it [Art+99; RU00; RU02]. Despite a rather simple intuition, the precise definition of CVaR for general random variables is surprisingly involved. We thus only provide a brief overview and direct to [KM18, Section 3] for an in-depth discussion.

Before we define CVaR, we need to introduce the *value-at-risk* (VaR). Both VaR and CVaR are parametrized by a threshold probability $p \in [0, 1]$. Given such a p, the VaR of a random variable X equals a value v such that a fraction of p outcomes yield a value smaller than v under X, i.e. it is the worst p-quantile of X. See Figure 6.1 for a graphical illustration of VaR and CVaR, taken from [KM18]. With p chosen to be 0.2, the VaR is the value such that the shaded area left of it equals 0.2. This value v supposedly describes 'What is a typical bad case?'. More formally, given a random variable X, $\mathrm{VaR}_p(X)$ is a value v such that $\mathbb{P}[X \leq v] = p$.[1] However, as suggested by the figure, VaR ignores the magnitude of outliers and potentially is very sensitive to small changes in X or p. As such, it has been called 'seductive, but dangerous' and 'not sufficient to control risk' [Bed95]. To smooth out these issues of VaR, CVaR denotes the expectation of all outcomes which are worse than this 'bad case'. Intuitively, this answers the question 'What to expect from a typical bad case?'. Formally, we set

[1] We intentionally omit several arising corner cases for the sake of readability.

$\mathrm{CVaR}_p(X) = \mathbb{E}[X \mid X \leq \mathrm{VaR}_p(X)]$. As such, CVaR incorporates all possible 'bad' outcomes without being overly pessimistic. Considering the 'overcharging' example from before, we obtain a CVaR of 0 MW for $p = 0.05$ and a CVaR of 60 MW for $p = 0.1$. As an interesting observation, note that CVaR smoothly interpolates between worst-case analysis ($p = 0$) and expectation ($p = 1$).

In order to fit CVaR in our framework of Section 2.2, observe that CVaR essentially offers a way of aggregating the value distribution obtained under a particular strategy. For example, given a weighted reachability objective specified by the target set T and reward function rew together with a threshold $p \in [0, 1]$, we might be interested in $\mathcal{V}^{\pi}_{\mathsf{rew}\Diamond T,\mathrm{CVaR}_p}$ instead of $\mathcal{V}^{\pi}_{\mathsf{rew}\Diamond T,\mathbb{E}}$, i.e. the CVaR of the values reached under π with threshold p instead of the expectation. Note that the expectation fully describes the distribution for qualitative functions such as reachability of LTL. Thus, we only consider CVaR in combination with quantitative performance measures, e.g., weighted reachability or mean payoff.

6.2 Model Checking CVaR in MDP

In this section, we outline our results regarding CVaR-aware model checking of weighted reachability and mean payoff. In summary, our contributions are threefold:

- We show that the single-dimensional case is solvable in polynomial time.
- We prove NP-hardness of the multi-dimensional case. However, the problem is polynomial for any fixed dimension.
- We exactly characterize the structure of optimal strategies.

Let us first describe our problem statement in more detail. Since we are interested in computational complexity, we consider the decision variant of our problem instead of optimization. For example, we may consider the question 'Is $\mathcal{V}^{\max}_{\mathsf{rew}\Diamond T,\mathrm{CVaR}_p}(\hat{s}) \geq \mathsf{c}$?' instead of asking for the precise value. However, our solution approach is based on a linear program and can directly be converted into an optimization procedure.

Formally, we are interested in the following problem.

Problem Statement

Let $\mathcal{M} = (S, \mathit{Act}, \mathit{Av}, \Delta)$ be an MDP, $\mathsf{perf} \in \{\mathsf{rew}\Diamond T, \mathsf{rew}\text{-}\mathrm{mp}\}$ either a weighted reachability or mean payoff objective, e the expectation threshold, p the CVaR probability, and c the CVaR threshold. Decide whether there exists a strategy $\pi \in \Pi_{\mathcal{M}}$ satisfying the constraints

$$\mathcal{V}^{\pi}_{\mathsf{perf},\mathbb{E}}(\hat{s}) \geq \mathsf{e} \quad \text{and} \quad \mathcal{V}^{\pi}_{\mathsf{perf},\mathrm{CVaR}_p}(\hat{s}) \geq \mathsf{c}.$$

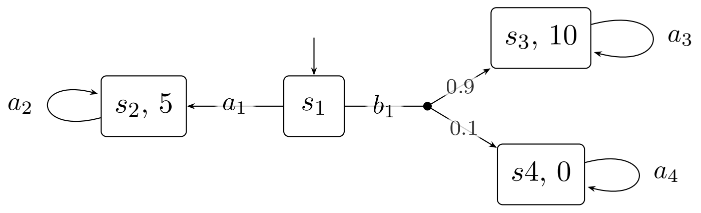


Figure 6.2: Example MDP used to demonstrate CVaR optimization of weighted reachability.

We also consider the multi-dimensional variant of this problem, mainly for the sake of complexity analysis. In this scenario, rew yields a d-dimensional reward and we replace the thresholds e and c and the threshold probability p with d-dimensional vectors each. The resulting query then asks whether there exists a strategy such that the above equation is satisfied in every dimension.

For an example of the single-dimensional case, consider the MDP of Figure 6.2. The left action a_1 yields an expectation of 5. Since there is no randomness associated with this outcome, the CVaR equals 5 for each p, too. The right action b_1 yields a much higher expectation of 9, however, for example, a CVaR of 0 for $p = 0.1$ and 5 for $p = 0.2$. If we consider $\mathsf{e} = 6$, $\mathsf{c} = 2$, and $p = 0.05$, the answer to our risk-aware query is 'yes': The strategy $\pi(s_1) = \{a_1 \mapsto \frac{3}{4}, b_1 \mapsto \frac{1}{4}\}$ achieves a value distribution of $\{0 \mapsto \frac{1}{40}, 5 \mapsto \frac{3}{4}, 10 \mapsto \frac{9}{40}\}$. The expected reward is $6 \geq \mathsf{e}$ and the CVaR with $p = 0.05$ is $2.5 \geq \mathsf{c}$. Observe that this already shows that randomizing strategies are necessary for such joint queries, since neither purely choosing a_1 nor b_1 satisfies the constraints.

We show that deterministic memoryless strategies are sufficient if we only have an expectation or CVaR constraint. Moreover, we show that randomization and one bit of memory is enough for queries with both expectation and CVaR constraints for both weighted reachability and mean payoff. In particular, memoryless strategies are sufficient for weighted reachability under mild assumptions which can be lifted by some pre-computation steps, adding one bit of memory to the strategy. Memoryless strategies allow for an intuitive characterization by an LP, i.e. every feasible solution of this LP corresponds to a memoryless strategy and vice versa (see, e.g., [CKK17]). This LP characterizes strategies by the 'flow' they achieve, i.e. how the probability mass of all runs distributes over the states.

To discuss our solution approach, first assume that we know the value-at-risk. Then, we can add constraints to the flow LP such that the flow is split according to the VaR. In particular, we require that a flow with mass p ends up in states achieving a value less than or equal to the given VaR. Dually, we require that the remaining $1 - p$ fraction ends up in states with reward at least as good as the VaR. The expectation and CVaR constraint can then easily be encoded into this LP. We show that there exists a strategy achieving the given VaR and satisfying the expectation and CVaR constraints iff the LP has a solution. See [KM18, Figure 4] for the full LP. As final piece of the puzzle, observe that the number of different values we can obtain under weighted reachability is bounded by the number of states: The reward function rew assigns one value to each state. By definition, VaR can only take one of these values. Thus, we can simply test for each possible VaR value whether we can find a strategy satisfying our constraints. Since there are only linearly many values to test and deciding an LP takes polynomial time, we obtain an overall polynomial complexity.

For mean payoff, we again use the reduction explained in Section 3.2.2. Intuitively, once we decide to 'stay' in a MEC, there is no harm in obtaining the best possible value, even in the presence of CVaR constraints. We formally prove this intuition in [KM18, Section 3] by showing that, similar to expectation, CVaR is monotone w.r.t. *stochastic dominance*, i.e. when we improve performance then the obtained CVaR does not decrease. With this, we again compute the mean payoff of each MEC separately by, e.g., linear programming, build the weighted MEC quotient, and solve the associated weighted reachability problem thereof. Together, we obtain polynomial time complexity.

This idea immediately yields an EXPTIME algorithm and proof of NP containment for weighted reachability when considering the multi-dimensional problem. By guessing the VaR of each dimension, we can verify a solution in polynomial time, and there are only exponentially many guesses. Note that for a fixed number of dimensions, we again only have polynomially many guesses. Interestingly, even when we only consider a CVaR constraint, the decision problem is NP-hard. Recall the MDP from Figure 6.1. If we take $\mathsf{c} = 5$ and $p = 0.2$, *only* purely choosing either a_1 or b_1 satisfies the query. Any strategy randomizing over a_1 and b_1 obtains a CVaR strictly less than 5. Essentially, this allows us to force a deterministic choice at a given state with a CVaR constraint. Through some further modelling we obtain a reduction from 3-SAT [KM18, Theorem 6.2] by encoding the deterministic value assignment for each variable in a simple gadget.

Unfortunately, multi-dimensional mean payoff proves to be a challenge. In particular, proving NP-completeness for mean payoff with CVaR constraints remains an open question, since it is unclear whether the optimal VaR values in each dimension are of polynomial size. We prove PSPACE containment through the existential theory of the reals, quantifying the VaR for each dimension. However, we strongly conjecture that mean payoff also is NP-complete.

6.3 Conclusion

We have demonstrated that risk-aware control of probabilistic systems is possible without an increase in computational complexity. Even multi-dimensional queries can be tackled. Hence, there seems to be no fundamental reason to neglect risk analysis. A first obvious goal for future work is an actual implementation and evaluation of risk-aware strategies. Moreover, as we argued in Chapter 3, LP based solutions often are underwhelming in terms of performance. A dynamic programming based implementation may be useful.

Since this quantified view of risk is novel, even the most basic questions remain open. For example, we do not know how 'costly' risk-aware control is in practice or how sensible the resulting strategies really are. In particular, it might turn out that CVaR optimal strategies are not very aligned with the human understanding of risk when considering probabilistic systems. Moreover, we might also be interested in completely different models of risk. For example, recalling the power plant model, our approach does not allow us to analyse the deviation of a single run from its mean payoff. Here, it might be more desirable to have a constant power output of 49 MW compared to a random alternation between 0 MW and 100 MW.

Bibliography

[Art+99] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber and David Heath. 'Coherent measures of risk'. In: *Mathematical finance* 9.3 (1999), pp. 203–228.

[Ash+17] Pranav Ashok, Krishnendu Chatterjee, Przemyslaw Daca, Jan Křetínský and Tobias Meggendorfer. 'Value Iteration for Long-Run Average Reward in Markov Decision Processes'. In: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I.* Ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10426. Lecture Notes in Computer Science. Springer, 2017, pp. 201–221. DOI: `10.1007/978-3-319-63387-9\_10`. URL: `https://doi.org/10.1007/978-3-319-63387-9%5C_10`.

[AT04] Rajeev Alur and Salvatore La Torre. 'Deterministic generators and games for Ltl fragments'. In: *ACM Trans. Comput. Log.* 5.1 (2004), pp. 1–25. DOI: `10.1145/963927.963928`. URL: `https://doi.org/10.1145/963927.963928`.

[Bab+15] Tomás Babiak, Frantisek Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker and Jan Strejcek. 'The Hanoi Omega-Automata Format'. In: *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I.* Ed. by Daniel Kroening and Corina S. Pasareanu. Vol. 9206. Lecture Notes in Computer Science. Springer, 2015, pp. 479–486. DOI: `10.1007/978-3-319-21690-4\_31`. URL: `https://doi.org/10.1007/978-3-319-21690-4%5C_31`.

[Bai+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: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I.* Ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10426. Lecture Notes in Computer Science. Springer, 2017, pp. 160–180. DOI: `10.1007/978-3-319-63387-9\_8`. URL: `https://doi.org/10.1007/978-3-319-63387-9%5C_8`.

[Bed95] Tanya Styblo Beder. 'VAR: Seductive but dangerous'. In: *Financial Analysts Journal* 51.5 (1995), pp. 12–24.

[Bel66] Richard Bellman. 'Dynamic programming'. In: *Science* 153.3731 (1966), pp. 34–37.

[Bil08] Patrick Billingsley. *Probability and measure.* John Wiley & Sons, 2008.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008. ISBN: 978-0-262-02649-9.

[Brá+14] Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Křetínský, Marta Z. Kwiatkowska, David Parker and Mateusz Ujma. 'Verification of Markov Decision Processes Using Learning Algorithms'. In: *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*. Ed. by Franck Cassez and Jean-François Raskin. Vol. 8837. Lecture Notes in Computer Science. Springer, 2014, pp. 98–114. DOI: `10.1007/978-3-319-11936-6\_8`. URL: `https://doi.org/10.1007/978-3-319-11936-6%5C_8`.

[Brá+15] Tomás Brázdil, Krishnendu Chatterjee, Vojtech Forejt and Antonín Kucera. 'MultiGain: A Controller Synthesis Tool for MDPs with Multiple Mean-Payoff Objectives'. In: *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*. Ed. by Christel Baier and Cesare Tinelli. Vol. 9035. Lecture Notes in Computer Science. Springer, 2015, pp. 181–187. DOI: `10.1007/978-3-662-46681-0\_12`. URL: `https://doi.org/10.1007/978-3-662-46681-0%5C_12`.

[CGK13] Krishnendu Chatterjee, Andreas Gaiser and Jan Křetínský. 'Automata with Generalized Rabin Pairs for Probabilistic Model Checking and LTL Synthesis'. In: *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*. Ed. by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013, pp. 559–575. DOI: `10.1007/978-3-642-39799-8\_37`. URL: `https://doi.org/10.1007/978-3-642-39799-8%5C_37`.

[CH11] Krishnendu Chatterjee and Monika Henzinger. 'Faster and Dynamic Algorithms for Maximal End-Component Decomposition and Related Graph Problems in Probabilistic Verification'. In: *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*. Ed. by Dana Randall. SIAM, 2011, pp. 1318–1336. DOI: `10.1137/1.9781611973082.101`. URL: `https://doi.org/10.1137/1.9781611973082.101`.

[CH12] Krishnendu Chatterjee and Monika Henzinger. 'An $O(n^2)$ time algorithm for alternating Büchi games'. In: *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*. Ed. by Yuval Rabani. SIAM, 2012, pp. 1386–1399. DOI: `10.1137/1.9781611973099.109`. URL: `https://doi.org/10.1137/1.9781611973099.109`.

[CH14] Krishnendu Chatterjee and Monika Henzinger. 'Efficient and Dynamic Algorithms for Alternating Büchi Games and Maximal End-Component Decomposition'. In: *J. ACM* 61.3 (2014), 15:1–15:40. DOI: `10.1145/2597631`. URL: `https://doi.org/10.1145/2597631`.

[CKK17] Krishnendu Chatterjee, Zuzana Kretínská and Jan Křetínský. 'Unifying Two Views on Multiple Mean-Payoff Objectives in Markov Decision Processes'. In: *Logical Methods in Computer Science* 13.2 (2017). DOI: `10.23638/LMCS-13(2:15)2017`. URL: `https://doi.org/10.23638/LMCS-13(2:15)2017`.

[Cou78] Patrick Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes.* 1978. URL: `https://tel.archives-ouvertes.fr/tel-00288657`.

[CY95] Costas Courcoubetis and Mihalis Yannakakis. 'The Complexity of Probabilistic Verification'. In: *J. ACM* 42.4 (1995), pp. 857–907. DOI: `10.1145/210332.210339`. URL: `https://doi.org/10.1145/210332.210339`.

[De 97] Luca De Alfaro. *Formal verification of probabilistic systems.* 1601. Citeseer, 1997.

[Deh+17] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen and Matthias Volk. 'A Storm is Coming: A Modern Probabilistic Model Checker'. In: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II.* Ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10427. Lecture Notes in Computer Science. Springer, 2017, pp. 592–600. DOI: `10.1007/978-3-319-63390-9\_31`. URL: `https://doi.org/10.1007/978-3-319-63390-9%5C_31`.

[DG08] Volker Diekert and Paul Gastin. 'First-order definable languages'. In: *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas].* Ed. by Jörg Flum, Erich Grädel and Thomas Wilke. Vol. 2. Texts in Logic and Games. Amsterdam University Press, 2008, pp. 261–306.

[Dur+16] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault and Laurent Xu. 'Spot 2.0 - A Framework for LTL and \omega -Automata Manipulation'. In: *Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings.* Ed. by Cyrille Artho, Axel Legay and Doron Peled. Vol. 9938. Lecture Notes in Computer Science. 2016, pp. 122–129. DOI: `10.1007/978-3-319-46520-3\_8`. URL: `https://doi.org/10.1007/978-3-319-46520-3%5C_8`.

[EKS16] Javier Esparza, Jan Křetínský and Salomon Sickert. 'From LTL to deterministic automata - A safraless compositional approach'. In: *Formal Methods Syst. Des.* 49.3 (2016), pp. 219–271. DOI: `10.1007/s10703-016-0259-2`. URL: `https://doi.org/10.1007/s10703-016-0259-2`.

[For+11] Vojtech Forejt, Marta Z. Kwiatkowska, Gethin Norman and David Parker. 'Automated Verification Techniques for Probabilistic Systems'. In: *Formal Methods for Eternal Networked Software Systems - 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures*. Ed. by Marco Bernardo and Valérie Issarny. Vol. 6659. Lecture Notes in Computer Science. Springer, 2011, pp. 53–113. DOI: `10.1007/978-3-642-21455-4\_3`. URL: `https://doi.org/10.1007/978-3-642-21455-4%5C_3`.

[FV96] Jerzy Filar and Koos Vrieze. 'Competitive Markov decision processes'. In: (1996).

[HM14] Serge Haddad and Benjamin Monmege. 'Reachability in MDPs: Refining Convergence of Value Iteration'. In: *Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings*. Ed. by Joël Ouaknine, Igor Potapov and James Worrell. Vol. 8762. Lecture Notes in Computer Science. Springer, 2014, pp. 125–137. DOI: `10.1007/978-3-319-11439-2\_10`. URL: `https://doi.org/10.1007/978-3-319-11439-2%5C_10`.

[How60] Ronald A Howard. 'Dynamic programming and markov processes.' In: (1960).

[Jac+19] Swen Jacobs, Roderick Bloem, Maximilien Colange, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Michael Luttenberger, Philipp J. Meyer, Thibaud Michaud, Mouhammad Sakr, Salomon Sickert, Leander Tentrup and Adam Walker. 'The 5th Reactive Synthesis Competition (SYNTCOMP 2018): Benchmarks, Participants & Results'. In: *CoRR* abs/1904.07736 (2019). arXiv: `1904.07736`. URL: `http://arxiv.org/abs/1904.07736`.

[Kar84] Narendra Karmarkar. 'A new polynomial-time algorithm for linear programming'. In: *Combinatorica* 4.4 (1984), pp. 373–396. DOI: `10.1007/BF02579150`. URL: `https://doi.org/10.1007/BF02579150`.

[KE12] Jan Křetínský and Javier Esparza. 'Deterministic Automata for the (F, G)-Fragment of LTL'. In: *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*. 2012, pp. 7–22. DOI: `10.1007/978-3-642-31424-7_7`. URL: `http://dx.doi.org/10.1007/978-3-642-31424-7_7`.

[Kha79] Leonid G Khachiyan. 'A polynomial algorithm in linear programming'. In: *Doklady Academii Nauk SSSR*. Vol. 244. 1979, pp. 1093–1096.

[KK14] Zuzana Komárková and Jan Křetínský. 'Rabinizer 3: Safraless Translation of LTL to Small Deterministic Automata'. In: *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*. Ed. by Franck Cassez and Jean-François Raskin. Vol. 8837. Lecture Notes in Computer

Science. Springer, 2014, pp. 235–241. DOI: `10.1007/978-3-319-11936-6\_17`. URL: `https://doi.org/10.1007/978-3-319-11936-6%5C_17`.

[KM] Jan Křetínský and Tobias Meggendorfer. 'Of Cores: A Partial-Exploration Framework for Markov Decision Processes'. In: *Logical Methods in Computer Science* Selected Papers of the 30th International Conference on Concurrency Theory (CONCUR 2019) (). To appear.

[KM17] Jan Křetínský and Tobias Meggendorfer. 'Efficient Strategy Iteration for Mean Payoff in Markov Decision Processes'. In: *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings.* Ed. by Deepak D'Souza and K. Narayan Kumar. Vol. 10482. Lecture Notes in Computer Science. Springer, 2017, pp. 380–399. DOI: `10.1007/978-3-319-68167-2\_25`. URL: `https://doi.org/10.1007/978-3-319-68167-2%5C_25`.

[KM18] Jan Křetínský and Tobias Meggendorfer. 'Conditional Value-at-Risk for Reachability and Mean Payoff in Markov Decision Processes'. In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018.* Ed. by Anuj Dawar and Erich Grädel. ACM, 2018, pp. 609–618. DOI: `10.1145/3209108.3209176`. URL: `https://doi.org/10.1145/3209108.3209176`.

[KM19] Jan Křetínský and Tobias Meggendorfer. 'Of Cores: A Partial-Exploration Framework for Markov Decision Processes'. In: *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands.* Ed. by Wan Fokkink and Rob van Glabbeek. Vol. 140. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 5:1–5:17. DOI: `10.4230/LIPIcs.CONCUR.2019.5`. URL: `https://doi.org/10.4230/LIPIcs.CONCUR.2019.5`.

[KMM19] Jan Křetínský, Alexander Manta and Tobias Meggendorfer. 'Semantic Labelling and Learning for Parity Game Solving in LTL Synthesis'. In: *Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings.* Ed. by Yu-Fang Chen, Chih-Hong Cheng and Javier Esparza. Vol. 11781. Lecture Notes in Computer Science. Springer, 2019, pp. 404–422. DOI: `10.1007/978-3-030-31784-3\_24`. URL: `https://doi.org/10.1007/978-3-030-31784-3%5C_24`.

[KMS18] Jan Křetínský, Tobias Meggendorfer and Salomon Sickert. 'Owl: A Library for ω-Words, Automata, and LTL'. In: *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings.* Ed. by Shuvendu K. Lahiri and Chao Wang. Vol. 11138. Lecture Notes in Computer Science. Springer, 2018, pp. 543–550. DOI: `10.1007/978-3-030-01090-4\_34`. URL: `https://doi.org/10.1007/978-3-030-01090-4%5C_34`.

[KNP11] Marta Z. Kwiatkowska, Gethin Norman and David Parker. 'PRISM 4.0: Verification of Probabilistic Real-Time Systems'. In: *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings.* Ed. by Ganesh Gopalakrishnan and Shaz Qadeer. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 585–591. DOI: `10.1007/978-3-642-22110-1\_47`. URL: `https://doi.org/10.1007/978-3-642-22110-1%5C_47`.

[Kře+17] Jan Křetínský, Tobias Meggendorfer, Clara Waldmann and Maximilian Weininger. 'Index Appearance Record for Transforming Rabin Automata into Parity Automata'. In: *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I.* Ed. by Axel Legay and Tiziana Margaria. Vol. 10205. Lecture Notes in Computer Science. 2017, pp. 443–460. DOI: `10.1007/978-3-662-54577-5\_26`. URL: `https://doi.org/10.1007/978-3-662-54577-5%5C_26`.

[Kře+18] Jan Křetínský, Tobias Meggendorfer, Salomon Sickert and Christopher Ziegler. 'Rabinizer 4: From LTL to Your Favourite Deterministic Automaton'. In: *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I.* Ed. by Hana Chockler and Georg Weissenbacher. Vol. 10981. Lecture Notes in Computer Science. Springer, 2018, pp. 567–577. DOI: `10.1007/978-3-319-96145-3\_30`. URL: `https://doi.org/10.1007/978-3-319-96145-3%5C_30`.

[Meg17] Tobias Meggendorfer. *JBDD: A Java BDD Library.* `https://github.com/incaseoftrouble/jbdd`. 2017.

[MS17] David Müller and Salomon Sickert. 'LTL to Deterministic Emerson-Lei Automata'. In: *Proceedings Eighth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2017, Roma, Italy, 20-22 September 2017.* Ed. by Patricia Bouyer, Andrea Orlandini and Pierluigi San Pietro. Vol. 256. EPTCS. 2017, pp. 180–194. DOI: `10.4204/EPTCS.256.13`. URL: `https://doi.org/10.4204/EPTCS.256.13`.

[MSL18] Philipp J. Meyer, Salomon Sickert and Michael Luttenberger. 'Strix: Explicit Reactive Synthesis Strikes Back!' In: *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I.* Ed. by Hana Chockler and Georg Weissenbacher. Vol. 10981. Lecture Notes in Computer Science. Springer, 2018, pp. 578–586. DOI: `10.1007/978-3-319-96145-3\_31`. URL: `https://doi.org/10.1007/978-3-319-96145-3%5C_31`.

[Mül20] Alexander Christian Müller. 'Proving Noninterference in Multi-Agent Systems'. Dissertation. München: Technische Universität München, 2020.

[Pnu77] Amir Pnueli. 'The Temporal Logic of Programs'. In: *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977.* 1977, pp. 46–57. DOI: `10.1109/SFCS.1977.32`. URL: `http://dx.doi.org/10.1109/SFCS.1977.32`.

[Put94] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley Series in Probability and Statistics. Wiley, 1994. ISBN: 978-0-47161977-2. DOI: `10.1002/9780470316887`. URL: `https://doi.org/10.1002/9780470316887`.

[Rab69] Michael O Rabin. 'Decidability of second-order theories and automata on infinite trees'. In: *Transactions of the american Mathematical Society* 141 (1969), pp. 1–35.

[RU00] R Tyrrell Rockafellar and Stanislav Uryasev. 'Optimization of conditional value-at-risk'. In: *Journal of risk* 2 (2000), pp. 21–42.

[RU02] R Tyrrell Rockafellar and Stanislav Uryasev. 'Conditional value-at-risk for general loss distributions'. In: *Journal of banking & finance* 26.7 (2002), pp. 1443–1471.

[Sch99] Alexander Schrijver. *Theory of linear and integer programming.* Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999. ISBN: 978-0-471-98232-6.

[Tar72] Robert Endre Tarjan. 'Depth-First Search and Linear Graph Algorithms'. In: *SIAM J. Comput.* 1.2 (1972), pp. 146–160. DOI: `10.1137/0201010`. URL: `https://doi.org/10.1137/0201010`.

[VW86] Moshe Y. Vardi and Pierre Wolper. 'An Automata-Theoretic Approach to Automatic Program Verification (Preliminary Report)'. In: *Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986.* IEEE Computer Society, 1986, pp. 332–344.

[Whi85] Douglas J White. 'Real applications of Markov decision processes'. In: *Interfaces* 15.6 (1985), pp. 73–83.

[Whi88] Douglas J White. 'Further real applications of Markov decision processes'. In: *Interfaces* 18.5 (1988), pp. 55–61.

[Whi93] Douglas J White. 'A survey of applications of Markov decision processes'. In: *Journal of the operational research society* 44.11 (1993), pp. 1073–1096.

Appendices

I Appended Papers

A Value Iteration for Long-Run Average Reward in Markov Decision Processes. CAV 2017

This section has been published as **peer-reviewed conference paper**.

Pranav Ashok, Krishnendu Chatterjee, Przemyslaw Daca, Jan Křetínský and Tobias Meggendorfer. 'Value Iteration for Long-Run Average Reward in Markov Decision Processes'. In: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I.* ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10426. Lecture Notes in Computer Science. Springer, 2017, pp. 201–221. DOI: `10.1007/978-3-319-63387-9\_10`. URL: `https://doi.org/10.1007/978-3-319-63387-9%5C_10`

Synopsis We consider the problem of optimizing *mean payoff* (also known as *long run average reward*) objectives on Markov decision processes. *Value iteration* (VI) is one of the simplest yet most efficient algorithms for the analysis of a wide range of objectives on MDP. Unfortunately, naively extending VI to mean payoff does not yield a *stopping criterion*, rendering this approach useless. In this work, we (i) refute a long standing conjecture about such a stopping criterion, (ii) present an alternative stopping criterion, based on topological pre-processing, and (iii) combine our new approach with the partial-exploration pattern introduced by [Brá+14], arriving at a sophisticated on-the-fly algorithm. This algorithm automatically identifies promising areas of the systems' state-space and focusses its computational effort on these regions.

Contributions of the thesis author Composition of Sections 3, 4, and 5. Discussion and revision of the entire manuscript. Discussion of all results and proofs presented in the paper. Development of proofs in Section 3.1 and 3.2. Leading role in the design and implementation of the presented tool. Creation of the software artefact for conference submission.

Value Iteration for Long-Run Average Reward in Markov Decision Processes

Pranav Ashok[1], Krishnendu Chatterjee[2], Przemysław Daca[2], Jan Křetínský[1(✉)], and Tobias Meggendorfer[1]

[1] Technical University of Munich, Munich, Germany
jan.kretinsky@gmail.com
[2] IST Austria, Klosterneuburg, Austria

Abstract. Markov decision processes (MDPs) are standard models for probabilistic systems with non-deterministic behaviours. Long-run average rewards provide a mathematically elegant formalism for expressing long term performance. Value iteration (VI) is one of the simplest and most efficient algorithmic approaches to MDPs with other properties, such as reachability objectives. Unfortunately, a naive extension of VI does not work for MDPs with long-run average rewards, as there is no known stopping criterion. In this work our contributions are threefold. (1) We refute a conjecture related to stopping criteria for MDPs with long-run average rewards. (2) We present two practical algorithms for MDPs with long-run average rewards based on VI. First, we show that a combination of applying VI locally for each maximal end-component (MEC) and VI for reachability objectives can provide approximation guarantees. Second, extending the above approach with a simulation-guided on-demand variant of VI, we present an anytime algorithm that is able to deal with very large models. (3) Finally, we present experimental results showing that our methods significantly outperform the standard approaches on several benchmarks.

1 Introduction

The analysis of probabilistic systems arises in diverse application contexts of computer science, e.g. analysis of randomized communication and security protocols, stochastic distributed systems, biological systems, and robot planning, to name a few. The standard model for the analysis of probabilistic systems that exhibit both probabilistic and non-deterministic behaviour are *Markov decision processes (MDPs)* [How60,FV97,Put94]. An MDP consists of a finite set of states, a finite set of actions, representing the non-deterministic choices, and

This work is partially supported by the Vienna Science and Technology Fund (WWTF) ICT15-003, the Austrian Science Fund (FWF) NFN grant No. S11407-N23 (RiSE/SHiNE), the ERC Starting grant (279307: Graph Games), the German Research Foundation (DFG) project “Verified Model Checkers”, the TUM International Graduate School of Science and Engineering (IGSSE) project PARSEC, and the Czech Science Foundation grant No. 15-17564S.

R. Majumdar and V. Kunčak (Eds.): CAV 2017, Part I, LNCS 10426, pp. 201–221, 2017.
DOI: 10.1007/978-3-319-63387-9_10

a transition function that given a state and an action gives the probability distribution over the successor states. In verification, MDPs are used as models for e.g. concurrent probabilistic systems [CY95] or probabilistic systems operating in open environments [Seg95], and are applied in a wide range of applications [BK08,KNP11].

Long-Run Average Reward. A *payoff* function in an MDP maps every infinite path (infinite sequence of state-action pairs) to a real value. One of the most well-studied and mathematically elegant payoff functions is the *long-run average reward* (also known as *mean-payoff* or *limit-average reward*, *steady-state reward* or simply *average reward*), where every state-action pair is assigned a real-valued reward, and the payoff of an infinite path is the long-run average of the rewards on the path [FV97,Put94]. Beyond the elegance, the long-run average reward is standard to model performance properties, such as the average delay between requests and corresponding grants, average rate of a particular event, etc. Therefore, determining the maximal or minimal expected long-run average reward of an MDP is a basic and fundamental problem in the quantitative analysis of probabilistic systems.

Classical Algorithms. A *strategy* (also known as *policy* or *scheduler*) in an MDP specifies how the non-deterministic choices of actions are resolved in every state. The *value* at a state is the maximal expected payoff that can be guaranteed among all strategies. The values of states in MDPs with payoff defined as the long-run average reward can be computed in polynomial-time using linear programming [FV97,Put94]. The corresponding linear program is quite involved though. The number of variables is proportional to the number of state-action pairs and the overall size of the program is linear in the number of transitions (hence potentially quadratic in the number of actions). While the linear programming approach gives a polynomial-time solution, it is quite slow in practice and does not scale to larger MDPs. Besides linear programming, other techniques are considered for MDPs, such as dynamic-programming through strategy iteration or value iteration [Put94, Chap. 9].

Value Iteration. A generic approach that works very well in practice for MDPs with other payoff functions is *value iteration (VI).* Intuitively, a particular one-step operator is applied iteratively and the crux is to show that this iterative computation converges to the correct solution (i.e. the value). The key advantages of VI are the following:

1. *Simplicity.* VI provides a very simple and intuitive dynamic-programming algorithm which is easy to adapt and extend.
2. *Efficiency.* For several other payoff functions, such as finite-horizon rewards (instantaneous or cumulative reward) or reachability objectives, applying the concept of VI yields a very efficient solution method. In fact, in most well-known tools such as PRISM [KNP11], value iteration performs much better than linear programming methods for reachability objectives.

3. *Scalability.* The simplicity and flexibility of VI allows for several improvements and adaptations of the idea, further increasing its performance and enabling quick processing of very large MDPs. For example, when considering reachability objectives, [PGT03] present point-based value-iteration (PBVI), applying the iteration operator only to a part of the state space, and [MLG05] introduce bounded real-time dynamic programming (BRTDP), where again only a fraction of the state space is explored based on partial strategies. Both of these approaches are simulation-guided, where simulations are used to decide how to explore the state space. The difference is that the former follows an offline computation, while the latter is online. Both scale well to large MDPs and use VI as the basic idea to build upon.

Value Iteration for Long-Run Average Reward. While VI is standard for reachability objectives or finite-horizon rewards, it does not work for general MDPs with long-run average reward. The two key problems pointed out in [Put94, Sects. 8.5, 9.4] are as follows: (a) if the MDP has some periodicity property, then VI does not converge; and (b) for general MDPs there are neither bounds on the speed of convergence nor stopping criteria to determine when the iteration can be stopped to guarantee approximation of the value. The first problem can be handled by adding self-loop transitions [Put94, Sect. 8.5.4]. However, the second problem is conceptually more challenging, and a solution is conjectured in [Put94, Sect. 9.4.2].

Our Contribution. In this work, our contributions are related to value iteration for MDPs with long-run average reward, they range from conceptual clarification to practical algorithms and experimental results. The details of our contributions are as follows.

– *Conceptual clarification.* We first present an example to refute the conjecture of [Put94, Sect. 9.4.2], showing that the approach proposed there does not suffice for VI on MDPs with long-run average reward.
– *Practical approaches.* We develop, in two steps, practical algorithms instantiating VI for approximating values in MDPs with long-run average reward. Our algorithms take advantage of the notion of maximal end-components (MECs) in MDPs. Intuitively, MECs for MDPs are conceptually similar to strongly connected components (SCCs) for graphs and recurrent classes for Markov chains. We exploit these MECs to arrive at our two methods:
    1. The first variant applies VI locally to each MEC in order to obtain an approximation of the values within the MEC. After the approximation in every MEC, we apply VI to solve a reachability problem in a modified MDP with collapsed MECs. We show that this simple combination of VI approaches ensures guarantees on the approximation of the value.
    2. We then build on the approach above to present a simulation-guided variant of VI. In this case, the approximation of values for each MEC and the reachability objectives are done at the same time using VI. For the reachability objective a BRDTP-style VI (similar to [BCC+14]) is

applied, and within MECs VI is applied on-demand (i.e. only when there is a requirement for more precise value bounds). The resulting algorithm furthermore is an *anytime* algorithm, i.e. it can be stopped at any time and give an upper and lower bounds on the result.

- *Experimental results.* We compare our new algorithms to the state-of-the-art tool MultiGain [BCFK15] on various models. The experiments show that MultiGain is vastly outperformed by our methods on nearly every model. Furthermore, we compare several variants of our methods and investigate the different domains of applicability.

In summary, we present the first instantiation of VI for general MDPs with long-run average reward. Moreover, we extend it with a simulation-based approach to obtain an efficient algorithm for large MDPs. Finally, we present experimental results demonstrating that these methods provide significant improvements over existing ones.

Further Related Work. There is a number of techniques to compute or approximate the long-run average reward in MDPs [Put94, How60, Vei66], ranging from linear programming to value iteration to strategy iteration. Symbolic and explicit techniques based on strategy iteration are combined in [WBB+10]. Further, the more general problem of MDPs with multiple long-run average rewards was first considered in [Cha07], a complete picture was presented in [BBC+14, CKK15] and partially implemented in [BCFK15]. The extension of our approach to multiple long-run average rewards, or combination of expectation and variance [BCFK13], are interesting directions for future work. Finally, VI for MDPs with guarantees for reachability objectives was considered in [BCC+14, HM14].

Proofs and supplementary material can be found in [ACD+17].

2 Preliminaries

2.1 Markov Decision Processes

A *probability distribution* on a finite set X is a mapping $\rho : X \mapsto [0, 1]$, such that $\sum_{x \in X} \rho(x) = 1$. We denote by $\mathcal{D}(X)$ the set of all probability distributions on X. Further, the *support* of a probability distribution ρ is denoted by $\operatorname{supp}(\rho) = \{x \in X \mid \rho(x) > 0\}$.

Definition 1 (MDP). *A* Markov decision processes (MDP) *is a tuple of the form* $\mathcal{M} = (S, s_{init}, Act, \mathsf{Av}, \Delta, r)$*, where* S *is a finite set of* states, $s_{init} \in S$ *is the* initial *state,* Act *is a finite set of* actions, $\mathsf{Av} : S \to 2^{Act}$ *assigns to every state a set of* available *actions,* $\Delta : S \times Act \to \mathcal{D}(S)$ *is a* transition function *that given a state* s *and an action* $a \in \mathsf{Av}(s)$ *yields a probability distribution over successor states, and* $r : S \times Act \to \mathbb{R}^{\geq 0}$ *is a* reward function*, assigning rewards to state-action pairs.*

For ease of notation, we write $\varDelta(s,a,s')$ instead of $\varDelta(s,a)(s')$.

An *infinite path* ρ in an MDP is an infinite word $\rho = s_0 a_0 s_1 a_1 \dots \in (S \times Act)^\omega$, such that for every $i \in \mathbb{N}$, $a_i \in \mathsf{Av}(s_i)$ and $\varDelta(s_i, a_i, s_{i+1}) > 0$. A *finite path* $w = s_0 a_0 s_1 a_1 \dots s_n \in (S \times Act)^* \times S$ is a finite prefix of an infinite path.

A *strategy* on an MDP is a function $\pi : (S \times Act)^* \times S \to \mathcal{D}(Act)$, which given a finite path $w = s_0 a_0 s_1 a_1 \dots s_n$ yields a probability distribution $\pi(w) \in \mathcal{D}(\mathsf{Av}(s_n))$ on the actions to be taken next. We call a strategy *memoryless randomized* (or *stationary*) if it is of the form $\pi : S \to \mathcal{D}(Act)$, and *memoryless deterministic* (or *positional*) if it is of the form $\pi : S \to Act$. We denote the set of all strategies of an MDP by $\varPi$, and the set of all memoryless deterministic strategies by $\varPi^{\mathsf{MD}}$. Fixing a strategy π and an initial state s on an MDP $\mathcal{M}$ gives a unique probability measure $\mathbb{P}^\pi_{\mathcal{M},s}$ over infinite paths [Put94, Sect. 2.1.6]. The expected value of a random variable F is defined as $\mathbb{E}^\pi_{\mathcal{M},s}[F] = \int F \, d\mathbb{P}^\pi_{\mathcal{M},s}$. When the MDP is clear from the context, we drop the corresponding subscript and write $\mathbb{P}^\pi_s$ and $\mathbb{E}^\pi_s$ instead of $\mathbb{P}^\pi_{\mathcal{M},s}$ and $\mathbb{E}^\pi_{\mathcal{M},s}$, respectively.

End Components. 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 $\operatorname{supp}(\varDelta(s,a)) \subseteq T$, and (ii) for all $s, s' \in T$ there is a finite path $w = s a_0 \dots a_n s' \in (T \times A)^* \times T$, i.e. w starts in s, ends in s', stays inside T and only uses actions in A.[1] Intuitively, an end component describes a set of states for which a particular strategy exists such that all possible paths remain inside these states and all of those states are visited infinitely often almost surely. 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 $A \subseteq A'$. Given an MDP $\mathcal{M}$, the set of its MECs is denoted by $\mathsf{MEC}(\mathcal{M})$. With these definitions, every state of an MDP belongs to at most one MEC and each MDP has at least one MEC.

Using the concept of MECs, we recall the standard notion of a *MEC quotient* [dA97]. To obtain this quotient, all MECs are merged into a single representative state, while transitions between MECs are preserved. Intuitively, this abstracts the MDP to its essential infinite time behaviour.

Definition 2 (MEC quotient [dA97]). *Let $\mathcal{M} = (S, s_{init}, Act, \mathsf{Av}, \varDelta, r)$ be an MDP with MECs $\mathsf{MEC}(\mathcal{M}) = \{(T_1, A_1), \dots, (T_n, A_n)\}$. Further, define $\mathsf{MEC}_S = \bigcup_{i=1}^n T_i$ as the set of all states contained in some MEC. The* MEC quotient *of $\mathcal{M}$ is defined as the MDP $\widehat{\mathcal{M}} = (\widehat{S}, \widehat{s}_{init}, \widehat{Act}, \widehat{\mathsf{Av}}, \widehat{\varDelta}, \widehat{r})$, where:*

- *$\widehat{S} = S \setminus \mathsf{MEC}_S \cup \{\widehat{s}_1, \dots, \widehat{s}_n\}$,*
- *if for some T_i we have $s_{init} \in T_i$, then $\widehat{s}_{init} = \widehat{s}_i$, otherwise $\widehat{s}_{init} = s_{init}$,*
- *$\widehat{Act} = \{(s,a) \mid s \in S, a \in \mathsf{Av}(s)\}$,*

[1] This standard definition assumes that actions are unique for each state, i.e. $\mathsf{Av}(s) \cap \mathsf{Av}(s') = \emptyset$ for $s \neq s'$. The usual procedure of achieving this in general is to replace Act by $S \times Act$ and adapting Av, $\varDelta$, and r appropriately.

– *the available actions* $\widehat{\mathsf{Av}}$ *are defined as*

$$\forall s \in S \setminus \mathsf{MEC}_S.\ \widehat{\mathsf{Av}}(s) = \{(s,a) \mid a \in \mathsf{Av}(s)\}$$
$$\forall 1 \le i \le n.\ \widehat{\mathsf{Av}}(\widehat{s}_i) = \{(s,a) \mid s \in T_i \wedge a \in \mathsf{Av}(s) \setminus A_i\},$$

– *the transition function* $\widehat{\Delta}$ *is defined as follows. Let* $\widehat{s} \in \widehat{S}$ *be some state in the quotient and* $(s,a) \in \mathsf{Av}(\widehat{s})$ *an action available in* $\widehat{s}$*. Then*

$$\widehat{\Delta}(\widehat{s},(s,a),\widehat{s}') = \begin{cases} \sum_{s' \in T_j} \Delta(s,a,s') & \mathit{if} \widehat{s}' = \widehat{s}_j, \\ \Delta(s,a,\widehat{s}') & \mathit{otherwise, i.e.}\ \widehat{s}' \in S \setminus \mathsf{MEC}_S. \end{cases}$$

For the sake of readability, we omit the added self-loop transitions of the form $\Delta(\widehat{s}_i,(s,a),\widehat{s}_i)$ *with* $s \in T_i$ *and* $a \in A_i$ *from all figures.*

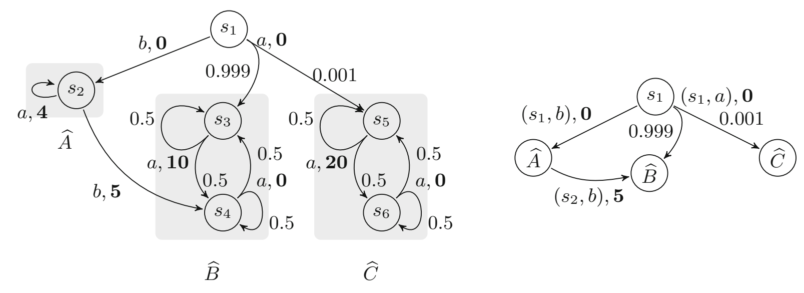
– *Finally, for* $\widehat{s} \in \widehat{S}$, $(s,a) \in \widehat{\mathsf{Av}}(\widehat{s})$*, we define* $\widehat{r}(s,(s,a)) = r(s,a)$.

Furthermore, we refer to $\widehat{s}_1,\dots,\widehat{s}_n$ *as* collapsed states *and identify them with the corresponding MECs.*

Example 1. Figure 1a shows an MDP with three MECs, $\widehat{A} = (\{s_2\},\{a\}), \widehat{B} = (\{s_3,s_4\},\{a\}), \widehat{C} = (\{s_5,s_6\},\{a\}))$. Its MEC quotient is shown in Fig. 1b. △

Remark 1. In general, the MEC quotient does not induce a DAG-structure, since there might be probabilistic transitions between MECs. Consider for example the MDP obtained by setting $\Delta(s_2,b,s_4) = \{s_1 \mapsto \frac{1}{2}, s_2 \mapsto \frac{1}{2}\}$ in the MDP of Fig. 1a. Its MEC quotient then has $\widehat{\Delta}(\widehat{A},(s_2,b)) = \{s_1 \mapsto \frac{1}{2}, \widehat{B} \mapsto \frac{1}{2}\}$.

Remark 2. The MEC decomposition of an MDP $\mathcal{M}$, i.e. the computation of $\mathsf{MEC}(\mathcal{M})$, can be achieved in polynomial time [CY95]. For improved algorithms on general MDPs and various special cases see [CH11, CH12, CH14, CL13].



(a) An MDP with three MECs. (b) The MEC quotient.

Fig. 1. An example of how the MEC quotient is constructed. By $a, \mathbf{r}$ we denote that the action a yields a reward of $\mathbf{r}$.

Definition 3 (MEC restricted MDP). *Let $\mathcal{M}$ be an MDP and $(T, A) \in \mathsf{MEC}(\mathcal{M})$ a MEC of $\mathcal{M}$. By picking some initial state $s'_{init} \in T$, we obtain the* restricted MDP $\mathcal{M}' = (T, s'_{init}, A, \mathsf{Av}', \Delta', r')$ *where*

- $\mathsf{Av}'(s) = \mathsf{Av}(s) \cap A$ *for* $s \in T$,
- $\Delta'(s, a, s') = \Delta(s, a, s')$ *for* $s, s' \in T$, $a \in A$, *and*
- $r'(s, a) = r(s, a)$ *for* $s \in T$, $a \in A$.

Classification of MDPs. If for some MDP $\mathcal{M}$, (S, Act) is a MEC, we call the MDP *strongly connected*. If it contains a single MEC plus potentially some transient states, it is called *(weakly) communicating*. Otherwise, it is called *multichain* [Put94, Sect. 8.3].

For a Markov chain, let $\Delta^n(s, s')$ denote the probability of going from the state s to state s' in n steps. The *period p of a pair s, s'* is the greatest common divisor of all n's with $\Delta^n(s, s') > 0$. The pair s, s' is called *periodic* if $p > 1$ and *aperiodic* otherwise. A Markov chain is called aperiodic if all pairs s, s' are aperiodic, otherwise the chain is called periodic. Similarly, an MDP is called aperiodic if every memoryless randomized strategy induces an aperiodic Markov chain, otherwise the MDP is called periodic.

Long-Run Average Reward. In this work, we consider the (maximum) *long-run average reward* (or *mean-payoff*) of an MDP, which intuitively describes the (maximum) average reward per step we expect to see when simulating the MDP for time going to infinity. Formally, let R_i be a random variable, which for an infinite path $\rho = s_0 a_0 s_1 a_1 \dots$ returns $R_i(\rho) = r(s_i, a_i)$, i.e. the reward observed at step $i \geq 0$. Given a strategy π, the n-step average reward then is

$$v_n^\pi(s) := \mathbb{E}_s^\pi \left(\frac{1}{n} \sum_{i=0}^{n-1} R_i \right),$$

and the *long-run average reward* of the strategy π is

$$v^\pi(s) := \liminf_{n \to \infty} v_n^\pi.$$

The lim inf is used in the definition, since the limit may not exist in general for an arbitrary strategy. Nevertheless, for finite MDPs the optimal limit-inferior (also called the *value*) is attained by some memoryless deterministic strategy $\pi^* \in \Pi^{\mathsf{MD}}$ and is in fact the limit [Put94, Theorem 8.1.2].

$$v(s) := \sup_{\pi \in \Pi} \liminf_{n \to \infty} \mathbb{E}_s^\pi \left(\frac{1}{n} \sum_{i=0}^{n-1} R_i \right) = \sup_{\pi \in \Pi} v^\pi(s) = \max_{\pi \in \Pi^{\mathsf{MD}}} v^\pi(s) = \lim_{n \to \infty} v_n^{\pi^*}.$$

An alternative well-known characterization we use in this paper is

$$v(s) = \max_{\pi \in \Pi^{\mathsf{MD}}} \sum_{M \in \mathsf{MEC}} \mathbb{P}_s^\pi[\Diamond \Box M] \cdot v(M), \tag{1}$$

where $\Diamond \Box M$ denotes the set of paths that eventually remain forever within M and $v(M)$ is the unique value achievable in the MDP restricted to the MEC M. Note that $v(M)$ does not depend on the initial state chosen for the restriction.

Algorithm 1. ValueIteration

```
Input: MDP $\mathcal{M} = (S, s_{\text{init}}, Act, \mathsf{Av}, \Delta, r)$, precision $\varepsilon > 0$
Output: $w$, s.t. $|w - v(s_{\text{init}})| < \varepsilon$
1: $t_0(\cdot) \leftarrow 0$, $n \leftarrow 0$.
2: while stopping criterion not met do
3:     $n \leftarrow n + 1$
4:     for $s \in S$ do
5:         $t_n(s) = \max_{a \in \mathsf{Av}(s)} \left(r(s,a) + \sum_{s' \in S} \Delta(s,a,s') t_{n-1}(s')\right)$
6: return $\frac{1}{n} t_n(s_{\text{init}})$
```

3 Value Iteration Solutions

3.1 Naive Value Iteration

Value iteration is a dynamic-programming technique applicable in many contexts. It is based on the idea of repetitively updating an approximation of the value for each state using the previous approximates until the outcome is precise enough. The standard value iteration for average reward [Put94, Sect. 8.5.1] is shown in Algorithm 1.

First, the algorithm sets $t_0(s) = 0$ for every $s \in S$. Then, in the inner loop, the value t_n is computed from the value of t_{n-1} by choosing the action which maximizes the expected reward plus successor values. This way, t_n in fact describes the optimal *expected n-step total reward*

$$t_n(s) = \max_{\pi \in \Pi^{\mathsf{MD}}} \mathbb{E}_s^\pi \left(\sum_{i=0}^{n-1} R_i \right) = n \cdot \max_{\pi \in \Pi^{\mathsf{MD}}} v_n^\pi(s).$$

Moreover, t_n approximates the n-multiple of the long-run average reward.

Theorem 1 [Put94, Theorem 9.4.1]**.** *For any MDP $\mathcal{M}$ and any $s \in S$ we have $\lim_{n\to\infty} \frac{1}{n} t_n(s) = v(s)$ for t_n obtained by Algorithm 1.*

Stopping Criteria. The convergence property of Theorem 1 is not enough to make the algorithm practical, since it is not known when to stop the approximation process in general. For this reason, we discuss stopping criteria which describe when it is safe to do so. More precisely, for a chosen $\varepsilon > 0$ the stopping criterion guarantees that when it is met, we can provide a value w that is ε-close to the average reward $v(s_{\text{init}})$.

We recall a stopping criterion for communicating MDPs defined and proven correct in [Put94, Sect. 9.5.3]. Note that in a communicating MDP, all states have the same average reward, which we simply denote by v. For ease of notation, we enumerate the states of the MDP $S = \{s_1, \ldots, s_n\}$ and treat the function t_n as a vector of values $\boldsymbol{t}_n = (t_n(s_1), \ldots, t_n(s_n))$. Further, we define the relative difference of the value iteration iterates as $\boldsymbol{\Delta}_n := \boldsymbol{t}_n - \boldsymbol{t}_{n-1}$ and introduce the *span semi-norm*, which is defined as the difference between the maximum and minimum element of a vector $\boldsymbol{w}$

$$\mathrm{sp}(\boldsymbol{w}) = \max_{s \in S} \boldsymbol{w}(s) - \min_{s \in S} \boldsymbol{w}(s).$$

The stopping criterion then is given by the condition

$$\mathrm{sp}(\boldsymbol{\Delta}_n) < \varepsilon. \tag{SC1}$$

When the criterion (SC1) is satisfied we have that

$$|\boldsymbol{\Delta}_n(s) - v| < \varepsilon \qquad \forall s \in S. \tag{2}$$

Moreover, we know that for communicating aperiodic MDPs the criterion (SC1) is satisfied after finitely many steps of Algorithm 1 [Put94, Theorem 8.5.2]. Furthermore, periodic MDPs can be transformed into aperiodic without affecting the average reward. The transformation works by introducing a self-loop on each state and adapting the rewards accordingly [Put94, Sect. 8.5.4]. Although this transformation may slow down VI, convergence can now be guaranteed and we can obtain ε-optimal values for any communicating MDP.

The intuition behind this stopping criterion can be explained as follows. When the computed span norm is small, $\boldsymbol{\Delta}_n$ contains nearly the same value in each component. This means that the difference between the expected $(n-1)$-step and n-step total reward is roughly the same in each state. Since in each state the n-step total reward is greedily optimized, there is no possibility of getting more than this difference per step.

Unfortunately, this stopping criterion cannot be applied on general MDPs, as it relies on the fact that all states have the same value, which is not true in general. Consider for example the MDP of Fig. 1a. There, we have that $v(s_5) = v(s_6) = 10$ but $v(s_3) = v(s_4) = 5$.

In [Put94, Sect. 9.4.2], it is conjectured that the following criterion may be applicable to general MDPs:

$$\mathrm{sp}(\boldsymbol{\Delta}_{n-1}) - \mathrm{sp}(\boldsymbol{\Delta}_n) < \varepsilon. \tag{SC2}$$

This stopping criterion requires that the difference of spans becomes small enough. While investigating the problem, we also conjectured a slight variation:

$$||\boldsymbol{\Delta}_n - \boldsymbol{\Delta}_{n-1}||_\infty < \varepsilon, \tag{SC3}$$

where $||\boldsymbol{w}||_\infty = \max_{s \in S} \boldsymbol{w}(s)$. Intuitively, both of these criteria try to extend the intuition of the communicating criterion to general MDPs, i.e. to require that in each state the reward gained per step stabilizes. Example 2 however demonstrates that neither (SC2) nor (SC3) is a valid stopping criterion.

Example 2. Consider the (aperiodic communicating) MDP in Fig. 2 with a parametrized reward value $\alpha \geq 0$. The optimal average reward is $v = \alpha$. But the first three vectors computed by value iteration are $\boldsymbol{t}_0 = (0, 0), \boldsymbol{t}_1 = (0.9 \cdot \alpha, \alpha)$,

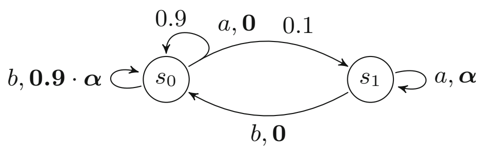


Fig. 2. A communicating MDP parametrized by the value α.

$\boldsymbol{t}_2 = (1.8 \cdot \alpha, 2 \cdot \alpha)$. Thus, the values of $\boldsymbol{\Delta}_1 = \boldsymbol{\Delta}_2 = (0.9 \cdot \alpha, \alpha)$ coincide, which means that for every choice of ε both stopping criteria (SC2) and (SC3) are satisfied by the third iteration. However, by increasing the value of α we can make the difference between the average reward $\boldsymbol{v}$ and $\boldsymbol{\Delta}_2$ arbitrary large, so no guarantee like in Eq. (2) is possible. △

3.2 Local Value Iteration

In order to remedy the lack of stopping criteria, we provide a modification of VI using MEC decomposition which is able to provide us with an ε-optimal result, utilizing the principle of Eq. (1). The idea is that for each MEC we compute an ε-optimal value, then consider these values fixed and propagate them through the MDP quotient.

Apart from providing a stopping criterion, this has another practical advantage. Observe that the naive algorithm updates all states of the model even if the approximation in a single MEC has not ε-converged. The same happens even when all MECs are already ε-converged and the values only need to propagate along the transient states. These additional updates of already ε-converged states may come at a high computational cost. Instead, our method adapts to the potentially very different speeds of convergence in each MEC.

The propagation of the MEC values can be done efficiently by transforming the whole problem to a reachability instance on a modified version of the MEC quotient, which can be solved by, for instance, VI. We call this variant the *weighted MEC quotient*. To obtain this weighted quotient, we assume that we have already computed approximate values $w(M)$ of each MEC M. We then collapse the MECs as in the MEC quotient but furthermore introduce new states s_+ and s_-, which can be reached from each collapsed state by a special action stay with probabilities corresponding to the approximate value of the MEC. Intuitively, by taking this action the strategy decides to "stay" in this MEC and obtain the average reward of the MEC.

Formally, we define the function f as the normalized approximated value, i.e. for some MEC M_i we set $f(\hat{s}_i) = \frac{1}{r_{\max}} w(M_i)$, so that it takes values in $[0, 1]$. Then, the probability of reaching s_+ upon taking the stay action in $\hat{s}_i$ is defined as $f(\hat{s}_i)$ and dually the transition to s_- is assigned $1 - f(\hat{s}_i)$ probability. If for example some MEC M had a value $v(M) = \frac{2}{3} r_{\max}$, we would have that $\Delta(\hat{s}, \mathsf{stay}, s_+) = \frac{2}{3}$. This way, we can interpret reaching s_+ as obtaining the maximal possible reward, and reaching s_- to obtaining no reward. With this intuition, we show in Theorem 2 that the problem of computing the average reward is reduced to computing the value of each MEC and determining the maximum probability of reaching the state s_+ in the weighted MEC quotient.

Definition 4 (Weighted MEC quotient). *Let* $\widehat{\mathcal{M}} = (\widehat{S}, \hat{s}_{init}, \widehat{Act}, \widehat{\mathsf{Av}}, \widehat{\Delta}, \widehat{r})$ *be the MEC quotient of an MDP* $\mathcal{M}$ *and let* $\mathsf{MEC}_{\widehat{S}} = \{\hat{s}_1, \ldots, \hat{s}_n\}$ *be the set of collapsed states. Further, let* $f : \mathsf{MEC}_{\widehat{S}} \to [0, 1]$ *be a function assigning a value to every collapsed state. We define the* weighted MEC quotient of $\mathcal{M}$ and f *as the MDP* $\mathcal{M}^f = (S^f, s^f_{init}, \widehat{Act} \cup \{\mathsf{stay}\}, \mathsf{Av}^f, \Delta^f, r^f)$*, where*

– $S^f = \widehat{S} \cup \{s_+, s_-\}$*,*
– $s^f_{init} = \hat{s}_{init}$*,*
– Av^f *is defined as*

$$\forall \hat{s} \in \widehat{S}.\ \mathsf{Av}^f(\hat{s}) = \begin{cases} \widehat{\mathsf{Av}}(\hat{s}) \cup \{\mathsf{stay}\} & if\, \hat{s} \in \mathsf{MEC}_{\widehat{S}}, \\ \widehat{\mathsf{Av}}(\hat{s}) & otherwise, \end{cases}$$

$$\mathsf{Av}^f(s_+) = \mathsf{Av}^f(s_-) = \emptyset,$$

– Δ^f *is defined as*

$$\forall \hat{s} \in \widehat{S}, \hat{a} \in \widehat{Act} \setminus \{\mathsf{stay}\}.\ \Delta^f(\hat{s}, \hat{a}) = \widehat{\Delta}(\hat{s}, \hat{a})$$

$$\forall \hat{s}_i \in \mathsf{MEC}_{\widehat{S}}.\ \Delta^f(\hat{s}_i, \mathsf{stay}) = \{s_+ \mapsto f(\hat{s}_i, s_- \mapsto 1 - f(\hat{s}_i)\},$$

– *and the reward function* $r^f(\hat{s}, \hat{a})$ *is chosen arbitrarily (e.g.* 0 *everywhere), since we only consider a reachability problem on* $\mathcal{M}^f$*.*

Example 3. Consider the MDP in Fig. 1a. The average rewards of the MECs are $v = \{\widehat{A} \mapsto 4, \widehat{B} \mapsto 5, \widehat{C} \mapsto 10\}$. With f defined as in Theorem 2, Fig. 3 shows the weighted MEC quotient $\mathcal{M}^f$. △

Theorem 2. *Given an MDP* $\mathcal{M}$ *with MECs* $\mathsf{MEC}(\mathcal{M}) = \{M_1, \ldots, M_n\}$*, define* $f(\hat{s}_i) = \frac{1}{r_{\max}} v(M_i)$ *the function mapping each MEC* M_i *to its value. Moreover, let* $\mathcal{M}^f$ *be the weighted MEC quotient of* $\mathcal{M}$ *and* f*. Then*

$$v(s_{init}) = r_{\max} \cdot \sup_{\pi \in \Pi} \mathbb{P}^{\pi}_{\mathcal{M}^f, s^f_{init}}(\Diamond s_+).$$

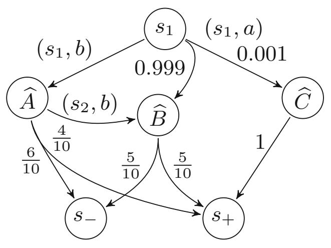


Fig. 3. The weighted quotient of the MDP in Fig. 1a and function $f = \{\widehat{A} \mapsto \frac{4}{10}, \widehat{B} \mapsto \frac{5}{10}, \widehat{C} \mapsto \frac{10}{10}\}$. Rewards and stay action labels omitted for readability.

Algorithm 2. LocalVI

```
Input: MDP $\mathcal{M} = (S, s_{\text{init}}, \mathit{Act}, \mathsf{Av}, \Delta, r)$, precision $\varepsilon > 0$
Output: $w$, s.t. $|w - v(s_{\text{init}})| < \varepsilon$
1: $f = \emptyset$
2: for $M_i = (T_i, A_i) \in \mathsf{MEC}(\mathcal{M})$ do                ▷ Determine values for MECs
3:     Compute the average reward $w(M_i)$ on $M$, such that $|w(M_i) - v(M_i)| < \frac{1}{2}\varepsilon$,
4:     $f(\hat{s}_i) \leftarrow \frac{1}{r_{\max}} w(M_i)$
5: $\mathcal{M}^f \leftarrow$ the weighted MEC quotient of $\mathcal{M}$ and $f$
6: Compute $p$ s.t. $|p - \sup_{\pi \in \Pi} \mathbb{P}^{\pi}_{\mathcal{M}^f, s^f_{\text{init}}}(\lozenge s_+)| < \frac{1}{2r_{\max}}\varepsilon$     ▷ Determine reachability
7: return $r_{\max} \cdot p$
```

The corresponding algorithm is shown in Algorithm 2. It takes an MDP and the required precision ε as input and returns a value w, which is ε-close to the average reward $v(s_{\text{init}})$. In the first part, for each MEC M the algorithm computes an approximate average reward $w(M)$ and assigns it to the function f (normalized by $r_{\max}$). Every MEC is a communicating MDP, therefore the value $w(M)$ can be computed using the naive VI with (SC1) as the stopping criterion. In the second part, the weighted MEC quotient of $\mathcal{M}$ and f is constructed and the maximum probability p of reaching s_+ in $\mathcal{M}^f$ is approximated.

Theorem 3. *For every MDP $\mathcal{M}$ and $\varepsilon > 0$, Algorithm 2 terminates and is correct, i.e. returns a value w, s.t. $|w - v(s_{init})| < \varepsilon$.*

For the correctness, we require that p is $\frac{\varepsilon}{2r_{\max}}$-close to the real maximum probability of reaching s_+. This can be achieved by using the VI algorithms for reachability from [BCC+14] or [HM14], which guarantee error bounds on the computed probability. Note that p can also be computed by other methods, such as linear programming. In Sect. 4 we empirically compare these approaches.

3.3 On-Demand Value Iteration

Observe that in Algorithm 2, the approximations for all MECs are equally precise, irrespective of the effect a MEC's value has on the overall value of the MDP. Moreover, the whole model is stored in memory and all the MECs are computed beforehand, which can be expensive for large MDPs. Often this is unnecessary, as we illustrate in the following example.

Example 4. There are three MECs $\widehat{A}, \widehat{B}, \widehat{C}$ in the MDP of Fig. 1a. Furthermore, we have that $\mathbb{P}^{\pi}_{s_{\text{init}}}(\lozenge \widehat{C}) \leq 0.001$. By using the intuition of Eq. (1), we see that no matter where in the interval $[0, r_{\max} = 20]$ its value lies, it contributes to the overall value $v(s_{\text{init}})$ at most by $0.001 \cdot r_{\max} = 0.02$. If the required precision were $\varepsilon = 0.1$, the effort invested in computing the value of $\widehat{C}$ would not pay off at all and one can completely omit constructing $\widehat{C}$.

Further, suppose that $\widehat{A}$ was a more complicated MEC, but after a few iterations the criterion (SC1) already shows that the value of $\widehat{A}$ is at most 4.4. Similarly, after several iterations in $\widehat{B}$, we might see that the value of $\widehat{B}$ is greater than 4.5. In this situation, there is no point in further approximating the value of $\widehat{A}$ since the action b leading to it will not be optimal anyway, and its precise value will not be reflected in the result. $\triangle$

To eliminate these inefficient updates, we employ the methodology of *bounded real-time dynamic programming* (BRTDP) [MLG05] adapted to the undiscounted setting in [BCC+14]. The word *bounded* refers to keeping and updating both a lower and an upper bound on the final result. It has been shown in [Put94, CI14] that bounds for the value of a MEC can be derived from the current maximum and minimum of the approximations of VI. The idea of the BRTDP approach is to perform updates not repetitively for all states in a fixed order, but more often on the *more important* states. Technically, finite runs of the system are sampled, and updates to the bounds are propagated only along the states of the current run. Since successors are sampled according to the transition probabilities, the frequently visited (and thus updated) states are those with high probability of being reached, and therefore also having more impact on the result. In order to guarantee convergence, the non-determinism is resolved by taking the *most promising action*, i.e. the one with the current highest upper bound. Intuitively, when after subsequent updates such an action turns out to be worse than hoped for, its upper bound decreases and a more promising action is chosen next time.

Since BRTDP of [BCC+14] is developed only for MDP with the reachability (and LTL) objective, we decompose our problem into a reachability and MEC analysis part. In order to avoid pre-computation of all MECs with the same precision, we instead compute the values for each MEC only when they could influence the long-run average reward starting from the initial state. Intuitively, the more a particular MEC is encountered while sampling, the more it is "reached" and the more precise information we require about its value.

To achieve this, we store upper and lower bounds on its value in the functions u and l and refine them on demand by applying VI. We modify the definition of the weighted MEC quotient to incorporate these lower and upper bounds by introducing the state $s_?$ (in addition to s_+, s_-). We call this construction the *bounded MEC quotient*. Intuitively, the probability of reaching s_+ from a collapsed state now represents the lower bound on its value, while the probability of reaching $s_?$ describes the gap between the upper and lower bound.

Definition 5 (Bounded MEC quotient). *Let $\widehat{\mathcal{M}} = (\widehat{S}, \hat{s}_{init}, \widehat{Act}, \widehat{\mathsf{Av}}, \widehat{\Delta}, \widehat{r})$ be the MEC quotient of an MDP $\mathcal{M}$ with collapsed states $\mathsf{MEC}_{\widehat{S}} = \{\hat{s}_1, \dots, \hat{s}_n\}$ and let $l, u : \{\hat{s}_1, \dots, \hat{s}_n\} \to [0,1]$ be functions that assign a lower and upper bound, respectively, to every collapsed state in $\widehat{\mathcal{M}}$. The* bounded MEC quotient $\mathcal{M}^{l,u}$ of $\mathcal{M}$ and l, u *is defined as in Definition 4 with the following changes.*

- $S^{l,u} = \widehat{S} \cup \{s_?\}$,
- $\mathsf{Av}^{l,u}(s_?) = \emptyset$,
- $\forall \hat{s} \in \mathsf{MEC}_{\widehat{S}}.\ \Delta^{l,u}(\hat{s}, \mathsf{stay}) = \{s_+ \mapsto l(\hat{s}), s_- \mapsto 1 - u(\hat{s}), s_? \mapsto u(\hat{s}) - l(\hat{s})\}$.

The unshortened definition can be found in [ACD+17, Appendix D].

Algorithm 3. OnDemandVI

```
Input: MDP $\mathcal{M} = (S, s_{\text{init}}, Act, \mathsf{Av}, \Delta, r)$, precision $\varepsilon > 0$, threshold $k \geq 2$
Output: $w$, s.t. $|w - v(s_{\text{init}})| < \varepsilon$
 1: Set $u(\cdot,\cdot) \leftarrow 1$, $u(s_-,\cdot) \leftarrow 0$; $l(\cdot,\cdot) \leftarrow 0$, $l(s_+,\cdot) \leftarrow 1$          ▷ Initialize
 2: Let $A(s) := \arg\max_{a \in \mathsf{Av}^{l,u}(s)} u(s,a)$
 3: Let $u(s) := \max_{a \in A(s)} u(s,a)$ and $l(s) := \max_{a \in A(s)} l(s,a)$
 4: repeat
 5:     $s \leftarrow s_{\text{init}}^{l,u}, w \leftarrow s$          ▷ Generate path
 6:     repeat
 7:         $a \leftarrow$ sampled uniformly from $A(s)$
 8:         $s \leftarrow$ sampled according to $\Delta^{l,u}(s,a)$
 9:         $w \leftarrow w, a, s$
10:     until $s \in \{s_+, s_-, s_?\}$ or $\mathsf{Appear}(s,w) = k$          ▷ Terminate path
11:     if $\mathrm{pop}(w) = s_?$ then          ▷ Refine MEC in which stay was taken
12:         $\mathrm{pop}(w)$
13:         $\widehat{q} \leftarrow \mathrm{top}(w)$
14:         Run VI on $\widehat{q}$, updating $u$ and $l$, until $u - l$ is halved
15:         Update $\Delta^{l,u}(\widehat{q}, \mathsf{stay})$ according to Definition 5
16:     else if $\mathsf{Appear}(s,w) = k$ then          ▷ Update EC-collapsing
17:         OnTheFlyEc
18:     repeat          ▷ Back-propagate values
19:         $a \leftarrow \mathrm{pop}(w)$, $s \leftarrow \mathrm{pop}(w)$
20:         $u(s,a) \leftarrow \sum_{s' \in S} \Delta(s,a,s') \cdot u(s')$
21:         $l(s,a) \leftarrow \sum_{s' \in S} \Delta(s,a,s') \cdot l(s')$
22:     until $w = \emptyset$
23: until $u(s_{\text{init}}) - l(s_{\text{init}}) < \frac{2\varepsilon}{r_{\max}}$          ▷ Terminate
24: return $r_{\max} \cdot \frac{1}{2}(u(s_{\text{init}}) + l(s_{\text{init}}))$
```

The probability of reaching s_+ and the probability of reaching $\{s_+, s_?\}$ give the lower and upper bound on the value $v(s_{\text{init}})$, respectively.

Corollary 1. *Let $\mathcal{M}$ be an MDP and l, u functions mapping each MEC M_i of $\mathcal{M}$ to (normalized) lower and upper bounds on the value, respectively, i.e. $l(\hat{s}_i) \leq \frac{1}{r_{\max}} v(M_i) \leq u(\hat{s}_i)$. Then*

$$r_{\max} \cdot \sup_{\pi \in \Pi} \mathbb{P}^{\pi}_{\mathcal{M}^{l,u}, s^{l,u}_{init}}(\Diamond s_+) \leq v(s_{init}) \leq r_{\max} \cdot \sup_{\pi \in \Pi} \mathbb{P}^{\pi}_{\mathcal{M}^{l,u}, s^{l,u}_{init}}(\Diamond \{s_+, s_?\}),$$

where $\mathcal{M}^{l,u}$ is the bounded MEC quotient of $\mathcal{M}$ and l, u.

Algorithm 3 shows the on-demand VI. The implementation maintains a partial model of the MDP and $\mathcal{M}^{l,u}$, which contains only the states explored by the runs. It interleaves two concepts: (i) naive VI is used to provide upper and lower bounds on the value of discovered end components, (ii) the method of [BCC+14] is used to compute the reachability on the collapsed MDP.

In lines 6–10 a random run is sampled following the “most promising” actions, i.e. the ones with maximal upper bound. The run terminates once it reaches s_+, s_- or $s_?$, which only happens if stay was one of the most promising actions.

Procedure 4. OnTheFlyEc

```
1: for $(T_i, A_i) \in \mathsf{MEC}(\mathcal{M}^{l,u})$ do
2:     Collapse $(T_i, A_i)$ to $\hat{s}_i$ in $\mathcal{M}^{l,u}$
3:     for $s \in T_i, a \in \mathsf{Av}(s) \setminus A_i$ do
4:         $u(\hat{s}_i, (s,a)) \leftarrow u(s,a)$
5:         $l(\hat{s}_i, (s,a)) \leftarrow l(s,a)$
6:     Add the stay action according to Definition 5.
```

A likely arrival to $s_?$ reflects a high difference between the upper and lower bound and, if the run ends up in $s_?$, this indicates that the upper and lower bounds of the MEC probably have to be refined. Therefore, in lines 11–15 the algorithm resumes VI on the corresponding MEC to get a more precise result. This decreases the gap between the upper and lower bound for the corresponding collapsed state, thus decreasing the probability of reaching $s_?$ again.

The algorithm uses the function $\mathsf{Appear}(s, w) = |\{i \in \mathbb{N} \mid s = w[i]\}|$ to count the number of occurrences of the state s on the path w. Whenever we encounter the same state k times (where k is given as a parameter), this indicates that the run may have got stuck in an end component. In such a case, the algorithm calls OnTheFlyEc [BCC+14], presented in Procedure 4, to detect and collapse end components of the partial model. By calling OnTheFlyEc we compute the bounded quotient of the MDP on the fly. Without collapsing the end components, our reachability method could remain forever in an end component, and thus never reach s_+, s_- or $s_?$. Finally, in lines 18–22 we back-propagate the upper and lower bounds along the states of the simulation run.

Theorem 4. *For every MDP $\mathcal{M}$, $\varepsilon > 0$ and $k \geq 2$, Algorithm 3 terminates almost surely and is correct, i.e. returns a value w, s.t. $|w - v(s_{init})| < \varepsilon$.*

4 Implementation and Experimental Results

In this section, we compare the runtime of our presented approaches to established tools. All benchmarks have been run on a 4.4.3-gentoo x64 virtual machine with 3.0 GHz per core, a time limit of one hour and memory limit of 8GB. The precision requirement for all approximative methods is $\varepsilon = 10^{-6}$. We implemented our constructions as a package in the PRISM Model Checker [KNP11]. We used the 64-bit Oracle JDK version `1.8.0_102-b14` as Java runtime for all executions. All measurements are given in seconds, measuring the total user CPU time of the PRISM process using the UNIX tool `time`.

4.1 Models

First, we briefly explain the examples used for evaluation. **virus** [KNPV09] models a virus spreading through a network. We reward each attack carried out by an infected machine. Note that in this model, no machine can "purge" the virus, hence eventually all machines will be infected. **cs_nfail** [KPC12] models a client-server mutual exclusion protocol with probabilistic failures of the clients. A reward is given for each successfully handled connection. **investor** [MM07,MM02] models an investor operating in a stock market. The investor can decide to sell his stocks and keep their value as a reward or hold them and wait to see how the market evolves. The rewards correspond to the value of the stocks when the investor decides to sell them, so maximizing the average reward corresponds to maximizing the expected selling value of the stocks. **phil_nofair** [DFP04] represents the (randomised) dining philosophers without fairness assumptions. We use two reward structures, one where a reward is granted each time a philosopher "thinks" or "eats", respectively. **rabin** [Rab82] is a well-known mutual exclusion protocol, where multiple processes repeatedly try to access a shared critical section. Each time a process successfully enters the critical section, a reward is given. **zeroconf** [KNPS06] is a network protocol designed to assign IP addresses to clients without the need of a central server while still avoiding address conflicts. We explain the reward assignment in the corresponding result section. **sensor** [KPC12] models a network of sensors sending values to a central processor over a lossy connection. A reward is granted for every *work* transition.

4.2 Tools

We will compare several different variants of our implementations, which are described in the following.

– Naive value iteration (`NVI`) runs the value iteration on the whole MDP as in Algorithm 1 of Sect. 3.1 together with the stopping criterion (SC2) conjectured by [Put94, Sect. 9.4.2]. As the stopping criterion is incorrect, we will not only include the runtime until the stopping criterion is fulfilled, but also until the computed value is ε-close to the known solution.
– Our MEC decomposition approach presented in Algorithm 2 of Sect. 3.2 is denoted by `MEC-`*`reach`*, where *`reach`* identifies one of the following reachability solver used on the quotient MDP.
  - PRISM's value iteration (`VI`), which iterates until none of the values change by more than 10^{-8}. While this method is theoretically imprecise, we did not observe this behaviour in our examples.[2]
  - An exact reachability solver based on linear programming (`LP`) [Gir14].
  - The BRTDP solver with guaranteed precision of [BCC+14] (`BRTDP`). This solver is highly configurable. Among others, one can specify the heuristic which is used to resolve probabilistic transitions in the simulation. This can happen according to transition probability (`PR`), round-robin (`RR`) or maximal difference (`MD`). Due to space constraints, we only compare to the `MD` exploration heuristic here. Results on the other heuristics can be found in [ACD+17, Appendix E]

[2] PRISM contains several other methods to solve reachability, which all are imprecise and behaved comparably in our tests.

– `ODV` is the implementation of the on-demand value iteration as in Algorithm 3 of Sect. 3.3. Analogously to the above, we only provide results on the `MD` heuristic here. The results on `ODV` together with the other heuristics can also be found in [ACD+17, Appendix E].

Furthermore, we will compare our methods to the state-of-the-art tool MultiGain, version 1.0.2 [BCFK15] abbreviated by `MG`. MultiGain uses linear programming to exactly solve mean payoff objectives among others. We use the commercial LP solver Gurobi 7.0.1 as backend[3]. We also instantiated ***reach*** by an implementation of the interval iteration algorithm presented in [HM14]. This variant performed comparable to `MEC-VI` and therefore we omitted it.

Table 1. Runtime comparison of our approaches to MultiGain on various, reasonably sized models. Timeouts (1h) are denoted by TO. Strongly connected models are denoted by "scon" in the MEC column. The best result in each row is marked in bold, excluding NVI due to its imprecisions. For NVI, we list both the time until the stopping criterion is satisfied and until the values actually converged.

Model	States	MECs	`MG`	`NVI`	`MEC-VI`	`MEC-LP`	`MEC-BRTDP`	`ODV`
virus	809	1	**3.76**	3.50/3.71	4.09	4.41	4.40	TO
cs_nfail4	960	176	4.86	10.2/TO	**4.38**	TO	9.39	16.0
investor	6688	837	16.75	4.23/TO	**8.83**	TO	64.5	18.7
phil-nofair5	93068	scon	TO	23.5/30.3	**70**	**70**	**70**	TO
rabin4	668836	scon	TO	87.8/164	**820**	**820**	**820**	TO

4.3 Results

The experiments outlined in Table 1 show that our methods outperform MultiGain significantly on most of the tested models. Furthermore, we want to highlight the **investor** model to demonstrate the advantage of `MEC-VI` over `MEC-LP`. With higher number of MECs in the initial MDP, which is linked to the size of the reachability LP, the runtime of `MEC-LP` tends to increase drastically, while `MEC-VI` performs quite well. Additionally, we see that `NVI` fails to obtain correct results on any of these examples.

`ODV` does not perform too well in these tests, which is primarily due to the significant overhead incurred by building the partial model dynamically. This is especially noticeable for strongly connected models like **phil-nofair** and **rabin**. For these models, every state has to be explored and `ODV` does a lot of superfluous computations until the model has been explored fully. On **virus**, the bad performance is due to the special topology of the model, which obstructs the back-propagation of values.

[3] MultiGain also supports usage of the LP solver `lp_solve 5.5` bundled with PRISM, which consistently performed worse than the Gurobi backend.

Moreover, on the two strongly connected models all MEC decomposition based methods perform worse than naive value iteration as they have to obtain the MEC decomposition first. Furthermore, all three of those methods need the same amount of for these models, as the weighted MEC quotient only has a single state (and the two special states), thus the reachability query is trivial.

In Table 2 we present results of some of our methods on **zeroconf** and **sensors**, which both have a structure better suited towards `ODV`. The **zeroconf** model consists of a big transient part and a lot of "final" states, i.e. states which only have a single self-loop. **sensors** contains a lot of small, often unlikely-to-be-reached MECs.

Table 2. Runtime comparison of our on-demand VI method with the previous approaches. All of those behaved comparable to `MEC-VI` or worse, and due to space constraints we omit them. MO denotes a memory-out. Aside from runtime, we furthermore list the number of explored states and MECs of `ODV`

Model	States	`MEC-VI`	`ODV`	`ODV` States	`ODV` MECs
zeroconf(40,10)	3001911	MO	5.05	481	3
avoid				582	3
zeroconf(300,15)	4730203	MO	16.6	873	3
avoid				5434	3
sensors(2)	7860	18.9	20.1	3281	917
sensors(3)	77766	2293	37.2	10941	2301

On the **zeroconf** model, we evaluate the average reward problem with two reward structures. In the default case, we assign a reward of 1 to every final state and zero elsewhere. This effectively is solving the reachability question and thus it is not surprising that our method gives similarly good results as the BRTDP solver of [BCC+14]. The *avoid* evaluation has the reward values flipped, i.e. all states except the final ones yield a payoff of 1. With this reward assignment, the algorithm performed slightly slower, but still extremely fast given the size of the model. We also tried assigning pseudo-random rewards to every non-final state, which did not influence the speed of convergence noticeably. We want to highlight that the mem-out of `MEC-VI` already occurred during the MEC-decomposition phase. Hence, no variant of our decomposition approach can solve this problem.

Interestingly, the naive value iteration actually converges on **zeroconf**(40,10) in roughly 20 min. Unfortunately, as in the previous experiments, the used incorrect stopping criterion was met a long time before that.

Further, when comparing **sensors**(2) to **sensors**(3), the runtime of `ODV` only doubled, while the number of states in the model increased by an order of magnitude and the runtime of `MEC-VI` even increased by two orders of magnitude.

These results show that for some models, `ODV` is able to obtain an ε-optimal estimate of the mean payoff while only exploring a tiny fraction of the state space. This allows us to solve many problems which previously were intractable simply due to an enormous state space.

5 Conclusion

We have discussed the use of value iteration for computing long-run average rewards in general MDPs. We have shown that the conjectured stopping criterion from literature is not valid, designed two modified versions of the algorithm and have shown guarantees on their results. The first one relies on decomposition into VI for long-run average on separate MECs and VI for reachability on the resulting quotient, achieving global error bounds from the two local stopping criteria. The second one additionally is simulation-guided in the BRTDP style, and is an anytime algorithm with a stopping criterion. The benchmarks show that depending on the topology, one or the other may be more efficient, and both outperform the existing linear programming on all larger models. For future work, we pose the question of how to automatically fine-tune the parameters of the algorithms to get the best performance. For instance, the precision increase in each further call of VI on a MEC could be driven by the current values of VI on the quotient, instead of just halving them. This may reduce the number of unnecessary updates while still achieving an increase in precision useful for the global result.

References

[ACD+17] Ashok, P., Chatterjee, K., Daca, P., Křetínský, J., Meggendorfer, T.: Value iteration for long-run average reward in Markov decision processes. Technical report arXiv:1705.02326, arXiv.org (2017)

[BBC+14] Brázdil, T., Brožek, V., Chatterjee, K., Forejt, V., Kučera, A.: Markov decision processes with multiple long-run average objectives. LMCS **10**(1), 1–29 (2014). doi:10.2168/LMCS-10(1:13)2014

[BCC+14] Brázdil, T., Chatterjee, K., Chmelík, M., Forejt, V., Křetínský, J., Kwiatkowska, M., Parker, D., Ujma, M.: 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). doi:10.1007/978-3-319-11936-6_8

[BCFK13] Brázdil, T., Chatterjee, K., Forejt, V., Kucera, A.: Trading performance for stability in Markov decision processes. In: LICS, pp. 331–340 (2013)

[BCFK15] Brázdil, T., Chatterjee, K., Forejt, V., Kučera, A.: MultiGain: a controller synthesis tool for MDPs with multiple mean-payoff objectives. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 181–187. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46681-0_12

[BK08] Baier, C., Katoen, J.-P.: Principles of Model Checking. MIT Press, Cambridge (2008)

[CH11] Chatterjee, K., Henzinger, M.: Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification. In: SODA, pp. 1318–1336. SIAM (2011)

[CH12] Chatterjee, K., Henzinger, M.: An O(n^2) time algorithm for alternating büchi games. In: SODA, pp. 1386–1399. SIAM (2012)

[CH14] Chatterjee, K., Henzinger, M.: Efficient and dynamic algorithms for alternating büchi games and maximal end-component decomposition. J. ACM **61**(3), 15:1–15:40 (2014)

[Cha07] Chatterjee, K.: Markov decision processes with multiple long-run average objectives. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 473–484. Springer, Heidelberg (2007). doi:10.1007/978-3-540-77050-3_39

[CI14] Chatterjee, K., Ibsen-Jensen, R.: The complexity of ergodic mean-payoff games. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8573, pp. 122–133. Springer, Heidelberg (2014). doi:10.1007/978-3-662-43951-7_11

[CKK15] Chatterjee, K., Komárková, Z., Křetínský, J.: Unifying two views on multiple mean-payoff objectives in Markov decision processes. In: LICS, pp. 244–256 (2015)

[CL13] Chatterjee, K., Łącki, J.: Faster algorithms for Markov decision processes with low treewidth. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 543–558. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39799-8_36

[CY95] Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J. ACM **42**(4), 857–907 (1995)

[dA97] de Alfaro, L.: Formal verification of probabilistic systems. Ph.D. thesis, Stanford University (1997)

[DFP04] Duflot, M., Fribourg, L., Picaronny, C.: Randomized dining philosophers without fairness assumption. Distrib. Comput. **17**(1), 65–76 (2004)

[FV97] Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer, New York (1997). doi:10.1007/978-1-4612-4054-9

[Gir14] Giro, S.: Optimal schedulers vs optimal bases: an approach for efficient exact solving of Markov decision processes. Theor. Comput. Sci. **538**, 70–83 (2014)

[HM14] Haddad, S., Monmege, B.: Reachability in MDPs: refining convergence of value iteration. In: Ouaknine, J., Potapov, I., Worrell, J. (eds.) RP 2014. LNCS, vol. 8762, pp. 125–137. Springer, Cham (2014). doi:10.1007/978-3-319-11439-2_10

[How60] Howard, R.A.: Dynamic Programming and Markov Processes. MIT Press, New York, London, Cambridge (1960)

[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). doi:10.1007/978-3-642-22110-1_47

[KNPS06] Kwiatkowska, M., Norman, G., Parker, D., Sproston, J.: Performance analysis of probabilistic timed automata using digital clocks. Formal Methods Syst. Des. **29**, 33–78 (2006)

[KNPV09] Kwiatkowska, M., Norman, G., Parker, D., Vigliotti, M.G.: Probabilistic mobile ambients. Theoret. Comput. Sci. **410**(12–13), 1272–1303 (2009)

[KPC12] Komuravelli, A., Păsăreanu, C.S., Clarke, E.M.: Assume-guarantee abstraction refinement for probabilistic systems. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 310–326. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31424-7_25

[MLG05] McMahan, H.B., Likhachev, M., Gordon, G.J.: Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In: ICML, pp. 569–576 (2005)

[MM02] McIver, A.K., Morgan, C.C.: Games, probability, and the quantitative μ-calculus $qM\mu$. In: Baaz, M., Voronkov, A. (eds.) LPAR 2002. LNCS, vol. 2514, pp. 292–310. Springer, Heidelberg (2002). doi:10.1007/3-540-36078-6_20

[MM07] McIver, A., Morgan, C.: Results on the quantitative μ-calculus qMu. ACM Trans. Comput. Logic **8**(1), 3 (2007)

[PGT03] Pineau, J., Gordon, G.J., Thrun, S.: Point-based value iteration: an anytime algorithm for POMDPs. In: IJCAI, pp. 1025–1032 (2003)

[Put94] Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, Hoboken (1994)

[Rab82] Michael, O.: N-Process mutual exclusion with bounded waiting by 4 Log2N-valued shared variable. J. Comput. Syst. Sci. **25**(1), 66–75 (1982)

[Seg95] Segala, R.: Modelling and verification of randomized distributed real time systems. Ph.D. thesis, Massachusetts Institute of Technology (1995)

[Vei66] Veinott, A.F.: On finding optimal policies in discrete dynamic programming with no discounting. Ann. Math. Statist. **37**(5), 1284–1294 (1966)

[WBB+10] Wimmer, R., Braitling, B., Becker, B., Hahn, E.M., Crouzen, P., Hermanns, H., Dhama, A., Theel, O.E.: Symblicit calculation of long-run averages for concurrent probabilistic systems. In: QEST, pp. 27–36 (2010)

B Efficient Strategy Iteration for Mean Payoff in Markov Decision Processes. ATVA 2017

This section has been published as **peer-reviewed conference paper**.

> Jan Křetínský and Tobias Meggendorfer. 'Efficient Strategy Iteration for Mean Payoff in Markov Decision Processes'. In: *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings*. Ed. by Deepak D'Souza and K. Narayan Kumar. Vol. 10482. Lecture Notes in Computer Science. Springer, 2017, pp. 380–399. DOI: `10.1007/978-3-319-68167-2\_25`. URL: `https://doi.org/10.1007/978-3-319-68167-2%5C_25`

Synopsis We present several practical optimizations of the *strategy iteration* (SI) approach to optimize *mean payoff* objectives in Markov decision processes. Typically, there are three main approaches towards solving mean payoff. Firstly, linear programming is theoretically appealing since it yields precise solutions in polynomial time. However, in practice, LP barely scales beyond systems with a few thousand states. Usually, dynamic programming programming approaches are considered. In [Ash+17] (see Paper A), we present a scalable *value iteration* (VI) approach for mean payoff. While VI scales extremely well, it only offers precise solutions in the limit and thus is only usable for approximative solutions. In contrast, SI offers precise solutions, practical scalability, and conceptual simplicity. By applying topological arguments and approximation heuristics, combining VI and SI, we obtain a fast yet precise solver for mean payoff, with performance comparable to state-of-the-art tools.

Contributions of the thesis author Composition, discussion and revision of the entire manuscript. Sole contribution of all results and proofs presented in the paper. Sole design and implementation of the presented tool. Creation of the software artefact for conference submission.

Efficient Strategy Iteration for Mean Payoff in Markov Decision Processes

Jan Křetínský and Tobias Meggendorfer(✉)

Technical University of Munich, Munich, Germany
tobias.meggendorfer@in.tum.de

Abstract. Markov decision processes (MDPs) are standard models for probabilistic systems with non-deterministic behaviours. Mean payoff (or long-run average reward) provides a mathematically elegant formalism to express performance related properties. Strategy iteration is one of the solution techniques applicable in this context. While in many other contexts it is the technique of choice due to advantages over e.g. value iteration, such as precision or possibility of domain-knowledge-aware initialization, it is rarely used for MDPs, since there it scales worse than value iteration. We provide several techniques that speed up strategy iteration by orders of magnitude for many MDPs, eliminating the performance disadvantage while preserving all its advantages.

1 Introduction

Markov decision processes (MDPs) [19,28,34] are a standard model for analysis of systems featuring both probabilistic and non-deterministic behaviour. They have found rich applications, ranging from communication protocols to biological systems and robotics. A classical objective to be optimized in MDPs is *mean payoff* (or *long-run average reward*). It captures the reward we can achieve on average per step when simulating the MDP. Technically, one considers partial averages (average over the first n steps) and let the time n go to infinity. This objective can be used to describe performance properties of systems, for example, average throughput, frequency of errors, average energy consumption, etc.

Strategy (or *policy*) *iteration* (or *improvement*) (SI) is a dynamic-programming technique applicable in many settings, including optimization of mean payoff in MDPs [28,34], but also mean payoff games [6,9], parity games [17,33,35,38], simple stochastic games [11], concurrent reachability games [25], or stochastic parity games [24]. The main principle of the technique is to start with an arbitrary strategy (or policy or controller of the system) and iteratively improve it locally in a greedy fashion until no more improvements can be done. The resulting strategy is guaranteed to be optimal.

SI has several advantages compared to other techniques used in these contexts. Most interestingly, domain knowledge or heuristics can be used to *initialize*

This work is partially supported by the German Research Foundation (DFG) project "Verified Model Checkers" and the Czech Science Foundation grant No. 15-17564S.

DOI: 10.1007/978-3-319-68167-2_25

with a reasonable strategy, thus speeding up the computation to a fraction of the usual analysis time. Further, SI is conceptually simple as it boils down to a search through a *finite space* of memoryless deterministic strategies, yielding arguments for correctness and termination of the algorithm.

More specifically, in the context of MDPs, it has advantages over the other two standard techniques. Firstly, compared to *linear programming* (LP), SI *scales* much better. LP provides a rich framework, which is able to encode many optimization problems on MDPs and in particular mean payoff. However, although the linear programs are typically of polynomial size and can be also solved in polynomial time, such procedures are not very useful in practice. For larger systems the solvers often time out or run out of memory already during the construction of the linear program. Furthermore, SI ensures that the current lower bounds on the mean payoff is *monotonically improving*. Consequently, the iteration can be stopped at any point, yielding a strategy at least as good as all previous iterations.

Secondly, compared to *value iteration* (VI), SI provides a *precise solution*, whereas VI is only optimal in the limit and the number of iterations before the numbers can be rounded in order to obtain a precise solution is very high [10]. Furthermore, stopping criteria for VI are limited to special cases or are very inefficient. Consequently, VI is practically used to produce results that may be erroneous even for simple, realistic examples in verification, see e.g. [23].

On the other hand, the main disadvantage of SI, in particular for mean payoff, is its *scalability*. Although SI scales better than LP, it is only rarely the case that SI is faster than VI. Firstly, in the worst case, we have to examine *exponentially* many strategies [15], in contrast to the discounted case, which is polynomial (for a fixed discount factor) [39] even for games [26]. However, note that even for parity games it was for long not known [21] whether all SI algorithms exhibit this property since the number of improvements is only rarely high in practice. Secondly, and more importantly, the *evaluation* of each strategy necessary for the greedy improvement takes enormous time since large systems of linear equations have to be solved. Consequently, VI typically is much faster than SI to obtain a similar precision, although it may also need an exponential number of updates.

This scalability limitation is even more pronounced by the following contrast. On the one hand, mean payoff games, parity games, and simple stochastic games are not known to be solvable in polynomial time, hence the exponential-time SI is an acceptable technique for these models. On the other hand, for problems on MDPs that are solvable in polynomial time, such as mean payoff, the exponential-time SI becomes less appealing. In summary, we can only afford to utilize the mentioned advantages of SI for MDPs if we make SI perform well in practice.

This paper suggest several heuristics and opens new directions to increase performance of SI for MDPs, in particular in the setting of mean payoff. Our contribution is the following:

– We present several techniques to significantly speed up SI in many cases, most importantly the evaluation of the current strategy. The first set of techniques (in Sect. 4) is based on maximal end component decomposition of the

MDP and strongly connected component decomposition of the Markov chain induced by the MDP and the currently considered strategy. The second class (in Sect. 5) is based on approximative techniques to compute mean payoff in these Markov chains. Both variants reduce the time taken by the strategy evaluation. Finally, we combine the two approaches in a non-trivial way in Sect. 5.1, giving rise to synergic optimizations and opening the door for approximation techniques.
– We provide experimental evaluation of the proposed techniques and compare to the approaches from literature. We show experimental evidence that our techniques are speeding up SI by orders of magnitude and make its performance (i) on par with VI, the prevalent technique which, however, only provides approximate solutions, and (ii) incomparably more scalable than the precise technique of LP.

Further related work. Strategy iteration for MDPs has been extensively studied [16,28,34]. Performance of SI for MDPs has been mainly improved in the discounted total reward case by, e.g., approximate evaluation of the strategy using iterative methods of linear algebra [36], model reduction by adaptive state-space aggregation [1] or close-to-optimal initialization [20]; for an overview see [5]. The treatment of the undiscounted case has focused on unichain MDPs [27,34]. Apart from solving the MDPs modelling probabilistic systems, the technique has found its applications in other domains, too, for example program analysis [22].

2 Preliminaries

In this section, we introduce some central notions. Furthermore, relevant technical notions from linear algebra can be found in [30, Appendix A].

A *probability distribution* on a finite set X is a mapping $\rho : X \to [0,1]$, such that $\sum_{x \in X} \rho(x) = 1$. Its *support* is denoted by $supp(\rho) = \{x \in X \mid \rho(x) > 0\}$. $\mathcal{D}(X)$ denotes the set of all probability distributions on X.

Definition 1. *A* Markov chain (MC) *is a tuple* $\mathsf{M} = (S, s_{init}, \Delta, r)$*, where* S *is a finite set of* states*,* $s_{init} \in S$ *is the* initial *state,* $\Delta : S \to \mathcal{D}(S)$ *is a* transition function *that for each state* s *yields a probability distribution over successor states and* $r : S \to \mathbb{R}^{\geq 0}$ *is a* reward function*, assigning rewards to states.*

Definition 2. *A* Markov decision process (MDP) *is a tuple of the form* $\mathcal{M} = (S, s_{init}, Act, \mathsf{Av}, \Delta, r)$*, where* S *is a finite set of* states*,* $s_{init} \in S$ *is the* initial *state,* Act *is a finite set of* actions*,* $\mathsf{Av} : S \to 2^{Act}$ *assigns to every state a set of* available *actions,* $\Delta : S \times Act \to \mathcal{D}(S)$ *is a* transition function *that for each state* s *and action* $a \in \mathsf{Av}(s)$ *yields a probability distribution over successor states and* $r : S \times Act \to \mathbb{R}^{\geq 0}$ *is a* reward function*, assigning rewards to state-action pairs.*

Furthermore, 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]

[1] The usual procedure of achieving this in general is to replace Act by $S \times Act$ and adapting Av, Δ, and r appropriately.

For ease of notation, we overload functions mapping 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 $\varDelta(s)(s')$ and $\varDelta(s,a)(s')$ we write $\varDelta(s,s')$ and $\varDelta(s,a,s')$, respectively. Further, given some MC M, a function $f : S \to \mathbb{R}$ and set of states $C \subseteq S$, we define $\mathbb{E}^C_\varDelta(f,s) := \sum_{s' \in C} \varDelta(s,s')f(s')$, i.e. the weighted sum of f over all the successors of s in C. Analogously, for some MDP $\mathcal{M}$, we set $\mathbb{E}^C_\varDelta(f,s,a) := \sum_{s' \in C} \varDelta(s,a,s')f(s')$. Further, we define $\mathbb{E}_\varDelta(f,s) := \mathbb{E}^S_\varDelta(f,s)$ and $\mathbb{E}_\varDelta(f,s,a) := \mathbb{E}^S_\varDelta(f,s,a)$.

An *infinite path* ρ in a Markov chain is an infinite sequence $\rho = s_0 s_1 \dots \in S^\omega$, such that for every $i \in \mathbb{N}$ we have that $\varDelta(s_i, s_{i+1}) > 0$. A *finite path* $w = s_0 s_1 \dots s_n \in S^*$ is a finite prefix of an infinite path. Similarly, an *infinite path* in an MDP is some infinite sequence $\rho = s_0 a_0 s_1 a_1 \dots \in (S \times Act)^\omega$, such that for every $i \in \mathbb{N}$, $a_i \in \mathsf{Av}(s_i)$ and $\varDelta(s_i, a_i, s_{i+1}) > 0$. *Finite paths* are defined analogously as elements of $(S \times Act)^* \times S$.

A Markov chain together with a state s induces a unique probability distribution $\mathbb{P}_s$ over measurable sets of infinite paths [3, Chapter 10]. For some $C \subseteq S$, we write $\Diamond C$ to denote the set of all paths which eventually reach C, i.e. $\Diamond C = \{\rho = s_0 s_1 \dots \mid \exists i \in \mathbb{N}.\ s_i \in C\}$, which is measurable [3, Sect. 10.1.1].

A *strategy* on an MDP is a function $\pi : (S \times Act)^* \times S \to \mathcal{D}(Act)$, which given a finite path $w = s_0 a_0 s_1 a_1 \dots s_n$ yields a probability distribution $\pi(w) \in \mathcal{D}(\mathsf{Av}(s_n))$ on the actions to be taken next. We call a strategy *memoryless randomized* (or *stationary*) if it is of the form $\pi : S \to \mathcal{D}(Act)$, and *memoryless deterministic* (or *positional*) if it is of the form $\pi : S \to Act$. We denote the set of all strategies of an MDP by $\varPi$, and the set of all memoryless deterministic strategies as $\varPi^{\mathsf{MD}}$. Note that $\varPi^{\mathsf{MD}}$ is finite, since at each state there exist only finitely many actions to choose from. Fixing any positional strategy π induces a Markov chain where $\varDelta(s,s') = \sum_{s \in \mathsf{Av}(s)} \pi(s,a) \cdot \varDelta(s,a,s')$ and $r(s) = \sum_{a \in \mathsf{Av}(s)} \pi(s,a) \cdot r(s,a)$.

Fixing a strategy π and an initial state s on an MDP $\mathcal{M}$ also gives a unique measure $\mathbb{P}^\pi_s$ over infinite paths [34, Sect. 2.1.6]. The expected value of a random variable F then is defined as $\mathbb{E}^\pi_s[F] = \int F \; d\mathbb{P}^\pi_s$.

Strongly Connected Components and End Components. A non-empty set of states $C \subseteq S$ in a Markov chain is *strongly connected* if for every pair $s, s' \in C$ there is a path from s to s', possibly of length zero. Such a set C is a *strongly connected component* (SCC) if it is inclusion maximal, i.e. there exists no strongly connected C' with $C \subsetneq C'$. Note that each state of an MC belongs to exactly one SCC[2]. An SCC is called *bottom strongly connected component* (BSCC) if additionally no path leads out of it, i.e. for $s \in C, s' \in S \setminus C$ we have $\varDelta(s,s') = 0$. The set of (B)SCCs in an MC M is denoted by $\mathsf{SCC}(\mathsf{M})$ and $\mathsf{BSCC}(\mathsf{M})$, respectively.

The concept of SCCs is generalized to MDPs by so called *(maximal) end components*. 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*

[2] Some authors deliberately exclude so called "trivial" or "transient" SCCs, which are single states without a self-loop.

component of an MDP $\mathcal{M}$ if (i) for all $s \in T, a \in A \cap \mathsf{Av}(s)$ we have $supp(\Delta(s,a)) \subseteq T$, and (ii) for all $s, s' \in T$ there is a finite path $w = sa_0 \ldots a_n s' \in (T \times A)^* \times T$, i.e. the path stays inside T and only uses actions in A. Note that we assumed actions to be unique for each state.

Intuitively, an end component describes a set of states for which a particular strategy exists such that all possible paths remain inside these states. 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 $A \subseteq A'$. Given an MDP $\mathcal{M}$, the set of its MECs is denoted by $\mathsf{MEC}(\mathcal{M})$.

Finally, given an MDP $\mathcal{M}$ let $(T, A) \in \mathsf{MEC}(\mathcal{M})$ some MEC in it. By picking some initial state $s'_{\text{init}} \in T$, defining the straightforward restrictions of Av and Δ by $\mathsf{Av}' : T \to 2^A$, $\mathsf{Av}'(s) := \mathsf{Av}(s) \cap A$ and $\Delta' : T \times A \to \mathcal{D}(T)$, $\Delta'(s,a) := \Delta(s,a)$ one obtains the *restricted MDP* $\mathcal{M}' = (T, s'_{\text{init}}, A, \mathsf{Av}', \Delta')$.

Remark 1. For a Markov chain M, the computation of $\mathsf{SCC}(\mathsf{M})$, $\mathsf{BSCC}(\mathsf{M})$ and a topological ordering of the SCCs can be achieved in linear time w.r.t. the number of states and transitions by, e.g., Tarjan's algorithm [37]. Similarly, the MEC decomposition of an MDP can be computed in polynomial time [12].

Long-Run Average Reward. (also called *mean payoff*) of a strategy π intuitively describes the optimal reward we can expect on average per step when simulating the MDP according to π. In the following, we will only consider the case of maximizing the average reward, but the presented methods easily can be transferred to the minimization case.

Formally, let R_i be a random variable which given an infinite path returns the reward obtained at step $i \geq 0$, i.e. for $\rho = s_0 a_0 s_1 a_1 \ldots$ we have $R_i(\rho) = r(s_i, a_i)$. Given a strategy π, the n-step (maximal) average reward then is defined as $g_n^\pi(s) = \mathbb{E}_s^\pi(\frac{1}{n}\sum_{i=0}^{n-1} R_i)$. The *long-run average reward* (in this context also traditionally called *gain* [34]) of the strategy π is $g^\pi(s) = \liminf_{n\to\infty} g_n^\pi(s)$.[3] Consequently, the *long-run average reward* (or *gain*) of a state s is defined as

$$g^*(s) := \sup_{\pi \in \Pi} g^\pi(s) = \sup_{\pi \in \Pi} \liminf_{n\to\infty} \mathbb{E}_s^\pi \left(\frac{1}{n} \sum_{i=0}^{n-1} R_i \right).$$

For finite MDPs $g^*(s)$ in fact is attained by a memoryless deterministic strategy $\pi^* \in \Pi^{\mathsf{MD}}$ and it further is the *limit* of the n-step average reward [34]. Formally,

$$g^*(s) = \max_{\pi \in \Pi^{\mathsf{MD}}} g^\pi(s) = \lim_{n\to\infty} g_n^{\pi^*}(s).$$

With this in mind, we now only consider memoryless deterministic strategies.

3 Strategy Iteration

One way of computing the optimal gain of an MDP (i.e. determining the optimal gain of each state) is *strategy iteration* (or *policy iteration* or *strategy*

[3] The $\liminf$ is used since the limit may not exist in general for an arbitrary strategy.

improvement). The general approach of strategy iteration is to (i) fix a strategy, (ii) evaluate it and (iii) improve each choice greedily, repeating the process until no improvement is possible any more. For an in depth theoretical exposé of strategy iteration for MDPs, we refer to e.g. [34, Sect. 9.2]. We briefly recall the necessary definitions.

Gain and Bias. As mentioned, the second step of strategy iteration requires to evaluate a given strategy. By investigating the Markov chain $\mathsf{M} = (S, s_{\text{init}}, \varDelta, r)$ induced by the MDP $\mathcal{M}$ together with a strategy $\pi \in \varPi^{\mathsf{MD}}$, one can employ the following system of linear equations characterizing the gain g [34]:

$$g(s) = \sum_{s' \in S} \varDelta(s, s') \cdot g(s') = \mathbb{E}_{\varDelta}(g, s) \quad \forall s \in S,$$
$$b(s) = \sum_{s' \in S} \varDelta(s, s') \cdot b(s') + r(s) - g(s) = \mathbb{E}_{\varDelta}(b, s) + r(s) - g(s) \quad \forall s \in S.$$

A solution (g, b) to these *gain/bias equations* yields the gain g and the so called *bias* b of the induced Markov chain, which we also refer to as gain g_π and bias b_π of the corresponding strategy π. Intuitively, the bias relates to the total expected deviation from the gain until the obtained rewards "stabilize" to the gain. Note that the equations uniquely determine the gain but not the bias. We refer the reader to [34, Sects. 9.1.1 and 9.2.1] for more detail but highlight the following result. A unique solution can be obtained by adding the constraints $b(s_i) = 0$ for one arbitrary but fixed state s_i in each BSCC [34, Condition 9.2.3]. Note this condition requires to fix the bias of the "first" state in the BSCC to zero. But, as the states can be numbered arbitrarily, any state of the BSCC is eligible. This is also briefly touched upon in the corresponding chapter of [34]. Unfortunately, this results in a non-square system matrix.

With these results, the strategy iteration for the average reward objective on MDPs is defined in Algorithm 1[4]. Reasoning of [34, Sect. 9.2.4] yields correctness.

Theorem 1. *The strategy iteration presented in Algorithm 1 terminates with a correct result for any input MDP.*

It might seem unintuitive why the bias improvement in Line 6 is necessary, since we are only interested in the gain after all. Intuitively, when optimizing the bias the algorithm seeks to improve the expected "bonus" until eventually stabilizing without reducing the obtained gain. This may lead to actually improving the overall gain, as illustrated in [30, Appendix C].

[4] Note that the procedure found in [34, Sect. 9.2.1] differs from our Algorithm in Line 6. There, the bias is improved over all available actions instead of the gain-optimal ones, which is erroneous. The proofs provided in the corresponding chapter actually prove the correctness of the algorithm as presented here.

Algorithm 1. SI

```
Input: MDP $\mathcal{M} = (S, s_{\text{init}}, Act, \mathsf{Av}, \Delta, r)$.
Output: $(g^*, \pi^*)$, s.t. $g^*$ is the optimal gain of the MDP and is obtained by $\pi^*$.
1: Set $n = 0$ and pick an arbitrary strategy $\pi_0 \in \Pi^{\mathsf{MD}}$.
2: Obtain $g_n$ and $b_n$ which satisfy the gain/bias equations.
3: Let                                                  ▷ Gain improvement
       $\mathsf{Av}_{g_n}(s) = \arg\max_{a \in \mathsf{Av}(s)} \mathbb{E}_\Delta(g_n, s, a)$,
   all actions maximizing the successor gains.
4: Pick $\pi_{n+1} \in \Pi^{\mathsf{MD}}$ s.t. $\pi_{n+1}(s) \in \mathsf{Av}_{g_n}(s)$, setting $\pi_{n+1}(s) = \pi_n(s)$ if possible.
5: if $\pi_{n+1} \neq \pi_n$ then increment $n$ by 1 and go to Line 2.
6: Pick $\pi_{n+1} \in \Pi^{\mathsf{MD}}$ which satisfies           ▷ Bias improvement
       $\pi_{n+1}(s) \in \arg\max_{a \in \mathsf{Av}_{g_n}(s)} r(s, a) + \mathbb{E}_\Delta(b_n, s, a)$,
   again setting $\pi_{n+1}(s) = \pi_n(s)$ if possible.
7: if $\pi_{n+1} \neq \pi_n$ then increment $n$ by 1 and go to Line 2.
8: return $(g_{n+1}, \pi_{n+1})$.
```

Advantages and Drawbacks of Strategy Iteration. Compared to other methods for solving the average reward objective, e.g. value iteration [2,10], strategy iteration offers some advantages:

(i) A *precise solution* can be obtained, compared to value iteration which is only optimal in the limit.
(ii) The gain of the strategy is *monotonically improving*, the iteration can be stopped at any point, yielding a strategy at least as good as the initial one.
(iii) It therefore is easy to *introduce knowledge* about the model or results of some pre-computation by initializing the algorithm with a sensible strategy.
(iv) On some models, strategy iteration performs *significantly faster* than value iteration, as outlined in [30, Appendix B].
(v) The algorithm searches through the *finite space* of memoryless deterministic strategies, simplifying termination and correctness proofs.

But on the other hand, the naive implementation of strategy iteration as presented in Algorithm 1 has several drawbacks:

(i) In order to determine the precise gain by solving the gain/bias equations, one necessarily has to determine the bias, too. Therefore, the algorithm has to determine *both gain and bias* in each step, while often only the gain is actually used for the improvement.
(ii) For reasonably sized models the equation system becomes *intractably large*. In the worst case, it contains $2n^2 + n$ non-zero entries and even for standard models there often are significantly more than n non-zero entries.

(iii) Furthermore, the gain/bias equation system is under-determined, ruling out a lot of fast solution methods for linear equation systems. Uniqueness can be introduced by adding several rows, which results in the matrix being non-square, again ruling out a lot of solution methods. Experimental results suggest that this equation system furthermore has rather large condition numbers (see [30, Appendix A]) even for small, realistic models, leading to numerical instabilities[5].

(iv) Lastly, the equation system is solved *precisely* for every improvement step, which often is unnecessary. To arrive at a precise solution, we often only need to identify states in which the strategy is not optimal, compared to having a precise measure of how non-optimal they are.

In the following two sections, we present approaches and ideas tackling each of the mentioned problems, arriving at procedures which perform orders of magnitude faster than the original approach.

4 Topological Optimizations

Our first set of optimizations is based on various topological arguments about both MDPs and MCs. They are used to eliminate unnecessary redundancies in the equation systems and identify sub-problems which can be solved separately, eventually leading to small, full-rank equation systems. Reduction in size and removal of redundancies naturally lead to significantly better condition numbers, which we also observed in our experiments.

Proofs of our claims can be found in [30, Appendix E].

4.1 MEC Decomposition

We presented a variant of this method in our previous work [2] in the context of value iteration. Due to space constraints we only give a short overview of the idea.

The central idea is that all states in a MEC of some MDP have the same optimal gain [34, Sect. 9.5][6]. Intuitively this is the case since any state in a particular MEC can reach every other state of the MEC almost surely. For some MEC M we define $g^*(M)$ to be this particular optimal value and call it the *gain of the MEC*. The optimal gain of the whole MDP then can be characterized by

$$g^*(s) = \max_{\pi \in \Pi^{\mathsf{MD}}} \sum_{M \in \mathsf{MEC}(\mathcal{M})} \mathbb{P}_s^\pi[\Diamond\Box M] \cdot g^*(M)$$

where $\Diamond\Box M$ denotes the measurable set of paths that eventually remain within M. This leads to a divide-and-conquer procedure for determining the gain of an

[5] On crafted models with less than 10 states we observed numerical errors leading to non-convergence and condition numbers of up to 10^5.

[6] Restricting a general MDP to a MEC results in a "communicating" MDP.

Algorithm 2. MEC-SI

```
Input: MDP $\mathcal{M} = (S, s_{\text{init}}, Act, \mathsf{Av}, \Delta, r)$.
Output: The optimal gain $g^*$ of the MDP.
1: $f \leftarrow \emptyset$, $r_{\max} \leftarrow \max_{s \in S, a \in \mathsf{Av}(s)} r(s, a)$.
2: for $M_i = (T_i, A_i) \in \mathsf{MEC}(\mathcal{M})$ do
3:     Compute $g^*(M_i)$ of the MEC by applying Algorithm 1 on the restricted MDP.
4:     Set $f(M_i) \leftarrow g^*(M_i)/r_{\max}$.
5: Compute the weighted MEC quotient $\mathcal{M}^f$.
6: Compute $p \leftarrow \mathbb{P}^{\max}_{\mathcal{M}^f}(\Diamond\{s_+\})$.
7: return $r_{\max} \cdot p$
```

MDP. Conceptually, the algorithm first computes the MEC decomposition [12], then for each MEC M determines its gain $g^*(M)$ by strategy iteration and finally solves a reachability query on the *weighted MEC quotient* $\mathcal{M}^f$ by, e.g., strategy iteration or (interval) value iteration [7,23].

The weighted MEC quotient $\mathcal{M}^f$ is a modification of the standard *MEC quotient* of [13], which for each MEC M introduces an action leading from the collapsed MEC M to a designated target sink s_+ with probability $f(M)$ (which is proportional to $g^*(M)$) and a non-target sink s_- with the remaining probability. With this construction, we can relate the maximal probability of reaching s_+ to the maximal gain in the original MDP. For a formal definition, see [30, Appendix D].

Using this idea, we define the first optimization of strategy iteration in Algorithm 2. Its correctness follows from [2, Theorem 2]. Since we are only concerned with the average reward and each state in the restriction can reach any other (under some strategy), the initial state we pick for the restriction in Line 3 is irrelevant. Note that while the restricted MDP consists of a single MEC, an induced Markov chain may still contain an arbitrary number of (B)SCCs.

This algorithm already performs significantly better on a lot of models, as shown by our experimental evaluation in Sect. 6. But, as to be expected, on models with large MECs this algorithm still is rather slow compared to, e.g., VI and may even add additional overhead when the whole model is a single MEC. To this end, we will improve strategy iteration in general. To combine these optimized variants with the ideas of Algorithm 2, one can simply apply them in Line 3.

4.2 Using Strongly Connected Components

The underlying ideas of the previous approach are independent of the procedure used to determine $g^*(M)$. Naturally, this optimization does not exploit any specific properties of strategy iteration to achieve the improvement. In this section, we will therefore focus on improving the core principle of strategy iteration, namely the evaluation of a particular strategy π on some MDP $\mathcal{M}$. As explained in Sect. 3, this problem is equivalent to determining the gain and bias of some

Markov chain M. Hence we fix such a Markov chain M throughout this section and present optimized methods for determining the required values precisely.

BSCC Compression. In this approach, we try to eliminate superfluous redundancies in the equation system. The basic idea is that all states in some BSCC have the same optimal gain. Moreover, the same gain is achieved in the *attractor* of B, i.e. all states from which almost all runs eventually end up in B.

Definition 3 (Attractor). *Let M be some Markov chain and $C \subseteq S$ some set of states in M. The* attractor *of C is defined as*

$$\mathsf{prob1}(C) := \{s \in S \mid \mathbb{P}_s[\Diamond C] = 1\},$$

i.e. the set of states which almost surely eventually reach C.

Lemma 1. *Let M be a Markov chain and B a BSCC. Then $g(s) = g(s')$ for all $s, s' \in \mathsf{prob1}(B)$.*

Proof. When interpreting the MC as a degenerate MDP with $|\mathsf{Av}(s)| = 1$ for all s, the gain of the MC coincides with the optimal gain of this MDP and each BSCC in the original MC is a MEC in the MDP. Using the reasoning from Sect. 4.1 and [34, Sect. 9.5], we obtain that all states in $\mathsf{prob1}(B)$ have the same gain. ☐

Therefore, instead of adding one gain variable per state to the equation system, we "compress" the gain of all states in the same BSCC (and its attractor) into one variable. Formally, the reduced equation system is formulated as follows.

Let $\{B_1, \dots, B_n\} = \mathsf{BSCC}(\mathsf{M})$ be the BSCC decomposition of the Markov chain. Further, define $A_i := \mathsf{prob1}(B_i)$ the attractors of each BSCC and $T := \bigcup_{i=1}^{n} A_i$ the set of all states which don't belong to any attractor. The *BSCC compressed gain/bias equations* then are defined as

$$\begin{aligned} g(s) &= \mathbb{E}_{\Delta}^{T'}(g, s) + \sum\nolimits_{A_i} \mathbb{E}_{\Delta}^{A_i}(g_i, s) \quad \forall s \in T, \\ b(s) &= \mathbb{E}_{\Delta}^{A_i}(b, s) + r(s) - g_i \quad \forall 1 \leq i \leq n, s \in A_i, \\ b(s) &= \mathbb{E}_{\Delta}(b, s) + r(s) - g(s) \quad \forall s \in T, \\ b(s_i) &= 0 \quad \text{for one arbitrary but fixed } s_i \in B_i,\ \forall 1 \leq i \leq n. \end{aligned} \tag{1}$$

Applying the reasoning of Lemma 1 immediately gives us correctness.

Corollary 1. *The values $g_1, \dots, g_n$, $g(s)$ and $b(s)$ are a solution to the equation system (1) if and only if*

$$g'(s) := \begin{cases} g_i & \textit{if } s \in A_i, \\ g(s) & \textit{otherwise.} \end{cases}$$

and $b(s)$ are a solution to the gain/bias equations.

Algorithm 3. SCC-SI

```
Input: MC $\mathsf{M} = (S, s_{\text{init}}, \Delta, r)$.
Output: $(g, b)$, s.t. $g$ and $b$ are solutions to the gain/bias equations.
 1: Obtain $\mathsf{BSCC}(\mathsf{M}) = \{B_1, \ldots, B_n\}$ and $\mathsf{SCC}(\mathsf{M}) \setminus \mathsf{BSCC}(\mathsf{M}) = \{S_1, \ldots, S_m\}$ with $S_i$
    in reverse topological order.
 2: for $B_i \in \mathsf{BSCC}(\mathsf{M})$ do                    ▷ Obtain gain and bias of BSCCs
 3:     Obtain $g_i$ and $b(s)$ for all $s \in B_i$ by solving the equations
            $b(s) = \mathbb{E}_{\Delta}^{B_i}(b, s) + r(s) - g_i \quad \forall s \in B_i,$
            $b(s_i) = 0 \quad \text{for one arbitrary but fixed } s_i \in B_i.$
 4:     Set $g(s) \leftarrow g_i$ for all $s \in B_i$.
 5: for $i$ from 1 to $m$ do                          ▷ Obtain gain and bias of non-BSCC states
 6:     Let $S^< := \bigcup_{j=1}^{i-1} S_j \cup \bigcup_{j=1}^{n} B_j$
 7:     Compute $\mathsf{succ}(\mathsf{g}) \leftarrow \{s' \in S^< \mid \exists s \in S_i.\ \Delta(s, s') > 0 \wedge g(s') = \mathsf{g}\}$.
 8:     Set $\mathsf{succg} = \{\mathsf{g} \mid \mathsf{succ}(\mathsf{g}) \neq \emptyset\}$.
 9:     For each $\mathsf{g} \in \mathsf{succg}$, obtain $p_{\mathsf{g}}$ by solving the equations
            $p_{\mathsf{g}}(s) = \mathbb{E}_{\Delta}^{S_i}(p_{\mathsf{g}}, s) + \sum_{s' \in \mathsf{succg}} \Delta(s, s') \quad \forall s \in S_i.$
10:     Set $g(s) \leftarrow \sum_{\mathsf{g} \in \mathsf{succg}} p_{\mathsf{g}}(s) \cdot \mathsf{g}$ for all $s \in S_i$.
11:     Obtain $b(s)$ for all $s \in S_i$ by solving the equations
            $b(s) = \mathbb{E}_{\Delta}^{S_i}(b, s) + \mathbb{E}_{\Delta}^{S^<}(b, s) + r(s) - g(s) \quad \forall s \in S_i.$
12: return $(g, b)$.
```

This equation system is significantly smaller for Markov chains which contain large BSCC-attractors. Furthermore, observe that the resulting system matrix also is square. We have $|\mathsf{BSCC}(\mathsf{M})| + |T|$ gain and $|S|$ bias variables but also $|T|$ gain and $|S| + |\mathsf{BSCC}(\mathsf{M})|$ bias equations. Additionally, by virtue of Corollary 1 and [34, Condition 9.2.3], the system has a unique solution. Together, this allows the use of more efficient solvers. Especially when combined with the previous MEC decomposition approach, significant speed-ups can be observed.

SCC Decomposition. The second approach extends the BSCC compression idea by further decomposing the problem into numerous sub-problems. A formal definition of the improved evaluation algorithm is given in Algorithm 3.

As with the compression approach, we exploit the fact that all states in some BSCC have the same gain. But instead of encoding this information into one big linear equation system, we use it to obtain multiple sub-problems.

First, we obtain gain and bias for each BSCC separately in Line 3. Note that there are only $|B_i| + 1$ variables and equations, since there only is a single gain variable. The last equation, setting bias to zero for some state of the BSCC, again induces a unique solution.

Now, these values are back-propagated through the MC. As mentioned, we can obtain a topological ordering of the SCCs, where a state s in a "later" SCC cannot reach any state s' in some earlier SCC. By processing the SCCs in reverse topological order, we can successively compute values of all states as follows.

Since the gain actually is only earned in BSCCs, the gain of some non-BSCC state naturally only depends on the probability of ending up in some BSCC. More generally, by a simple inductive argument, the gain of such a non-BSCC state only depends on the gains of the states it ends up in after moving to a later SCC. In other words, the gain only depends on the reachability of the successor gains. So, instead of constructing a linear equation system involving both gain and bias for each SCC, we determine the different "gain outcomes" in Line 8 and then compute the probability of these outcomes in Line 9, i.e. the probability of reaching a state obtaining some particular successor gain. Finally, we simply set the gain of some state as the expected outcome in Line 10. Only then the bias is computed in Line 11 by solving the bias equation with the computed gain values inserted as constants.

At first glance, this might seem rather expensive, as there are $|\mathsf{succg}|+1$ linear equation systems instead of one. But the corresponding matrices of the systems in Lines 9 and 11 actually are (i) square with a unique solution, allowing the use of LU decomposition; and (ii) are the same for a particular SCC, enabling reuse of the obtained decomposition.

Note how this in fact generalizes the idea of computing attractors in the BSCC-compression approach. Suppose a non-BSCC state $s \in S_j$ is in the attractor of a particular BSCC B_i. Since moving to B_i is the only possible outcome, succg as computed in Line 8 actually is a singleton set containing only the gain g_i of the BSCC. Then $p_{g_i}(s) = 1$ for all states in S_j and we can immediately set $g(s) = g_i$.

5 Approximation-Guided Solutions

This section introduces another idea to increase efficiency of the strategy iteration. Section 5.1 then combines this method with optimizations of the previous section in a non-trivial way, yielding a super-additive optimization effect. Our new approach relies on the following observation. In order to improve a strategy, it is not always necessary to know the exact gain in each state; sufficiently tight bounds are enough to decide that the current action is sub-optimal. To this end, we assume that we are given an approximative oracle for the gain of any state under some strategy[7]. Formally, we require a function $g^{\approx} : \Pi^{\mathsf{MD}} \times S \to \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$ and call it *consistent* if for $g^{\approx}(\pi, s) = (g_L(\pi, s), g_U(\pi, s))$ we have that $g^{\pi}(s) \in [g_L(\pi, s), g_U(\pi, s)]$. For readability, we write $g_L(\pi)$ and $g_U(\pi)$ for the functions $s \mapsto g_L(\pi, s)$ and $s \mapsto g_U(\pi, s)$, respectively.

In Algorithm 4, we define a variant of strategy iteration, which incorporates this approximation for gain improvement. Let us focus on this improvement in

[7] We will go into detail why we do not deal with bias later on.

Algorithm 4. Approx-SI

```
Input: MDP $\mathcal{M} = (S, s_{\text{init}}, Act, \mathsf{Av}, \Delta, r)$ and consistent gain approximation $g^{\approx}$.
Output: $(g^*, \pi^*)$, s.t. $g^*$ is the optimal gain of the MDP and is obtained by $\pi^*$.
1: Set $n \leftarrow 0$, and pick an arbitrary strategy $\pi_0 \in \Pi^{\mathsf{MD}}$.
2: Set $\pi_{n+1} = \pi_n$
3: for $s \in S$ do                                        ▷ Approximate gain improvement
4:     if $g_U(\pi_n, s) < \max_{a \in \mathsf{Av}(s)} \mathbb{E}_\Delta(g_L(\pi_n), s)$ then
5:         Pick $\pi_{n+1} \in \arg\max_{a \in \mathsf{Av}(s)} \mathbb{E}_\Delta(g_L(\pi_n), s, a)$.
6: if $\pi_{n+1} \neq \pi_n$ then increment $n$ by 1, go to Line 2.
7: Obtain $g_{n+2}$ and $\pi_{n+2}$ by one step of precise SI.      ▷ Precise improvement
8: if $\pi_{n+2} \neq \pi_{n+1}$ then increment $n$ by 2, go to Line 2.
9: return $(g_{n+2}, \pi_{n+2})$
```

Line 5. There are three cases to distinguish. (1) If the test on Line 4 holds, i.e. the upper bound on the gain in the current state is smaller than the lower bound under some other action a, then a definitely gives us a better gain. Therefore, we switch the strategy to this action. If the test does not hold, there are two other cases to distinguish: (2) If in contrast, the lower bound on the gain in the current state is bigger than the upper bound under any other action, the current gain definitely is better than the gain achievable under any other action. Hence the current action is optimal and the strategy should not be changed. (3) Otherwise, the approximation does not offer us enough information to conclude anything. The current action is neither a clear winner nor a clear loser compared to the other actions. In this case we also refrain from changing the strategy. Intuitively, if there are any changes to be done in Case (3), we postpone them until no further improvements can be done based solely on the approximations. They will be dealt with in Line 7, where we determine the gain precisely.

Theorem 2. *Algorithm 4 terminates for any MDP and consistent gain approximation function. Furthermore, the gain and corresponding strategy returned by the algorithm is optimal.*

Implementing Gain Approximations. In order to make Algorithm 4 practical, we provide a prototype for such a gain approximation. To this end, we can again interpret the MC M as a degenerate MDP $\mathcal{M}$ and apply variants of the value iteration methods of [2, Algorithm 2]. We want to emphasize that there are no restrictions on the oracle except consistency, hence there may be other, faster methods applicable here. This also opens the door for more fine-tuning and optimizations. For instance, instead of "giving up" on the estimation and solving the equations precisely, the gain approximation could be asked to refine the estimate for all states where there is uncertainty and Case (3) occurs.

Difficulties in Using Bias Estimations. One may wonder why we did not include a bias estimation function in the previous algorithm. There are two

Procedure 5. MEC-Approx

```
1: Set $g_L^{\max}(\pi_n) \leftarrow \max_{s \in M} g_L(\pi_n, s)$, $S_- \leftarrow \{s \mid g_U(\pi_n, s) < g_L^{\max}(\pi_n)\}$, $S_+ = M \setminus S_-$.
2: if $S_- = \emptyset$ then Continue with precise improvement.
3: else
4:     while $S_- \neq \emptyset$ do
5:         Obtain $s \in S_-$ and $a \in \mathsf{Av}(s)$ such that $\sum_{s' \in S_+} \Delta(s, a, s') > 0$.
6:         Set $\pi_{n+1}(s) \leftarrow a$, $S_+ \leftarrow S_+ \cup \{s\}$, $S_- \leftarrow S_- \setminus \{s\}$.
7:     Increment $n$ by 1, go to Line 1.
```

main reasons for this, namely (i) by naively using the bias approximation, the algorithm may not converge any more (even with ε-precise approximations) and (ii) it seems rather difficult to efficiently obtain a reasonable bias estimate. We provide more detail and intuition in [30, Appendix F].

5.1 Synergy of the Approaches

In order to further improve the approximation-guided approach, we combine it with the idea of MEC decomposition, which in turn allows for even more optimizations. As already mentioned, each state in a MEC has the same optimal gain. In combination with the idea of the algorithm in [34, Sect. 9.5.1], this allows us to further enhance the gain improvement step as follows.

The gain $g^*(M)$ of some MEC M certainly is higher than the lower bound achieved through some strategy in any state of the MEC, which is $g_L^{\max}(\pi_n) := \max_{s \in M} g_L(\pi_n, s)$. Hence, any state of the MEC which has an upper bound less than $g_L^{\max}(\pi_n)$ is suboptimal, as we can adapt the strategy such that it achieves at least this value in every state of the MEC. With this, the gain improvement step can be changed to (i) determine the maximal lower bound $g_L^{\max}(\pi_n)$, (ii) identify all states S_+ which have an upper bound greater than this lower bound and (iii) update the strategy in all other states S_- to move to this "optimal" region. Algorithm 5 then is obtained by replacing the approximate gain improvement in Lines 3 to 2 of Algorithm 4 by Procedure 5.

Theorem 3. *Algorithm 5 terminates for any MDP and consistent gain approximation function. Furthermore, the gain and corresponding strategy returned by the algorithm indeed is optimal.*

6 Experimental Evaluation

In this section, we compare the presented approaches to established tools.

Implementation Details. We implemented our constructions[8] in the PRISM Model Checker [31]. We also added several general purpose optimizations to

[8] Accessible at https://www7.in.tum.de/~meggendo/artifacts/2017/atva_si.txt.

PRISM, improving the used data structures. This may influence the comparability of these results to other works implemented in PRISM.

In order to solve the arising systems of linear equations, we used the `ojAlgo` Java library[9]. Whenever possible, we employed LU decomposition to solve the equation systems and SVD otherwise. We use `double` precision for all computations, which implies that results are only precise modulo numerical issues. The implementation can easily be extended to arbitrary precision, at the cost of performance. Further, our implementation only uses the parallelization of `ojAlgo`. Since the vast majority of computation time is consumed by solving equation systems, we did not implement further parallelization.

Experimental Setup. All benchmarks have been run on a `4.4.3-gentoo` x64 virtual machine with 16 cores of 3.0 GHz each, a time limit of 10 min and memory limit of 32 GB, using the 64-bit Oracle JDK version `1.8.0_102-b14`. All time measurements are given in seconds and are averaged over 10 executions. Instead of measuring the time which is spent in a particular algorithm, we decide to measure the overall *user CPU time* of the PRISM process using the UNIX tool `time`. This metric has several advantages. It allows for an easy and fair comparison between, e.g., multithreaded executions, symbolic methods or implementations which do not construct the whole model. Further, it reduces variance in measurements caused by the operating system through, e.g., the scheduler. Note that this also allows for measurements larger than the specified timeout, as the process may spend this timeout on each of the 16 cores.

6.1 Models

We briefly explain the examples used for evaluation. **virus** [32] models a *virus spreading through a network*. We reward each attack carried out by an infected machine. **cs_nfail** [29] models a *client-server mutual exclusion protocol* with probabilistic failures of the clients. A reward is given for each successfully handled connection. **phil_nofair** [14] represents the (randomised) *dining philosophers* without fairness assumptions. We use two reward structures, rewarding "thinking" and "eating", respectively. **sensor** [29] models a *network of sensors* sending values to a central processor over a lossy connection. Processing received data is rewarded. **mer** [18] captures the behaviour of a *resource arbiter* on a Mars exploration rover. We reward each time some user is granted access to a resource by the arbiter.

6.2 Tools

Since we are unaware of other implementations, we implemented standard SI as in Algorithm 1 by ourselves. We compare the following variants of SI.

[9] http://ojalgo.org/.

- `SI`: Standard SI as presented in Algorithm 1.
- `BSCC`: SI with BSCC compression gain/bias equations.
- `SCC`: The SCC decomposition approach of Algorithm 3.
- $\texttt{SCC}_A$: The SCC decomposition approach combined with the approximation methods from Sect. 5.

Further, a "M" superscript denotes use of the MEC decomposition approach as in Algorithm 2. In the case of $\texttt{SCC}_A^M$, we use the improved method of Sect. 5.1. More details and evaluation of some further variants can be found in [30, Appendix G]. During our experiments, we observed that the algorithm used to solve the resulting reachability problem did not influence the results significantly, since the weighted quotients are considerably simpler than the original models.

We compare our methods to the value iteration approach we presented in [2, Algorithm 2] with a required precision of 10^{-8} (`VI`). This comparison has to be evaluated with care, since (i) value iteration inherently is only ε-precise and (ii) it needs a MEC decomposition for soundness. Note that topological optimizations for value iteration as suggested by, e.g., [4] are partially incorporated by `VI`, since each MEC is iterated separately.

We also provide a comparison to the LP-based MultiGain [8] in [30, Appendix G]. In summary, the LP approach is soundly beaten by our optimized approaches. A more detailed comparison can be found in [2].

We are unaware of other implementations capable of solving the mean payoff objective for MDPs. Neither did we find a mean payoff solver for stochastic games which we could easily set up to process the PRISM models.

6.3 Results

We will highlight various conclusions to be drawn from Table 1. Comparing the naive SI with our enhanced versions `BSCC` and `SCC`, the number of strategy improvements does not differ, but the evaluation of each strategy is significantly faster, yielding the differences displayed in the table.

On the smaller models (**cs_nfail** and **virus**) nearly all of the optimized methods perform comparable, a majority of the runtime actually is consumed by the start-up of PRISM. Especially on **virus**, all the MEC-decomposition approaches have practically the same execution time due to the model only having a single MEC with a single state, which the problem trivial for these approaches.

The results immediately show how intractable naive strategy iteration is. On models with only a few hundred states, the computation already times out. The BSCC compression approach `BSCC` suffers from the same issues, but already performs significantly better than `SI`. In particular, when combined with MEC decomposition, it is able to solve more models within the given time.

Further, we see immense benefits of using the SCC approach, regularly beating even the quite performant (and imprecise) value iteration approach. Interestingly, the variants using approximation often perform worse than the "pure" SCC method. We conjecture that this is due the gain approximation function we used. It computes the gain up to some adaptively chosen precision instead of

Table 1. Comparison of various variants on the presented models. Timeouts and memouts are denoted by a hyphen. The best results in each row are marked in bold, excluding `VI`. The number of states and MECs are written next to the model.

Model	`SI`	`SI`^{M}	`BSCC`	`BSCC`^{M}	`SCC`	`SCC`$_A$	`SCC`$_A^M$	`VI`
cs_nfail3 (184, 38)	17	**4**	**4**	**4**	**4**	**4**	**4**	4
cs_nfail4 (960, 176)	1129	6	16	**5**	**5**	**5**	6	5
virus (809, 1)	–	**4**	10	**4**	5	5	**4**	4
phil_nofair3 (856, 1)	–	–	112	112	**6**	10	7	5
phil_nofair4 (9440, 1)	–	–	–	–	**15**	310	107	18
sensors1 (462, 132)	–	13	23	**4**	**4**	6	**4**	5
sensors2 (7860, 4001)	–	89	–	14	13	168	**11**	15
sensors3 (77766, 46621)	–	–	–	78	**40**	–	46	72
mer3 (15622, 9451)	–	21	–	26	**16**	244	22	15
mer4 (119305, 71952)	–	58	–	281	**42**	–	84	64
mer5 (841300, 498175)	–	–	–	–	**474**	–	–	–

computing up to a certain number of iterations. Changing this precision bound gave mixed results, on some models performance increased, on some it decreased. Comparing the two approximation-based approaches `SCC`$_A$ and `SCC`$_A^M$, we highlight the improvements of Algorithm 5, speeding up convergence even though a MEC decomposition is computed.

Finally, we want to emphasize the `mer` results. Here, our `SCC` approach manages to obtain a solution within the time- and memory-bound, while all other approaches, including VI, fail due to a time-out.

7 Conclusion

We have proposed and evaluated several techniques to speed up strategy iteration. The combined speed ups are in orders of magnitude. This makes strategy iteration competitive even with the most used and generally imprecise value iteration and shows the potential of strategy iteration in the context of MDPs.

In future work, we will further develop this potential. Firstly, building upon the *SCC decomposition*, we can see opportunities to interleave the SCC computation and the improvements of the current strategy. Secondly, the *gain approximation* technique used is quite naive. Here we could further adapt our recent results on VI [2], in order to improve the performance of the approximation. Besides, we suggest to use simulations to evaluate the strategies. Nevertheless, the incomplete confidence arising form stochastic simulation has to be taken into account here. Thirdly, techniques for efficient *bias* approximation and algorithms to utilize it would be desirable. Finally, a fully configurable tool would be helpful to find the sweet-spot combinations of these techniques and useful as the first scalable tool for mean payoff optimization in MDPs.

Acknowledgments. We thank the anonymous reviewers for their insightful comments and valuable suggestions. In particular, a considerable improvement to the BSCC compression approach of Sect. 4.2 has been proposed.

References

1. Abate, A., Češka, M., Kwiatkowska, M.: Approximate policy iteration for Markov Decision Processes via quantitative adaptive aggregations. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 13–31. Springer, Cham (2016). doi:10.1007/978-3-319-46520-3_2
2. Ashok, P., Chatterjee, K., Daca, P., Křetínský, J., Meggendorfer, T.: Value iteration for long-run average reward in Markov Decision Processes. In: CAV (2017). To appear
3. Baier, C., Katoen, J.-P.: Principles of Model Checking (2008)
4. Baier, C., Klein, J., Leuschner, L., Parker, D., Wunderlich, S.: Ensuring the reliability of your model checker: Interval iteration for Markov Decision Processes. In: CAV (2017). To appear
5. Bertsekas, D.P.: Approximate policy iteration: a survey and some new methods. J. Control Theor. Appl. **9**(3), 310–335 (2011)
6. Björklund, H., Vorobyov, S.G.: A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. DAM **155**(2), 210–229 (2007)
7. Brázdil, T., Chatterjee, K., Chmelík, M., Forejt, V., Křetínský, J., Kwiatkowska, M., Parker, D., Ujma, M.: 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). doi:10.1007/978-3-319-11936-6_8
8. Brázdil, T., Chatterjee, K., Forejt, V., Kučera, A.: MultiGain: a controller synthesis tool for MDPs with multiple mean-payoff objectives. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 181–187. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46681-0_12
9. Brim, L., Chaloupka, J.: Using strategy improvement to stay alive. IJCSIS **23**(3), 585–608 (2012)
10. Chatterjee, K., Henzinger, T.: Value iteration. 25 Years of Model Checking, pp. 107–138 (2008)
11. Condon, A.: On algorithms for simple stochastic games. In: Advances in Computational Complexity Theory, pp. 51–72 (1990)
12. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J. ACM **42**(4), 857–907 (1995)
13. de Alfaro, L.: Formal verification of probabilistic systems. Ph.D thesis (1997)
14. Duflot, M., Fribourg, L., Picaronny, C.: Randomized dining philosophers without fairness assumption. Distrib. Comput. **17**(1), 65–76 (2004)
15. Fearnley, J.: Exponential lower bounds for policy iteration. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 551–562. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14162-1_46
16. Fearnley, J.: Strategy iteration algorithms for games and Markov Decision Processes. Ph.D thesis, University of Warwick (2010)
17. Fearnley, J.: Efficient parallel strategy improvement for parity games. In: CAV (2017). To appear

18. Feng, L., Kwiatkowska, M., Parker, D.: Automated learning of probabilistic assumptions for compositional reasoning. In: Giannakopoulou, D., Orejas, F. (eds.) FASE 2011. LNCS, vol. 6603, pp. 2–17. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19811-3_2
19. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer, New York (1997)
20. Frausto-Solis, J., Santiago, E., Mora-Vargas, J.: Cosine policy iteration for solving infinite-horizon Markov Decision Processes. In: Aguirre, A.H., Borja, R.M., Garciá, C.A.R. (eds.) MICAI 2009. LNCS, vol. 5845, pp. 75–86. Springer, Heidelberg (2009). doi:10.1007/978-3-642-05258-3_7
21. Friedmann, O.: An exponential lower bound for the parity game strategy improvement algorithm as we know it. In: LICS, pp. 145–156 (2009)
22. Gawlitza, T.M., Schwarz, M.D., Seidl, H.: Parametric strategy iteration. arXiv preprint arXiv:1406.5457 (2014)
23. Haddad, S., Monmege, B.: Reachability in MDPs: refining convergence of value iteration. In: Ouaknine, J., Potapov, I., Worrell, J. (eds.) RP 2014. LNCS, vol. 8762, pp. 125–137. Springer, Cham (2014). doi:10.1007/978-3-319-11439-2_10
24. Hahn, E.M., Schewe, S., Turrini, A., Zhang, L.: Synthesising strategy improvement and recursive algorithms for solving 2.5 player parity games. In: Bouajjani, A., Monniaux, D. (eds.) VMCAI 2017. LNCS, vol. 10145, pp. 266–287. Springer, Cham (2017). doi:10.1007/978-3-319-52234-0_15
25. Hansen, K.A., Ibsen-Jensen, R., Miltersen, P.B.: The complexity of solving reachability games using value and strategy iteration. Theor. Comput. Syst. **55**(2), 380–403 (2014)
26. Hansen, T.D., Miltersen, P.B., Zwick, U.: 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)
27. Hordijk, A., Puterman, M.L.: On the convergence of policy iteration in finite state undiscounted Markov Decision Processes: the unichain case. MMOR **12**(1), 163–176 (1987)
28. Howard, R.A.: Dynamic Programming and Markov Processes (1960)
29. Komuravelli, A., Păsăreanu, C.S., Clarke, E.M.: Assume-guarantee abstraction refinement for probabilistic systems. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 310–326. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31424-7_25
30. Křetínský, J., Meggendorfer, T.: Efficient strategy iteration for mean payoff in Markov Decision Processes. Technical report abs/1707.01859. arXiv.org (2017)
31. 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). doi:10.1007/978-3-642-22110-1_47
32. Kwiatkowska, M., Norman, G., Parker, D., Vigliotti, M.G.: Probabilistic mobile ambients. Theoret. Comput. Sci. **410**(12–13), 1272–1303 (2009)
33. Luttenberger, M.: Strategy iteration using non-deterministic strategies for solving parity games. CoRR, abs/0806.2923 (2008)
34. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley (2014)
35. Schewe, S.: An optimal strategy improvement algorithm for solving parity and payoff games. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 369–384. Springer, Heidelberg (2008). doi:10.1007/978-3-540-87531-4_27
36. Shlakhter, O., Lee, C.-G.: Accelerated modified policy iteration algorithms for Markov Decision Processes. MMOR **78**(1), 61–76 (2013)

37. Tarjan, R.: Depth-first search and linear graph algorithms. SICOMP **1**(2), 146–160 (1972)
38. Vöge, J., Jurdziński, M.: A discrete strategy improvement algorithm for solving parity games. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 202–215. Springer, Heidelberg (2000). doi:10.1007/10722167_18
39. Ye, Y.: The simplex and policy-iteration methods are strongly polynomial for the Markov decision problem with a fixed discount rate. MMOR **36**(4), 593–603 (2011)

C Rabinizer 4: From LTL to Your Favourite Deterministic Automaton. CAV 2018

This section has been published as **peer-reviewed conference paper**.

> Jan Křetínský, Tobias Meggendorfer, Salomon Sickert and Christopher Ziegler. 'Rabinizer 4: From LTL to Your Favourite Deterministic Automaton'. In: *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I.* ed. by Hana Chockler and Georg Weissenbacher. Vol. 10981. Lecture Notes in Computer Science. Springer, 2018, pp. 567–577. DOI: `10.1007/978-3-319-96145-3\_30`. URL: `https://doi.org/10.1007/978-3-319-96145-3%5C_30`

Synopsis We present Rabinizer 4, a tool set for translating *linear temporal logic* (LTL) to various kinds of automata. It is a complete rewrite of the previous version Rabinizer 3.1 [EKS16] with many under-the-hood improvements and significant extensions to general applicability as well as usability. Key contributions include:

- Several new types of translations, e.g., LTL to LDBA, and DRA to DPA.
- The first translation of *frequency LTL* to *generalized Rabin mean payoff automata*.
- Significant performance improvements to all constructions due to both specific and general optimizations.
- Support of a wider range of input and output formats, such as the complete syntax of LTL, the PGsolver game format, etc.

Contributions of the thesis author Discussion and revision of the entire manuscript. Significant contributions towards the overall design of the tool. Sole design and (re-)implementation of the `ltl2dgra` sub-tool, loosely based on Rabinizer 3.1, as well as `dra2dpa`, based on [Kře+17].

Rabinizer 4: From LTL to Your Favourite Deterministic Automaton

Jan Křetínský[(✉)], Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler

Technical University of Munich, Munich, Germany
jan.kretinsky@gmail.com, {meggendo,sickert}@in.tum.de

Abstract. We present Rabinizer 4, a tool set for translating formulae of linear temporal logic to different types of deterministic ω-automata. The tool set implements and optimizes several recent constructions, including the first implementation translating the frequency extension of LTL. Further, we provide a distribution of PRISM that links Rabinizer and offers model checking procedures for probabilistic systems that are not in the official PRISM distribution. Finally, we evaluate the performance and in cases with any previous implementations we show enhancements both in terms of the size of the automata and the computational time, due to algorithmic as well as implementation improvements.

1 Introduction

Automata-theoretic approach [VW86] is a key technique for verification and synthesis of systems with linear-time specifications, such as formulae of linear temporal logic (LTL) [Pnu77]. It proceeds in two steps: first, the formula is translated into a corresponding automaton; second, the product of the system and the automaton is further analyzed. The size of the automaton is important as it directly affects the size of the product and thus largely also the analysis time, particularly for deterministic automata and probabilistic model checking in a very direct proportion. For verification of non-deterministic systems, mostly non-deterministic Büchi automata (NBA) are used [EH00,SB00,GO01,GL02, BKŘS12,DLLF+16] since they are typically very small and easy to produce.

Probabilistic LTL model checking cannot profit directly from NBA. Even the qualitative question, whether a formula holds with probability 0 or 1, requires automata with at least a restricted form of determinism. The prime example are the limit-deterministic (also called semi-deterministic) Büchi automata (LDBA) [CY88] and the generalized LDBA (LDGBA). However, for the general quantitative questions, where the probability of satisfaction is computed, general limit-determinism is not sufficient. Instead, deterministic Rabin automata (DRA) have

This work has been partially supported by the Czech Science Foundation grant No. P202/12/G061 and the German Research Foundation (DFG) project KR 4890/1-1 "Verified Model Checkers" (317422601). A part of the frequency extension has been implemented within Google Summer of Code 2016.

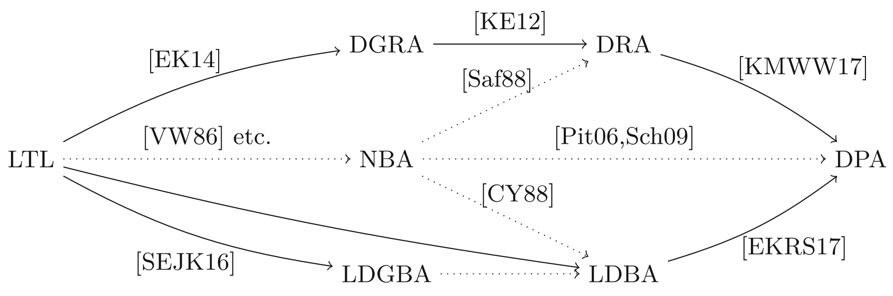


Fig. 1. LTL translations to different types of automata. Translations implemented in Rabinizer 4 are indicated with a solid line. The traditional approaches are depicted as dotted arrows. The determinization of NBA to DRA is implemented in ltl2dstar [Kle], to LDBA in Seminator [BDK+17] and to (mostly) DPA in spot [DLLF+16].

been mostly used [KNP11] and recently also deterministic generalized Rabin automata (DGRA) [CGK13]. In principle, all standard types of deterministic automata are applicable here except for deterministic Büchi automata (DBA), which are not as expressive as LTL. However, other types of automata, such as deterministic Muller and deterministic parity automata (DPA) are typically larger than DGRA in terms of acceptance condition or the state space, respectively.[1] Recently, several approaches with specific LDBA were proved applicable to the quantitative setting [HLS+15, SEJK16] and competitive with DGRA. Besides, model checking MDP against LTL properties involving frequency operators [BDL12] also allows for an automata-theoretic approach, via deterministic generalized Rabin mean-payoff automata (DGRMA) [FKK15].

LTL synthesis can also be solved using the automata-theoretic approach. Although DRA and DGRA transformed into games can be used here, the algorithms for the resulting Rabin games [PP06] are not very efficient in practice. In contrast, DPA may be larger, but in this setting they are the automata of choice due to the good practical performance of parity-game solvers [FL09, ML16, JBB+17].

Types of Translations. The translations of LTL to NBA, e.g., [VW86], are typically *"semantic"* in the sense that each state is given by a set of logical formulae and the language of the state can be captured in terms of semantics of these formulae. In contrast, the determinization of Safra [Saf88] or its improvements [Pit06, Sch09, TD14, FL15] are not "semantic" in the sense that they ignore the structure and produce trees as the new states that, however, lack the logical interpretation. As a result, if we apply Safra's determinization on semantically created NBA, we obtain DRA that lack the structure and, moreover, are unnecessarily large since the construction cannot utilize the original structure. In contrast, the

[1] Note that every DGRA can be written as a Muller automaton on the same state space with an exponentially-sized acceptance condition, and DPA are a special case of DRA and thus DGRA.

recent works [KE12,KLG13,EK14,KV15,SEJK16,EKRS17,MS17,KV17] provide "semantic" constructions, often producing smaller automata. Furthermore, various transformations such as degeneralization [KE12], index appearance record [KMWW17] or determinization of limit-deterministic automata [EKRS17] preserve the semantic description, allowing for further optimizations of the resulting automata.

Our Contribution. While all previous versions of Rabinizer [GKE12,KLG13, KK14] featured only the translation LTL→DGRA→DRA, Rabinizer 4 now implements all the translations depicted by the solid arrows in Fig. 1. It improves all these translations, both algorithmically and implementation-wise, and moreover, features the first implementation of the translation of a frequency extension of LTL [FKK15].

Further, in order to utilize the resulting automata for verification, we provide our own distribution[2] of the PRISM model checker [KNP11], which allows for model checking MDP against LTL using not only DRA and DGRA, but also using LDBA and against frequency LTL using DGRMA. Finally, the tool can turn the produced DPA into parity games between the players with input and output variables. Therefore, when linked to parity-game solvers, Rabinizer 4 can be also used for LTL synthesis.

Rabinizer 4 is freely available at http://rabinizer.model.in.tum.de together with an on-line demo, visualization, usage instructions and examples.

2 Functionality

We recall that the previous version Rabinizer 3 has the following functionality:

- It translates LTL formulae into equivalent DGRA or DRA.
- It is linked to PRISM, allowing for probabilistic verification using DGRA (previously PRISM could only use DRA).

2.1 Translations

Rabinizer 4 inputs formulae of LTL and outputs automata in the standard HOA format [BBD+15], which is used, e.g., as the input format in PRISM. Automata in the HOA format can be directly visualized, displaying the "semantic" description of the states. Rabinizer 4 features the following command-line tools for the respective translations depicted as the solid arrows in Fig. 1:

ltl2dgra and **ltl2dra** correspond to the original functionality of Rabinizer 3, i.e., they translate LTL (now with the extended syntax, including all common temporal operators) to DGRA and DRA [EK14], respectively.

[2] Merging these features into the public release of PRISM as well as linking the new version of Rabinizer is subject to current collaboration with the authors of PRISM.

ltl2ldgba and **ltl2ldba** translate LTL to LDGBA using the construction of [SEJK16] and to LDBA, respectively. The latter is our modification of the former, which produces smaller automata than chaining the former with the standard degeneralization.

ltl2dpa translates LTL to DPA using two modes:

- The default mode uses the translation to LDBA, followed by a LDBA-to-DPA determinization [EKRS17] specially tailored to LDBA with the "semantic" labelling of states, avoiding additional exponential blow-up of the resulting automaton.
- The alternative mode uses the translation to DRA, followed by our improvement of the index appearance record of [KMWW17].

fltl2dgrma translates the frequency extension of $\text{LTL}_{\setminus\mathbf{GU}}$, i.e. $\text{LTL}_{\setminus\mathbf{GU}}$ [KLG13] with $\mathbf{G}^{\sim\rho}$ operator[3], to DGRMA using the construction of [FKK15].

2.2 Verification and Synthesis

The resulting automata can be used for model checking probabilistic systems and for LTL synthesis. To this end, we provide our own distribution of the probabilistic model checker PRISM as well as a procedure transforming automata into games to be solved.

Model checking: PRISM distribution. For model checking Markov chains and Markov decision processes, PRISM [KNP11] uses DRA and recently also more efficient DGRA [CGK13,KK14]. Our distribution, which links Rabinizer, additionally features model checking using the LDBA [SEJK16, SK16] that are created by our **ltl2ldba**.
Further, the distribution provides an implementation of frequency $\text{LTL}_{\setminus\mathbf{GU}}$ model checking, using DGRMA. To the best of our knowledge, there are no other implemented procedures for logics with frequency. Here, techniques of linear programming for multi-dimensional mean-payoff satisfaction [CKK15] and the model-checking procedure of [FKK15] are implemented and applied.

Synthesis: Games. The automata-theoretic approach to LTL synthesis requires to transform the LTL formula into a game of the input and output players. We provide this transformer and thus an end-to-end LTL synthesis solution, provided a respective game solver is linked. Since current solutions to Rabin games are not very efficient we implemented a transformation of DPA into parity games and a serialization to the format of PG Solver [FL09]. Due to the explicit serialization, we foresee the main use in quick prototyping.

[3] The *frequential globally* construct [BDL12,BMM14] $\mathbf{G}^{\sim\rho}\varphi$ with $\sim\ \in \{\geq, >, \leq, <\}, \rho \in [0,1]$ intuitively means that the fraction of positions satisfying φ satisfies $\sim\rho$. Formally, the fraction on an infinite run is defined using the long-run average [BMM14].

3 Optimizations, Implementation, and Evaluation

Compared to the theoretical constructions and previous implementations, there are numerous improvements, heuristics, and engineering enhancements. We evaluate the improvements both in terms of the size of the resulting automaton as well as the running time. When comparing with respect to the original Rabinizer functionality, we compare our implementation **ltl2dgra** to the previous version Rabinizer 3.1, which is already a significantly faster [EKS16] re-implementation of the official release Rabinizer 3 [KK14]. All of the benchmarks have been executed on a host with i7-4700MQ CPU (4x2.4 GHz), running Linux 4.9.0-5-amd64 and the Oracle JRE 9.0.4+11 JVM. Due to the start-up time of JVM, all times below 2 s are denoted by <2 and not specified more precisely. All experiments were given a time-out of 900 s and mem-out of 4GB, denoted by $-$.

Algorithmic improvements and heuristics for each of the translations:

ltl2dgra and **ltl2dra.** These translations create a master automaton monitoring the satisfaction of the given formula and a dedicated slave automaton for each subformula of the form $\mathbf{G}\psi$ [EK14]. We (i) simplify several classes of slaves and (ii) "suspend" (in the spirit of [BBDL+13]) some so that they appear in the final product only in some states. The effect on the size of the state space is illustrated in Table 1 on a nested formula. Further, (iii) the acceptance condition is considered separately for each strongly connected component (SCC) and then combined. On a concrete example of Table 2, the automaton for $i = 8$ has 31 atomic propositions, whereas the number of atomic propositions relevant in each component of the master automaton is constant, which we utilize and thus improve performance on this family both in terms of size and time.

ltl2ldba. This translation is based on breakpoints for subformulae of the form $\mathbf{G}\psi$. We provide a heuristic that avoids breakpoints when ψ is a safety or co-safety subformula, see Table 3.
Besides, we add an option to generate a non-deterministic initial component for the LDBA instead of a deterministic one. Although the LDBA is then no more suitable for quantitative probabilistic model checking, it still is for qualitative model checking. At the same time, it can be much smaller, see Table 4 which shows a significant improvement on the particular formula.

ltl2dpa. Both modes inherit the improvements of the respective **ltl2ldba** and **ltl2dgra** translations. Further, since complementing DPA is trivial, we can run in parallel both the translation of the input formula and of its negation, returning the smaller of the two results. Finally, we introduce several heuristics to optimize the treatment of safety subformulae of the input formula.

dra2dpa. The index appearance record of [KMWW17] keeps track of a permutation (ordering) of Rabin pairs. To do so, all ties between pairs have to be resolved. In our implementation, we keep a pre-order instead, where irrelevant ties are not resolved. Consequently, it cannot happen that an irrelevant tie is resolved in two different ways like in [KMWW17], thus effectively merging such states.

Table 1. Effect of simplifications and suspension for **ltl2dgra** on the formulae $\psi_i = \mathbf{G}\phi_i$ where $\phi_1 = a_1, \phi(i) = (a_i \mathbf{U}(\mathbf{X}\phi_{i-1}))$, and $\psi'_i = \mathbf{G}\phi'_i$ where $\phi'_1 = a_1$, $\phi'_1 = (\phi'_{i-1}\mathbf{U}(\mathbf{X}^i a_i))$, displaying execution time in seconds/#states.

	ψ_2	ψ_3	ψ_4	ψ_5	ψ_6
Rabinizer 3.1 [EKS16]	<2/4	<2/16	<2/73	3/332	60/1463
ltl2dgra	<2/3	<2/7	<2/35	3/199	13/1155
	ψ'_2	ψ'_3	ψ'_4	ψ'_5	ψ'_6
Rabinizer 3.1 [EKS16]	<2/4	<2/16	2/104	128/670	–
ltl2dgra	<2/3	<2/10	<2/38	7/175	239/1330

Table 2. Effect of computing acceptance sets per SCC on formulae $\psi_1 = x_1 \wedge \phi_1$, $\psi_2 = (x_1 \wedge \phi_1) \vee (\neg x_1 \wedge \phi_2)$, $\psi_3 = (x_1 \wedge x_2 \wedge \phi_1) \vee (\neg x_1 \wedge x_2 \wedge \phi_2) \vee (x_1 \wedge \neg x_2 \wedge \phi_3), \ldots,$ where $\phi_i = \mathbf{XG}((a_i \mathbf{U} b_i) \vee (c_i \mathbf{U} d_i))$, displaying execution time in seconds/#acceptance sets.

	ψ_1	ψ_2	ψ_3	ψ_4	ψ_5	$\ldots$	ψ_8
Rabinizer 3.1 [EKS16]	<2/2	<2/7	<2/19	–	–		–
ltl2dgra	<2/1	<2/1	<2/1	<2/1	<2/1		<2/1

Table 3. Effect of break-point elimination for **ltl2ldba** on safety formulae $s(n,m) = \bigwedge_{i=1}^{n} \mathbf{G}(a_i \vee \mathbf{X}^m b_i)$ and for **ltl2ldgba** on liveness formulae $l(n,m) = \bigwedge_{i=1}^{n} \mathbf{GF}(a_i \wedge \mathbf{X}^m b_i)$, displaying #states (#Büchi conditions)

	$s(1,3)$	$s(2,3)$	$s(3,3)$	$s(4,3)$	$s(1,4)$	$s(2,4)$	$s(3,4)$	$s(4,4)$
[SEJK16]	20 (1)	400 (2)	$8 \cdot 10^3$(3)	$16 \cdot 10^4$(4)	48 (1)	2304 (2)	110592 (3)	–
ltl2ldba	8 (1)	64 (1)	512 (1)	4096 (1)	16 (1)	256 (1)	4096 (1)	65536 (1)
	$l(1,1)$	$l(2,1)$	$l(3,1)$	$l(4,1)$	$l(1,4)$	$l(2,4)$	$l(3,4)$	$l(4,4)$
[SEJK16]	3 (1)	9 (2)	27 (3)	81 (4)	10 (1)	100 (2)	10^3 (3)	10^4 (4)
ltl2ldgba	3 (1)	5 (2)	9 (3)	17 (4)	3 (1)	5 (2)	9 (3)	17 (4)

Table 4. Effect of non-determinism of the initial component for **ltl2ldba** on formulae $f(i) = \mathbf{F}(a \wedge \mathbf{X}^i \mathbf{G} b)$, displaying #states (#Büchi conditions)

	$f(1)$	$f(2)$	$f(3)$	$f(4)$	$f(5)$	$f(6)$
[SEJK16]	4 (1)	6 (1)	10 (1)	18 (1)	34 (1)	66 (1)
ltl2ldba	2 (1)	3 (1)	4 (1)	5 (1)	6 (1)	7 (1)

Table 5. Comparison of the average performance with the previous version of Rabinizer. The statistics are taken over a set of 200 standard formulae [KMS18] used, e.g., in [BKS13,EKS16], run in a batch mode for both tools to eliminate the effect of the JVM start-up overhead.

Tool	Avg # states	Avg # acc. sets	Avg runtime
Rabinizer 3.1 [EKS16]	6.3	6.7	0.23
ltl2dgra	6.2	4.4	0.12

Implementation. The main performance bottleneck of the older implementations is that explicit data structures for the transition system are not efficient for larger alphabets. To this end, Rabinizer 3.1 provided symbolic (BDD) representation of states and edge labels. On the top, Rabinizer 4 represents the transition function symbolically, too.

Besides, there are further engineering improvements on issues such as storing the acceptance condition only as a local edge labelling, caching, data-structure overheads, SCC-based divide-and-conquer constructions, or the introduction of parallelization for batch inputs.

Average Performance Evaluation. We have already illustrated the improvements on several hand-crafted families of formulae. In Tables 1 and 2 we have even seen the respective running-time speed-ups. As the basis for the overall evaluation of the improvements, we use some established datasets from literature, see [KMS18], altogether two hundred formulae. The results in Table 5 indicate that the performance improved also on average among the more realistic formulae.

4 Conclusion

We have presented Rabinizer 4, a tool set to translate LTL to various deterministic automata and to use them in probabilistic model checking and in synthesis. The tool set extends the previous functionality of Rabinizer, improves on previous translations, and also gives the very first implementations of frequency LTL translation as well as model checking. Finally, the tool set is also more user-friendly due to richer input syntax, its connection to PRISM and PG Solver, and the on-line version with direct visualization, which can be found at http://rabinizer.model.in.tum.de.

References

[BBD+15] Babiak, T., et al.: The hanoi omega-automata format. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 479–486. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_31

[BBDL+13] Babiak, T., Badie, T., Duret-Lutz, A., Křetínský, M., Strejček, J.: Compositional approach to suspension and other improvements to LTL translation. In: Bartocci, E., Ramakrishnan, C.R. (eds.) SPIN 2013. LNCS, vol. 7976, pp. 81–98. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39176-7_6

[BDK+17] Blahoudek, F., Duret-Lutz, A., Klokočka, M., Křetínský, M., Strejček, J.: Seminator: a tool for semi-determinization of omega-automata. In: LPAR, pp. 356–367 (2017)

[BDL12] Bollig, B., Decker, N., Leucker, M.: Frequency linear-time temporal logic. In: TASE, pp. 85–92 (2012)

[BKŘS12] Babiak, T., Křetínský, M., Řehák, V., Strejček, J.: LTL to Büchi automata translation: fast and more deterministic. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 95–109. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28756-5_8

[BKS13] Blahoudek, F., Křetínský, M., Strejček, J.: Comparison of LTL to deterministic rabin automata translators. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR 2013. LNCS, vol. 8312, pp. 164–172. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45221-5_12

[BMM14] Bouyer, P., Markey, N., Matteplackel, R.M.: Averaging in LTL. In: Baldan, P., Gorla, D. (eds.) CONCUR 2014. LNCS, vol. 8704, pp. 266–280. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44584-6_19

[CGK13] Chatterjee, K., Gaiser, A., Křetínský, J.: Automata with generalized rabin pairs for probabilistic model checking and LTL synthesis. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 559–575. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_37

[CKK15] Chatterjee, K., Komárková, Z., Křetínský, J.: Unifying two views on multiple mean-payoff objectives in Markov decision processes. In: LICS, pp. 244–256 (2015)

[CY88] Courcoubetis, C., Yannakakis, M.: Verifying temporal properties of finite-state probabilistic programs. In: FOCS, pp. 338–345 (1988)

[DLLF+16] Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, É., Xu, L.: Spot 2.0 — a framework for LTL and ω-automata manipulation. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 122–129. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46520-3_8

[EH00] Etessami, K., Holzmann, G.J.: Optimizing Büchi automata. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 153–168. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44618-4_13

[EK14] Esparza, J., Křetínský, J.: From LTL to deterministic automata: a safraless compositional approach. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 192–208. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_13

[EKRS17] Esparza, J., Křetínský, J., Raskin, J.-F., Sickert, S.: From LTL and limit-deterministic Büchi automata to deterministic parity automata. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 426–442. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_25

[EKS16] Esparza, J., Kretínský, J., Sickert, S.: From LTL to deterministic automata - a safraless compositional approach. Formal Methods Syst. Des. **49**(3), 219–271 (2016)

[FKK15] Forejt, V., Krčál, J., Křetínský, J.: Controller synthesis for MDPs and frequency LTL\GU. In: LPAR, pp. 162–177 (2015)

[FL09] Friedmann, O., Lange, M.: Solving parity games in practice. In: Liu, Z., Ravn, A.P. (eds.) ATVA 2009. LNCS, vol. 5799, pp. 182–196. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04761-9_15

[FL15] Fisman, D., Lustig, Y.: A modular approach for büchi determinization. In: CONCUR, pp. 368–382 (2015)

[GKE12] Gaiser, A., Křetínský, J., Esparza, J.: Rabinizer: small deterministic automata for LTL(**F**,**G**). In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, pp. 72–76. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33386-6_7

[GL02] Giannakopoulou, D., Lerda, F.: From states to transitions: improving translation of LTL formulae to Büchi automata. In: Peled, D.A., Vardi, M.Y. (eds.) FORTE 2002. LNCS, vol. 2529, pp. 308–326. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36135-9_20

[GO01] Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 53–65. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44585-4_6. http://www.lsv.ens-cachan.fr/ gastin/ltl2ba/

[HLS+15] Hahn, E.M., Li, G., Schewe, S., Turrini, A., Zhang, L.: Lazy probabilistic model checking without determinisation. In: CONCUR. LIPIcs, vol. 42, pp. 354–367 (2015)

[JBB+17] Jacobs, S., Basset, N., Bloem, R., Brenguier, R., Colange, M., Faymonville, P., Finkbeiner, B., Khalimov, A., Klein, F., Michaud, T., Pérez, G.A., Raskin, J.-F., Sankur, O., Tentrup, L.: The 4th reactive synthesis competition (SYNTCOMP 2017): benchmarks, participants & results. CoRR, abs/1711.11439 (2017)

[KE12] Křetínský, J., Esparza, J.: Deterministic automata for the (F,G)-fragment of LTL. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 7–22. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_7

[KK14] Komárková, Z., Křetínský, J.: Rabinizer 3: safraless translation of LTL to small deterministic automata. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 235–241. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11936-6_17

[Kle] Klein, J.: ltl2dstar - LTL to deterministic Streett and Rabin automata. http://www.ltl2dstar.de/

[KLG13] Křetínský, J., Garza, R.L.: Rabinizer 2: Small Deterministic Automata for LTL\GU. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 446–450. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-02444-8_32

[KMS18] Křetínský, J., Meggendorfer, T., Sickert, S.: LTL store: repository of LTL formulae from literature and case studies. CoRR, abs/1807.03296 (2018)

[KMWW17] Křetínský, J., Meggendorfer, T., Waldmann, C., Weininger, M.: Index appearance record for transforming rabin automata into parity automata. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 443–460. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_26

[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

[KV15] Kini, D., Viswanathan, M.: Limit deterministic and probabilistic automata for LTL\GU. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 628–642. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_57

[KV17] Kini, D., Viswanathan, M.: Optimal translation of LTL to limit deterministic automata. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 113–129. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54580-5_7

[ML16] Meyer, P.J., Luttenberger, M.: Solving mean-payoff games on the GPU. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 262–267. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46520-3_17

[MS17] Müller, D., Sickert, S.: LTL to deterministic Emerson-Lei automata. In: GandALF, pp. 180–194 (2017)

[Pit06] Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: LICS, pp. 255–264 (2006)

[Pnu77] Pnueli, A.: The temporal logic of programs. In: FOCS, pp. 46–57 (1977)

[PP06] Piterman, N., Pnueli, A.: Faster solutions of Rabin and Streett games. In: LICS, pp. 275–284 (2006)

[Saf88] Safra, S.: On the complexity of omega-automata. In: FOCS, pp. 319–327 (1988)

[SB00] Somenzi, F., Bloem, R.: Efficient Büchi automata from LTL formulae. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 248–263. Springer, Heidelberg (2000). https://doi.org/10.1007/10722167_21

[Sch09] Schewe, S.: Tighter bounds for the determinisation of Büchi automata. In: de Alfaro, L. (ed.) FoSSaCS 2009. LNCS, vol. 5504, pp. 167–181. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00596-1_13

[SEJK16] Sickert, S., Esparza, J., Jaax, S., Křetínský, J.: Limit-deterministic Büchi automata for linear temporal logic. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9780, pp. 312–332. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41540-6_17

[SK16] Sickert, S., Křetínský, J.: MoChiBA: probabilistic LTL model checking using limit-deterministic Büchi automata. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 130–137. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46520-3_9

[TD14] Tian, C., Duan, Z.: Buchi determinization made tighter. Technical report abs/1404.1436, arXiv.org (2014)

[VW86] Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification (preliminary report). In: LICS, pp. 332–344 (1986)

D Owl: A Library for ω-Words, Automata, and LTL. ATVA 2018

This section has been published as **peer-reviewed conference paper**.

Jan Křetínský, Tobias Meggendorfer and Salomon Sickert. 'Owl: A Library for ω-Words, Automata, and LTL'. in: *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings.* Ed. by Shuvendu K. Lahiri and Chao Wang. Vol. 11138. Lecture Notes in Computer Science. Springer, 2018, pp. 543–550. DOI: `10.1007/978-3-030-01090-4\_34`. URL: `https://doi.org/10.1007/978-3-030-01090-4%5C_34`

Synopsis We present the library Owl (**O**mega-**W**ords, automata, and **L**TL). It forms a feature-rich framework for constructions involving ω-automata and LTL. Among others, it hosts the complete Rabinizer 4 tool suite [Kře+18] (see Paper C). Moreover, many user-friendly features and utility methods are provided to ease development of further translations and use in, for example, student projects and teaching. Implementing a new construction in Owl is possible with a few hundred lines of code, including integration with the command line interface and the provided automatic test suite. Nevertheless, the library is heavily optimized and implementations based on Owl are competitive with existing tools in this area. For example, Rabinizer 4 constructions' are competitive with Spot [Dur+16], the other de-factor standard tool for LTL-to-automaton translations. Strix [MSL18], a reactive synthesis tool partly based on Owl, repeatedly won the synthesis competition SYNTCOMP [Jac+19].

Contributions of the thesis author Discussion and revision of the entire manuscript. Significant contributions to the design and implementation of Owl, including a full implementation of BDDs in Java, `JBDD` [Meg17]. At the time of writing, the author still is actively contributing to Owl.

Owl: A Library for ω-Words, Automata, and LTL

Jan Křetínský(✉), Tobias Meggendorfer, and Salomon Sickert

Technical University of Munich, Munich, Germany
{jan.kretinsky,tobias.meggendorfer,sickert}@in.tum.de

Abstract. We present the library `Owl` (**O**mega-**W**ords, automata, and **L**TL) for ω-automata and linear temporal logic. It forms a backbone of several translations from LTL to automata and related tools by different authors. We describe the functionality of the library and the recent experience, which has already shown the library is apt for easy prototyping of new tools in this area.

1 An Owl is Born: Introduction

ω-automata are finite automata over infinite words. As opposed to finite automata over finite words, there is not a single acceptance condition, but a wide variety of possibilities, each being more appropriate for certain applications. To give a few examples, non-deterministic Büchi automata are the most used kind, useful in many contexts, including the modelling and analysis of reactive systems, where both the system and the property of interest, say in linear temporal logic (LTL) [33], are transformed into these automata. In contrast, the classical approach for synthesis of reactive systems [34] prefers deterministic parity automata. Further, while the textbook approach to probabilistic LTL model checking suggests to translate LTL formulas to deterministic Rabin automata [4], recent approaches show that deterministic generalized Rabin automata or limit-deterministic automata are more preferable [6,37,38]. Consequently, a zoo of automata arises, both due to theoretical limitations of certain kinds as well as practical efficiency. While the theoretical complexity of the transformations between the automata and of translations from LTL to automata is long settled, the research on practically more efficient approaches is flourishing, both for non-deterministic [3,7–9,15–17,39] and more recently deterministic [2,11–14,19,20,23,37] automata. Notably, while these constructions are based on diverse ideas, their implementation requires almost the same infrastructure.

Tools in this area have very different purposes, ranging from tools for one specific task, e.g. translating LTL into a particular type of automaton, e.g. [3,16,20–22],

This research was supported in part by the Czech Science Foundation grant No. P202/12/G061 and the the German Research Foundation (DFG) projects "Statistical Unbounded Verification" (383882557) and "Verified Model Checkers" (317422601).

S. K. Lahiri and C. Wang (Eds.): ATVA 2018, LNCS 11138, pp. 543–550, 2018.
https://doi.org/10.1007/978-3-030-01090-4_34

to educational GUI tools demonstrating the constructions, e.g. `JFLAP` [35], to tools implementing a comprehensive collection of algorithms from literature, e.g. `GOAL` [41] and `Spot` [9]. We contribute to this spectrum the library `Owl`, which enables easy and fast development of transformation/translation tools, yet yielding efficient implementations.

Owl is a full-fledged library for manipulating ω-automata and LTL. One of the main characteristics is that it links the functionality for automata and logic in a very tight and explicit way, providing additional support for "semantic" translations of LTL to automata. These are translations where states are described using structures over logical formulas, as we know it from the classical, e.g. the tableaux-based, tradition. This tradition was disrupted for deterministic automata due to Safra's construction [36], where the meaning of a state (the language it recognizes) cannot be easily described in terms of the meaning of the corresponding formulas. The "semantic" tradition has been restored recently in the works on deterministic automata cited above and `Owl` provides specialised operations (see below) on LTL that are the building blocks for obtaining such a translation.

Apart from this characteristics, our library has several other user-friendly traits and distinguishing features. For instance, it is built according to the on-the-fly philosophy, it is written in Java (with no memory management issues left for the user, being more accessible to students), extensive CLI support for quick and easy prototyping, and a testing framework checking correctness of translations written with the library.

In this tool paper, we briefly describe the functionality of the library and then provide a series of actual use cases (not only by the authors), demonstrating the usability and particular advantages of this library.

2 The Anatomy of the Owl: Functionality

`Owl` (**O**mega-**W**ords, automata, and **L**TL) arose from the needs when implementing `Rabinizer` 3.1 [13,22] and `ltl2ldba` [37]. When developing such translations a lot of infrastructure is necessary, e.g., LTL parsing and representation, while the actual construction is only a small fraction of the written code. Thus, we implemented commonly needed functionality in a reusable Java library for LTL and ω-automata and extended it with numerous features to provide a flexible infrastructure for rapid and seamless development of algorithms in these domains.

2.1 Data Structures and Algorithms

The majority of data structures and algorithms concerns LTL and automata.

LTL. The library provides an LTL parser, a simplifier with state-of-the-art rewrite rules, classification into syntactic fragments and transformation into normal forms. Additionally, a parser for the synthesis specification format TLSF [18] is available and includes a conversion to LTL.

Further, the LTL support comes with efficient rewriting according to the LTL expansion laws, e.g. [4]. This enables the decomposition of temporal formulas into directly checkable assertions on the current position and on the immediate temporal successor, e.g. $a\mathbf{U}b \equiv b \vee (a \wedge \mathbf{X}(a\mathbf{U}b))$. As such, they are a core component of both classic, e.g. tableaux-based, as well as recent semantic translations.

Automata. The library provides support for deterministic and non-deterministic ω-automata with both classic acceptance conditions, e.g., Büchi, coBüchi, Rabin and parity, as well as, e.g., like generalized Rabin [28] or Emerson-Lei acceptance [10]. Internally, acceptance is represented as transition-based acceptance and a conversion to and from state-based acceptance for interfacing with external tools is present.

Automata can either be stored and modified explicitly, meaning the whole state-space and transitions are kept in memory, or defined implicitly by specifying initial states and a method for successor computation. The latter approach has two main advantages: First, new constructions can be implemented with little effort, transferring the definition of the successor relation into code. For example, see [24] for a ca. 60 lines Java implementation of Safra's determinization procedure. Second, automata can be conveniently traversed on the fly without storing the transition system, allowing operations on huge or potentially even infinite transition structures.

For automata, classic algorithms such as decomposition into strongly connected components (SCC) and lasso-based emptiness checks are included. Furthermore, constructions such as union, intersection and degeneralization are present. In addition, modifications of the transition structure and the acceptance conditions are supported, e.g., removal of non-accepting or unreachable parts of the state space, completing the transition relation, and simplifications of the acceptance condition. Acceptance sets are stored as edge labels for efficient rewriting, supporting arbitrarily sized acceptances, compared to, e.g., `Spot` [9], which at the time of writing supports only an at compile-time determined bounded number of sets.

2.2 Interfacing

There are two ways to interact with `Owl`: On the one hand, there is a command-line interface with text-based formats, e.g., (`Spot`-compatible) LTL, TLSF [18], and the Hanoi ω-automaton format (HOA) [1]. This approach is completely agnostic of the implementation, but always requires a complete construction, which is prohibitively expensive for huge outputs where only a small fraction might be needed. On the other hand, there is a Java and a (specialized) C++ API offered by `Owl`, which allows fine-grained access and exposes the on-the-fly nature to external code.

Command-line Interface. Major functionality of the library is available via a pipe-style CLI, which makes it easy to specify the sequence of procedures (input parsing, translations, conversions, statistics and serialization) to be performed. For example, `owl ltl --- simplify-ltl ---`

`ltl2dpa --- hoa` reads LTL formulas from `stdin` line-by-line, simplifies them using the default simplifier, translates them to DPAs and writes them to `stdout` in the HOA format. This can be extended to advanced pipelines, e.g., `owl -I "in.ltl" --- ltl --- ltl2dgra --- aut-stat "DGRA:%s" --- dgra2dra --- aut-stat "DRA:%s" --- null`.

This pipeline reads LTL formulas from the file `in.ltl`, translates them to DGRAs and DRAs, while outputting the respective sizes of the automata, and finally discards the actual output, saving the time needed for serialization.

Moreover, we support several sources and sinks for data. While one can simply process data from files and the command line, we also added a server mode to reduce the JVM start-up cost, where I/O is bound to a socket. Further details on the CLI together with an in-depth example can be found on [24].

Java and C++ API. Java and Java-like (e.g., Scala) applications can import `Owl` and have fine-grained control. For C++ tools, there exists a specialized interface to access core functionality of the library. Among other things, this enables C++ code to iteratively explore automata state by state instead of forcing a complete construction. This iterative exploration is a core component of the state-of-the-art synthesis tool `Strix` [31] and is crucial for its performance.

2.3 Development Infrastructure and Scalable Architecture

Testing. Small changes to a translation can easily introduce bugs. Thus a test suite is included, which provides several input sets and cross-checks each translation, developed with `Owl`, on hundreds of formulas [25] using `ltlcross` [9]. Apart from detecting bugs, the test suite offers further conveniences, e.g., it automatically generates an image of an erroneous automaton together with an erroneous run. Moreover, various statistics of the generated automata are displayed, usable for performance testing. Lastly, integration of a newly developed translation can be achieved by a few lines of JSON, see [24] for an example.

BDDs. Both the LTL part and the automata part of the library use binary decision diagrams (BDD) for some aspects of their functionality, e.g., for a compact representation edge sets and (propositional) equivalence checks of formulas. We implemented our own pure Java BDD library `JBDD` [30], to (i) achieve portability, not requiring users to compile, e.g., `CUDD`, and (ii) provide an efficient and tuned implementation for all used BDD operations, e.g. substitution of variables, called `compose`. Particularly, `compose` is fundamental for a symbolic implementation of the semantic constructions and greatly improves their runtime compared to the explicit variants.

3 The Owl in the Wild: Use Cases

`Owl` has been successfully used for several published tools and student projects, demonstrating versatility and usability even for less experienced users. To name a few, the following published tools (in alphabetical order) using `Owl` are available:

Delag [32] translates LTL into deterministic Emerson-Lei automata.Reusing other translations based on Owl, see Rabinizer [26], it adds specialized constructions for fragments of LTL, exploiting a succinct encoding coupled to the Emerson-Lei acceptance condition. The current distribution of Owl includes the latest version of it.

MoChiBa [38] is an extension of PRISM [29] and uses limit-deterministic automata for quantitative model checking of Markov decision processes [37]. Due to a tight integration with Owl, additional information on the automata can be accessed, optimizing the construction.

Rabinizer [26] is a collection of tools translating LTL to various types of deterministic automata. It uses a fully BDD-based successor computation of Owl, improving performance over the previous versions. The current distribution of Owl includes the latest version of Rabinizer (4.0).

Strix [31] synthesises controllers (either Mealy machines or AIGER circuits) from LTL specifications via parity games. Constructing the underlying automata and solving the parity games take an incremental approach and make use of the on-the-fly implementations.

The list of student projects includes[1]

– a re-implementation of Seminator [5],
– a specialized translation of the $(\mathbf{F}, \mathbf{G}, \mathbf{X})$-fragment of LTL to deterministic parity automata, and
– reactive synthesis exploiting the Owl-supported semantic labelling of the automata produced by Rabinizer through learning approaches.

Furthermore, rLTL (robust LTL) [40] can be easily transformed into LTL using Owl[2]. Finally, to illustrate the ease with which new translations can be written, we implemented the notoriously complicated and hard-to-implement [27] Safra's determinization procedure [36], which can be found on [24]. A detailed analysis of the lines of code needed to implement the mentioned translations and the percentage of library that is used can be found on [24].

4 This is Not the End: Conclusion

We have presented the library Owl, which provides infrastructure for easy development of efficient prototypes in the area of LTL and automata. It has already demonstrated its re-usability in several projects, also without the presence of the library authors. For instance, our experience with Master students has demonstrated that a tool for a complex translation, such as [5], can be easily implemented using roughly 400 lines of code, achieving performance comparable to the original dedicated tool. One simply defines the mathematical type of the state space, the initial state, the successor function with the acceptance marking, whereas the rest is taken care of by the library. The library can be found at https://owl.model.in.tum.de, including code, documentation, references and an online demo. We greatly appreciate comments and suggestions.

[1] Authored by Florian Barta, Matthias Franze, and Sebastian Fiss, respectively.
[2] Originally implemented by Daniel Neider.

References

1. Babiak, T., et al.: The Hanoi Omega-automata format. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 479–486. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_31
2. Babiak, T., Blahoudek, F., Křetínský, M., Strejček, J.: Effective translation of LTL to deterministic Rabin automata: beyond the (F,G)-fragment. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 24–39. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-02444-8_4
3. Babiak, T., Křetínský, M., Řehák, V., Strejček, J.: LTL to Büchi automata translation: fast and more deterministic. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 95–109. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28756-5_8
4. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press, Cambridge (2008)
5. Blahoudek, F., Duret-Lutz, A., Klokočka, M., Křetínský, M., Strejček, J.: Seminator: a tool for semi-determinization of omega-automata. In: LPAR (2017)
6. Chatterjee, K., Gaiser, A., Křetínský, J.: Automata with generalized rabin pairs for probabilistic model checking and LTL synthesis. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 559–575. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_37
7. Couvreur, J.-M.: On-the-fly verification of linear temporal logic. In: Wing, J.M., Woodcock, J., Davies, J. (eds.) FM 1999. LNCS, vol. 1708, pp. 253–271. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48119-2_16
8. Daniele, M., Giunchiglia, F., Vardi, M.Y.: Improved automata generation for linear temporal logic. In: Halbwachs, N., Peled, D. (eds.) CAV 1999. LNCS, vol. 1633, pp. 249–260. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48683-6_23
9. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, É., Xu, L.: Spot 2.0 — a framework for LTL and ω-automata manipulation. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 122–129. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46520-3_8
10. Emerson, E.A., Lei, C.: Modalities for model checking: branching time strikes back. In: POPL (1985)
11. Esparza, J., Křetínský, J.: From LTL to deterministic automata: a safraless compositional approach. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 192–208. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_13
12. Esparza, J., Křetínský, J., Raskin, J.-F., Sickert, S.: From LTL and limit-deterministic Büchi automata to deterministic parity automata. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 426–442. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_25
13. Esparza, J., Křetínský, J., Sickert, S.: From LTL to deterministic automata - a safraless compositional approach. Form. Methods Syst. Des. (2016)
14. Esparza, J., Křetínský, J., Sickert, S.: One theorem to rule them all: a unified translation of LTL into ω-automata. In: LICS (2018)
15. Etessami, K., Holzmann, G.J.: Optimizing Büchi automata. In: Palamidessi, C. (ed.) CONCUR 2000. LNCS, vol. 1877, pp. 153–168. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44618-4_13
16. Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. In: CAV (2001). Tool accessible at http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/
17. Giannakopoulou, D., Lerda, F.: From states to transitions: improving translation of LTL formulae to Büchi automata. In: Peled, D.A., Vardi, M.Y. (eds.) FORTE

2002. LNCS, vol. 2529, pp. 308–326. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36135-9_20
18. Jacobs, S., Klein, F., Schirmer, S.: A high-level LTL synthesis format: TLSF v1.1. In: Fifth Workshop on Synthesis (SYNT@CAV) (2016)
19. Kini, D., Viswanathan, M.: Limit deterministic and probabilistic automata for LTL \ GU. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 628–642. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_57
20. Kini, D., Viswanathan, M.: Optimal translation of LTL to limit deterministic automata. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 113–129. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54580-5_7
21. Klein, J.: ltl2dstar - LTL to deterministic Streett and Rabin automata. http://www.ltl2dstar.de/
22. Komárková, Z., Křetínský, J.: Rabinizer 3: safraless translation of LTL to small deterministic automata. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 235–241. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11936-6_17
23. Křetínský, J., Esparza, J.: Deterministic automata for the (F,G)-fragment of LTL. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 7–22. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_7
24. Křetínský, J., Meggendorfer, T., Sickert, S.: Owl: a library for ω-words, automata, and LTL. https://owl.model.in.tum.de. Accessed July 2018
25. Křetínský, J., Meggendorfer, T., Sickert, S.: LTL store: repository of LTL formulae from literature and case studies. CoRR, abs/1807.03296 (2018)
26. Křetínský, J., Meggendorfer, T., Sickert, S., Ziegler, C.: Rabinizer 4: from LTL to your favourite deterministic automaton. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 567–577. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_30
27. Kupferman, O.: Recent challenges and ideas in temporal synthesis. In: Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S., Turán, G. (eds.) SOFSEM 2012. LNCS, vol. 7147, pp. 88–98. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27660-6_8
28. Křetínský, J., Esparza, J.: Deterministic automata for the (F,G)-fragment of LTL. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 7–22. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_7
29. 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
30. Meggendorfer, T.: JBDD: a java BDD library. https://github.com/incaseoftrouble/jbdd. Accessed July 2018
31. Meyer, P.J., Sickert, S., Luttenberger, M.: Strix: explicit reactive synthesis strikes back!. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 578–586. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_31
32. Müller, D., Sickert, S.: LTL to deterministic Emerson-Lei automata. In: Gandalf (2017)
33. Pnueli, A.: The temporal logic of programs. In: FOCS (1977)
34. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: POPL (1989)
35. Rodger, S.H., Qin, H., Su, J.: Changes to JFLAP to increase its use in courses. In: SIGCSE (2011)

36. Safra, S.: On the complexity of omega-automata. In: FOCS (1988)
37. Sickert, S., Esparza, J., Jaax, S., Křetínský, J.: Limit-deterministic Büchi automata for linear temporal logic. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9780, pp. 312–332. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41540-6_17
38. Sickert, S., Křetínský, J.: MoChiBA: probabilistic LTL model checking using limit-deterministic Büchi automata. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 130–137. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46520-3_9
39. Somenzi, F., Bloem, R.: Efficient Büchi automata from LTL formulae. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 248–263. Springer, Heidelberg (2000). https://doi.org/10.1007/10722167_21
40. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 92–106. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15297-9_9
41. Tsai, M.-H., Tsay, Y.-K., Hwang, Y.-S.: GOAL for games, omega-automata, and logics. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 883–889. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_62

E Of Cores: A Partial-Exploration Framework for Markov Decision Processes. CONCUR 2019

This section has been published as **peer-reviewed conference paper**.

> Jan Křetínský and Tobias Meggendorfer. 'Of Cores: A Partial-Exploration Framework for Markov Decision Processes'. In: *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*. Ed. by Wan Fokkink and Rob van Glabbeek. Vol. 140. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 5:1–5:17. DOI: `10.4230/LIPIcs.CONCUR.2019.5`. URL: `https://doi.org/10.4230/LIPIcs.CONCUR.2019.5`

Synopsis Inspired by previous approaches [Brá+14; Ash+17] (see Paper A), we introduce a new framework for partial exploration, investigating fundamental limitations and possible extensions of this approach. Previous works explore the system relative to a given objective, i.e. gather enough information such that this objective can be answered up to a given precision. Instead, we identify the essential part of the system, called *core*, sufficient to answer *any* such query up to the given precision. Asides from immediate advantages such as re-usability, this property-agnostic analysis allows for further understanding of systems themselves. In particular, we introduced the concept of *step-bounded cores* and *stability*, which are particularly useful for understanding systems and early detection of design faults.

At the time of writing, a significantly extended journal version of this paper is under submission.

Contributions of the thesis author Composition of the manuscript except abstract and introduction. Discussion and revision of the entire manuscript. Sole contribution of all results and proofs presented in the paper, with the exception of the proof idea for Theorem 6. Sole design and implementation of the presented tool.

Of Cores: A Partial-Exploration Framework for Markov Decision Processes

Jan Křetínský
Technical University of Munich, Germany
jan.kretinsky@in.tum.de

Tobias Meggendorfer
Technical University of Munich, Germany
tobias.meggendorfer@in.tum.de

Abstract

We introduce a framework for approximate analysis of Markov decision processes (MDP) with bounded-, unbounded-, and infinite-horizon properties. The main idea is to identify a "core" of an MDP, i.e., a subsystem where we provably remain with high probability, and to avoid computation on the less relevant rest of the state space. Although we identify the core using simulations and statistical techniques, it allows for rigorous error bounds in the analysis. Consequently, we obtain efficient analysis algorithms based on partial exploration for various settings, including the challenging case of strongly connected systems.

2012 ACM Subject Classification Theory of computation → Verification by model checking; Theory of computation → Random walks and Markov chains

Keywords and phrases Markov Decision Processes, Reachability, Approximation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.5

Related Version A full version of the paper is available at `https://arxiv.org/abs/1906.06931`.

Funding This work has been partially supported by the Czech Science Foundation grant No. 18-11193S and the German Research Foundation (DFG) project KR 4890/2 "Statistical Unbounded Verification".

1 Introduction

Markov decision processes (MDP) are a well established formalism for modelling, analysis and optimization of probabilistic systems with non-determinism, with a large range of application domains [18]. Classical objectives such as reachability of a given state or the long-run average reward (mean payoff) can be solved by a variety of approaches. In theory, the most suitable approach is linear programming as it provides exact answers (rational numbers with no representation imprecision) in polynomial time. However, in practice for systems with more than a few thousand states, linear programming is not very usable, see, e.g., [2]. As an alternative, one can apply dynamic programming, typically value iteration (VI) [4], the default method in the probabilistic model checkers PRISM [13] and Storm [9].

Despite better practical scalability of VI, systems with more than a few million states still remain out of reach of the analysis not only because of time-outs, but now also memory-outs, see, e.g., [6]. There are various heuristics designed to deal with so large state spaces, including abstractions, e.g., [8, 11], or a dual approach based on restricting the analysis to a part of the state space. Examples of the latter approach are asynchronous VI in probabilistic planning, e.g., [17], or projections in approximate dynamic programming, e.g., [5]. In both, only a certain subset of states is considered for analysis, leading to speed ups in orders of magnitude. These are best-effort solutions, which can only guarantee convergence to the true result in the limit, with no error bounds at any finite time. Surprisingly, this was the case with the standard VI as well until recently, when an error bound (and thus a stopping criterion) was given independently in [10, 6]. The error bound follows from the under- and (newly obtained) over-approximations converging to the true value. This resulted not only in error bounds on VI, but opened the door to error bounds for other techniques, including those where even convergence is not guaranteed. For instance, while VI iteratively approximates the value of all states, the above-mentioned *asynchronous VI* evaluates states at different paces. Thus convergence is often unclear and even the rate of convergence is unknown and very hard to analyze. However, it is not too hard to extend the error bound technique for VI to asynchronous VI. A prime example is the modification of BRTDP [17] to reachability [6] with error bounds. These ideas are further developed for, e.g., settings with long-run average reward [2] or continuous time [1].

While these solutions are efficient, they are ad-hoc, sharing the idea of *simulation / learning-based partial exploration* of the system, but not the correctness proof. In this paper, we build the foundations for designing such frameworks and provide a new perspective on these approaches, leading to algorithms for settings where previous ideas cannot apply.

The previous algorithms use (i) simulations to explore the state space and (ii) planning heuristics and machine learning to analyze the experience and to bias further simulations to areas that seem more relevant for the analysis of the given property (e.g., reaching a state s_{42}), where (iii) the exact VI computation takes place and yields results with a guaranteed error bound. In contrast, this paper identifies a general concept of a "*core*" of the MDP, independently of the particular objective (which states to reach) and, to a certain extent, even of the type of property (reachability, mean payoff, linear temporal logic formulae, etc.). This core intuitively consists of states that are important for the analysis of the MDP, whereas the remaining parts of the state space affect the result only negligibly. To this end, the defining property of a core is that the system stays within the core with high probability.

There are several advantages of cores, compared to the tailored techniques. Since the core is agnostic of any particular property, it can be *re-used* for multiple queries. Thus, the repetitive effort spent by the simulations and heuristics to explore the relevant parts of the state space by the previous algorithms can be saved. Furthermore, identifying the core can serve to better *understand* the typical behaviour of the system. Indeed, the core is typically a lot smaller than the system (and thus more amenable to understand) and only contains the more likely behaviours, even for real-world models as shown in the experimental evaluation. Finally, the general concept of core provides a *unified* argument for the correctness of the previous algorithms since – implicitly – they gradually construct a core. This abstract view thus allows for easier development of further partial-exploration techniques within this framework.

More importantly, making the notion of core explicit naturally leads us to identify a new standpoint and approach for the more complicated case of strongly connected systems, where the previous algorithms as well as cores cannot help. In technical terms, minimal cores are closed under end components. Consequently, the minimal core for a strongly connected system is the whole system. And indeed, it is impossible to give guarantees on infinite-horizon behaviour whenever a single state is ignored. In order to provide *some* feasible analysis for this case, we introduce the n-step core. It is defined by the system staying there with high probability for *some* time. However, instead of n-step analysis, we suggest to observe the "stability" of the core, i.e. the tendency of the probability to leave this core if longer and longer runs are considered. We shall argue that this yields (i) rigorous bounds for N-step analysis for $N \gg n$ more efficiently than a direct N-step analysis, and (ii) finer information on the "long run" behaviour (for different lengths) than the summary for the infinite run, which, n.b., never occurs in reality. This opens the door towards a rigorous analysis of "typical" behaviour of the system, with many possible applications in the design and interpretation of complex systems.

Our contribution can be summarized as follows:

- We introduce the notion of core, study its basic properties, in its light re-interpret previous results in a unified way, and discuss its advantages.
- We stipulate a new view on long-run properties as rather corresponding to long runs than an infinite one. Then a modified version of cores allows for an efficient analysis of strongly connected systems, where other partial-exploration techniques necessarily fail.
- We show how these modified cores can aid in design and interpretation of systems.
- We provide efficient algorithms for computing both types of cores and evaluate them on several examples.

2 Preliminaries

In this section, we recall basics of probabilistic systems and set up the notation. We assume familiarity with the central ideas of measure theory. As usual, $\mathbb{N}$ and $\mathbb{R}$ refers to the (positive) natural numbers and real numbers, respectively. For any set S, we use $\overline{S}$ to denote its complement. A *probability distribution* on a finite set X is a mapping $p : X \to [0, 1]$, such that $\sum_{x \in X} p(x) = 1$. Its *support* is denoted by $\operatorname{supp}(p) = \{x \in X \mid p(x) > 0\}$. $\mathcal{D}(X)$ denotes the set of all probability distributions on X. An event happens *almost surely* (a.s.) if it happens with probability 1.

▶ **Definition 1.** *A* Markov chain (MC) *is a tuple* $\mathsf{M} = (S, s_0, \delta)$*, where* S *is a countable set of* states*,* $s_0 \in S$ *is the* initial *state, and* $\delta : S \to \mathcal{D}(S)$ *is a* transition function *that for each state* s *yields a probability distribution over successor states.*

▶ **Definition 2.** *A* Markov decision process (MDP) *is a tuple of the form* $\mathcal{M} = (S, s_0, A, \mathsf{Av}, \Delta)$*, 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*, and* $\Delta : S \times A \to \mathcal{D}(S)$ *is a* transition function *that for each state* s *and (available) action* $a \in \mathsf{Av}(s)$ *yields a probability distribution over successor states. Furthermore, we assume w.l.o.g. that actions are unique for each state, i.e.* $\mathsf{Av}(s) \cap \mathsf{Av}(s') = \emptyset$ *for* $s \neq s'$ *(which can be achieved by replacing* A *with* $S \times A$ *and adapting* Av *and* Δ *appropriately).*

For ease of notation, we overload functions mapping 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 $\delta(s)(s')$ and $\Delta(s, a)(s')$ we write $\delta(s, s')$ and $\Delta(s, a, s')$, respectively.

An *infinite path* ρ in a Markov chain is an infinite sequence $\rho = s_0 s_1 \ldots \in S^\omega$, such that for every $i \in \mathbb{N}$ we have that $\delta(s_i, s_{i+1}) > 0$. A *finite path* (or *history*) $\varrho = s_0 s_1 \ldots s_n \in S^*$ is a finite prefix of an infinite path. Similarly, an *infinite path* in an MDP is some infinite sequence $\rho = s_0 a_0 s_1 a_1 \ldots \in (S \times A)^\omega$, such that for every $i \in \mathbb{N}$, $a_i \in \mathsf{Av}(s_i)$ and $\Delta(s_i, a_i, s_{i+1}) > 0$. *Finite paths* ϱ are defined analogously as elements of $(S \times A)^* \times S$. We use ρ_i and ϱ_i to refer to the i-th state in the given (in)finite path.

A *strategy* on an MDP is a function $\pi : (S \times A)^* \times S \to \mathcal{D}(A)$, which given a finite path $\varrho = s_0 a_0 s_1 a_1 \ldots s_n$ yields a probability distribution $\pi(\varrho) \in \mathcal{D}(\mathsf{Av}(s_n))$ on the actions to be taken next. We denote the set of all strategies of an MDP by Π. Fixing any strategy π induces a Markov chain $\mathcal{M}^\pi = (S^\pi, s_0^\pi, \delta^\pi)$, where the states are given by $S^\pi = (S \times A)^* \times S$ and, for some state $\varrho = s_0 a_0 \ldots s_n \in S^\pi$, the successor distribution is defined as $\delta^\pi(\varrho, \varrho a_{n+1} s_{n+1}) = \pi(\varrho, a_{n+1}) \cdot \Delta(s_n, a_{n+1}, s_{n+1})$.

Any Markov chain M induces a unique measure $\mathbb{P}^{\mathsf{M}}$ over infinite paths [3, p. 758]. Assuming we fixed some MDP $\mathcal{M}$, we use $\mathbb{P}^{\pi}_{s}$ to refer to the probability measure induced by the Markov chain $\mathcal{M}^{\pi}$ with initial state s. Whenever π or s are clear from the context, we may omit them, in particular, $\mathbb{P}^{\pi}$ refers to $\mathbb{P}^{\pi}_{s_0}$. See [18, Sec. 2.1.6] for further details. For a given MDP $\mathcal{M}$ and measurable event E, we use the shorthand $\mathbb{P}^{\max}[E] := \sup_{\pi \in \Pi} \mathbb{P}^{\pi}[E]$ and $\mathbb{P}^{\max}_{s}[E] := \sup_{\pi \in \Pi} \mathbb{P}^{\pi}_{s}[E]$ to refer to the maximal probability of E over all strategies (starting in s). Analogously, $\mathbb{P}^{\min}[E]$ and $\mathbb{P}^{\min}_{s}[E]$ refer to the respective minimal probabilities.

A pair (T, B), where $\emptyset \neq T \subseteq S$ and $\emptyset \neq B \subseteq \bigcup_{s \in T} \mathsf{Av}(s)$, is an *end component* of an MDP $\mathcal{M}$ if (i) for all $s \in T, a \in B \cap \mathsf{Av}(s)$ we have $\operatorname{supp}(\Delta(s, a)) \subseteq T$, and (ii) for all $s, s' \in T$ there is a finite path $\varrho = s a_0 \dots a_n s' \in (T \times B)^* \times T$, i.e. the path stays inside T and only uses actions in B. Intuitively, an end component describes a set of states for which a particular strategy exists such that all possible paths remain inside these states. By abuse of notation, we identify an end component with the respective set of states, e.g., $s \in E = (T, B)$ means $s \in T$. An end component (T, B) is a *maximal end component (MEC)* if there is no other end component (T', B') such that $T \subseteq T'$ and $B \subseteq B'$. The set of MECs of an MDP $\mathcal{M}$ is denoted by $\mathsf{MEC}(\mathcal{M})$ and can be obtained in polynomial time [7].

In the following, we will primarily deal with unbounded and bounded variants of *reachability* queries. Essentially, for a given MDP and set of states, the task is to determine the maximal probability of reaching them, potentially within a certain number of steps. Technically, we are interested in determining $\mathbb{P}^{\max}[\lozenge T]$ and $\mathbb{P}^{\max}[\lozenge^{\leq n} T]$, where T is the set of target states and $\lozenge T$ ($\lozenge^{\leq n} T$) refers to the measurable set of runs that visit T at least once (in the first n steps). The dual operators $\square T$ and $\square^{\leq n} T$ refer to the set of runs which remain inside T forever or for the first n steps, respectively. See [3, Sec. 10.1.1] for further details. Our techniques are easily extendable to other related objectives like *long run average reward* (*mean payoff*) [18], *LTL formulae* or ω-regular objectives [3]. We briefly comment on this in Section 3.3.

We are interested in finding approximate solutions efficiently, i.e. trading precision for speed of computation. In our case, "approximate" means ε-optimal for some given precision $\varepsilon > 0$, i.e. the value we determine has an absolute error of less than ε. For example, given a reachability query $\mathbb{P}^{\max}[\lozenge T]$ and precision ε, we are interested in finding a value v with $|\mathbb{P}^{\max}[\lozenge T] - v| < \varepsilon$.

3 The Core Idea

In this section, we present the novel concept of *cores*, inspired by the approach of [6], where a specific reachability query was answered approximately through heuristic based methods. Omitted proofs can be found in [12, App. A.1]. We first establish a running example to motivate our work and explain the difference to previous approaches.

Consider a flight of an air plane. The controller, e.g. the pilot and the flight computer, can take many decisions to control the plane. The system as a whole can be in many different states. One may be interested in the maximal probability of arriving safely. This intuitively describes how likely it is to arrive, assuming that the pilot acts optimally and the computer is bug-free. Of course, this probability may be less than 100%, since some components may fail even under optimal conditions. See Figure 1 for a simplified MDP modelling this example.

The key observation in [6] is that some extreme situations may be very unlikely and we can simply assume the worst case for them without losing too much precision. This allows us to completely ignore these situations. For example, consider the unlikely event of hazardous bit flips during the flight due to cosmic radiation. This event might eventually lead to a

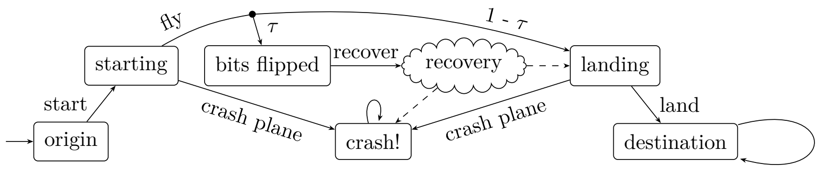


Figure 1 A simplified model of a flight, where $\tau = 10^{-10}$ is the probability of potentially hazardous bit flips occurring during the flight. The "recovery" node represents a complex recovery procedure, comprising many states.

crash or it might have no influence on the evolution of the system at all. Since this event is so unlikely to occur, we can simply assume that it always leads to a crash and still get a very precise result. Consequently, we do not need to explore the corresponding part of the state space (the "recovery" part), saving resources. As shown in the experimental evaluation in Section 5, many real-world models indeed exhibit a similar structure.

In [6], the state space was explored relative to a particular reachability objective, storing upper and lower bounds on each state for the objective in consideration. We make use of the same principle idea, but approach it from a different perspective, agnostic of any objective. We are interested in finding *all* relevant states of the system, i.e. all states which are reasonably likely to be reached. Such a set of states is an *intrinsic property* of the system, and we show that this set is both sufficient and necessary to answer *any* non-trivial reachability query ε-precisely. In particular, once computed, this set can be reused for multiple queries.

3.1 Infinite-Horizon Cores

First, we define the notion of an *ε-core*. Intuitively, an ε-core is a set of states which can only be exited with probability less than ε.

▶ **Definition 3** (Core). *Let $\mathcal{M}$ be an MDP and $\varepsilon > 0$. A set $S_\varepsilon \subseteq S$ is an* ε-core *if $\mathbb{P}^{\max}[\Diamond \overline{S_\varepsilon}] < \varepsilon$, i.e. the probability of ever exiting S_ε is smaller than ε.*

When ε is clear from the context, we may refer to an ε-core by "core". Observe that the core condition is equivalent to $\mathbb{P}^{\min}[\Box S_\varepsilon] \geq 1 - \varepsilon$.

The set of all states S trivially is a core for any ε. Naturally, we are interested in finding a core which is as small as possible, which we call a *minimal core*.

▶ **Definition 4** (Minimal Core). *Let $\mathcal{M}$ be an MDP and $\varepsilon > 0$. $S^*_\varepsilon \subseteq S$ is a* minimal ε-core *if it is inclusion minimal, i.e. S^*_ε is an ε-core and there exists no ε-core $S'_\varepsilon \subsetneq S^*_\varepsilon$.*

When ε is clear from the context, we may refer to a minimal ε-core by "minimal core". In the running example, a minimal core for $\varepsilon = 10^{-6}$ would contain all states except the "bit flipped" state and the "recovery" subsystem, since they are reached only with probability $\tau < \varepsilon$.

▶ Remark 5. We note that this idea may seem similar to the one of [19], but is subtly different. In that work, the authors consider a classical reachability problem using value iteration. They approximate the exit probability of a *fixed* set $S_?$ to bound the error on the computed reachability.

In the following, we derive basic properties of cores, show how to efficiently construct them, and relate them to the approaches of [2, 6].

First, we observe that finding a core of a given size (for a non-trivial ε) is NP-complete.

▶ **Theorem 6.** *For $0 < \varepsilon < \frac{1}{4}$, $\{(\mathcal{M}, k) \mid \mathcal{M}$ has an ε-core of size $k\}$ is NP-complete.*

Observe that this result only implies that finding minimal cores is hard. In the following section, we introduce a learning-based approach which quickly identifies reasonably sized cores.

▶ **Theorem 7.** *Let $\mathcal{M}$ be an MDP and $\varepsilon > 0$. A set $S_\varepsilon \subseteq S$ is an ε-core of $\mathcal{M}$ if and only if for every subset of states $R \subseteq S$ we have that $0 \le \mathbb{P}^{\max}[\Diamond R] - \mathbb{P}^{\max}[\Diamond(R \cap S_\varepsilon) \cap \Box S_\varepsilon] < \varepsilon$.*

This theorem shows that for any reachability objective R, we can determine $\mathbb{P}^{\max}[\Diamond R]$ up to ε precision by determining the reachability of R on the sub-model induced by any ε-core, i.e. by only considering runs which remain inside S_ε. This also shows that, in general, cores are necessary to determine reachability up to precision ε.

We emphasize that this does *not* imply that identifying a core is necessary for all queries. For example, we have that $\mathbb{P}^{\max}_{s_0}[\Diamond\{s_0\}] = 1$ even without constructing any core. Nevertheless, for any non-trivial property, i.e. a reachability query with value less than $1-\varepsilon$, a computation restricted to a subset which does not satisfy the core property cannot give ε-guarantees on its results – only lower bounds can be proven. Thus, a set of states satisfying the core property has to be considered for non-trivial properties. In particular, the approach of [6] implicitly builds a core for such properties.

Of course, one could simply construct the whole state set S for the computation, which trivially satisfies the core condition. But, using the methods presented in the following section, we can efficiently identify a considerably smaller core. In particular, we observe in Section 5 that for some models we are able to identify a very small core orders of magnitude faster than the construction of the state set S, speeding up subsequent computations drastically.

3.2 Learning a Core

We introduce a sampling based algorithm which builds a core. In the interest of space, we only briefly describe the algorithm. Further discussion can be found in [12, App. A.2] and [6]. The algorithm is structurally very similar to the one presented in [6]. Nevertheless, we present it explicitly here since (i) it is significantly simpler and (ii) we introduce modifications later on.

We assume that the model is described by an initial state and a successor function, yielding all possible actions and the resulting distribution over successor states. This allows us to only construct a small fraction of the state space and achieve sub-linear runtime for some models.

During the execution of the algorithm, the system is traversed by following the successor function, starting from the initial state. Each state encountered is stored in a set of *explored* states, all other, not yet visited states are *unexplored.* Unexplored successors of explored states are called *partially explored.* Furthermore, the algorithm stores for each (explored) state s an upper bound $U(s)$ on the probability of reaching some unexplored state starting from s. The algorithm gradually grows the set of explored states and simultaneously updates these upper bounds, until the desired threshold is achieved in the initial state, i.e. $U(s_0) < \varepsilon$, and thus the set of explored states provably satisfies the core property. In particular, the upper bound is updated by sampling a path according to SamplePath and back-propagating the values along that path using Bellman backups.

SamplePath samples paths following some heuristic. In particular, it does not have to follow the transition probabilities given by the successor function. For example, a successor might be sampled with probability proportional to its upper bound times the transition

Algorithm 1 LearnCore.

```
Input: MDP $\mathcal{M}$, precision $\varepsilon > 0$, upper bounds $U$, state set $S_\varepsilon$ with $s_0 \in S_\varepsilon$
Output: $S_\varepsilon$ s.t. $S_\varepsilon$ is an $\varepsilon$-core
1: while $U(s_0) \geq \varepsilon$ do
2:     $\varrho \leftarrow$ SamplePath($s_0$, $U$)                              ▷ Generate path
3:     $S_\varepsilon \leftarrow S_\varepsilon \cup \varrho$                                     ▷ Expand core
4:     UpdateECs($S_\varepsilon, U$)
5:     for $s$ in $\varrho$ in reverse order do                          ▷ Back-propagate values
6:         $U(s) \leftarrow \max_{a \in A(s)} \sum_{s' \in S} \Delta(s, a, s') \cdot U(s')$
7: return $S_\varepsilon$
```

probability. The intuition behind this approach is that all states which are unlikely to be reached are not relevant and hence do not need to be included in the core. By trying to reach unexplored states the algorithm likely only reaches states which indeed are important.

UpdateECs identifies MECs of the currently explored sub-system and "collapses" them into a single representative state. This is necessary to ensure convergence of the upper bounds to the correct value – technically this process removes spurious fixed points of U.

For (a.s.) termination, we only require that the sampling heuristic is "(almost surely) fair". This means that (i) any partially explored state is reached eventually (a.s.), in order to explore a sufficient part of the state space, and (ii) any explored state with $U(s) > 0$ is visited infinitely often (a.s.), in order to back-propagate values accordingly. Further, we require that the initial upper bounds are consistent with the given state set, i.e. $U(s) \geq \mathbb{P}_s^{\max}[\Diamond \overline{S_\varepsilon}]$. This is trivially satisfied by $U(\cdot) = 1$. Note that in contrast to [6], the set whose reachability we approximate dynamically changes and, further, only upper bounds are computed.

▶ **Theorem 8.** *Algorithm 1 is correct and terminates (a.s.) if* SamplePath *is (a.s.) fair and the given upper bounds U are consistent with the given set S_ε.*

Proof Sketch. *Correctness*: By assumption $U(s)$ initially is a correct upper bound for the "escape" probability, i.e. $U(s) \geq \mathbb{P}_s^{\max}[\Diamond \overline{S_\varepsilon}]$. Each update (a Bellman backup) preserves correctness, independent of the sampled path. Hence, if $U(s_0) < \varepsilon$, we have $\mathbb{P}_s^{\max}[\Diamond \overline{S_\varepsilon}] < \varepsilon$.

Termination: As we assumed that SamplePath is (a.s.) fair, eventually (a.s.) the whole model will be explored, and all MECs will be collapsed by UpdateECs. Then, all states are visited infinitely often (a.s.), and thus all upper bounds will eventually converge to 0. ◀

As Algorithm 1 is correct and terminates for any faithful upper bounds and initial state set, we can restart the algorithm and interleave it with other approaches refining the upper bounds. For example, one could periodically update the upper bounds using, e.g., strategy iteration. Further, we can reuse the computed upper bounds and state set to compute a core for a tighter precision.

3.3 Using Cores for Verification

We explain how a core can be used for verification and how our approach differs from existing ones. Clearly, we can compute reachability or safety objectives on a given core ε-precisely. In this case, our approach is not too different from the one in [6]. Yet, we argue that our approach yields a stronger result. Due to cores being an intrinsic object, we are able to reuse and adapt this idea easily to many other objectives. Observe that a dedicated adaption may still yield slightly better performance, but requires significantly more work. For example, see [2] for an adaption to mean payoff.

To see how we can connect our idea to mean payoff, we briefly explain this objective and then recall an observation of [2]. First, rational rewards are assigned to each state, which are obtained on each visit. Then, the mean payoff of a particular run is the limit average reward obtained from the visited states. The mean payoff under a particular strategy then is obtained by integrating over the set of all runs. As mentioned by [2], a mean payoff objective can be decomposed into a separate analysis of each (explored) MEC and a (weighted) reachability query

$$\text{optimal mean payoff} = \sup_{\pi \in \Pi} \sum\nolimits_{M \in \mathsf{MEC}(\mathcal{M})} \text{mean payoff of } \pi \text{ in } M \cdot \mathbb{P}^{\pi}\left[\Diamond\Box M\right].$$

Since we can bound the reachability on unexplored MECs by the core property, we can easily bound the error on the computed mean payoff (assuming we know an a-priori lower and upper bound on the reward function). Consequently, we can approximate the optimal mean payoff by only analysing the corresponding core.

Similarly, LTL queries and parity objectives can be answered by a decomposition into analysis of MECs and their reachability. Intuitively, given a MEC one can decide whether the MEC is “winning” or “losing” for these objectives. The overall probability of satisfying the objective then equals the probability of reaching a winning MEC [3]. Again, we can bound the reachability of unexplored MECs and thus the error we incur when only analysing the core.

In general, many verification tasks can be decomposed into a reachability query and analysis of specific parts of the system. Since our framework is agnostic of the verification task in question, it can be transparently plugged in to obtain significant speed-ups.

We highlight that our approach is directly applicable to models with infinite state space, since finite cores still may exist for these models.

4 Beyond Infinite Horizon

In the previous section, we have seen that MECs play an essential role for many objectives. Hence, we study the interplay between cores and MECs.

▶ **Proposition 9.** *Let $M \in \mathsf{MEC}(\mathcal{M})$ be a MEC. If there is a state $s \in M$ with $\mathbb{P}^{\max}[\Diamond\{s\}] \geq \varepsilon$ then $M \subseteq S_\varepsilon$ for every ε-core S_ε.*

Proof. Recall that for $s, s' \in M$, we have $\mathbb{P}^{\max}_s[\Diamond\{s'\}] = 1$, thus $\mathbb{P}^{\max}[\Diamond\{s\}] = \mathbb{P}^{\max}[\Diamond\{s'\}] \geq \varepsilon$ and thus $s' \in S_\varepsilon$. ◀

This implies that sufficiently reachable MECs always need to be contained in a core *entirely*. Many models comprise only a few or even a single MEC, e.g., restarting protocols like mutual exclusion or biochemical models of reversible reactions. Together with the result of Theorem 7, i.e. constructing a core is necessary for ε-precise answers, this shows that in general we cannot hope for any reduction in state space, even when only requiring ε-optimal solutions for *any* of the discussed properties. In particular, the approach of [6] necessarily has to explore the full model. Yet, real-world models often exhibit a particular structure, with many states only being visited infrequently. Since we necessarily have to give up on something to obtain further savings, we propose an extension of our idea, motivated by a modification of our running example.

Instead of a one-way trip, consider the plane going back and forth between the origin and the destination, as shown in Figure 2. Clearly, the plane eventually will suffer from a bit flip, *independently* of the strategy. Furthermore, assuming that there is a non-zero probability of not being able to recover from the error, the plane will eventually crash.

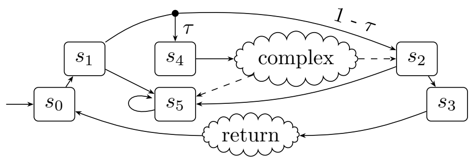


Figure 2 An adaptation of the model from Figure 1, with an added return trip, represented by the "return" node. State and action labels have been omitted in the interest of space.

We make two observations. First, any core needs to contain at least parts of the recovery sub-system, since it is reached with probability 1. Thus, this (complex) sub-system has to be constructed. Second, the witness strategy is meaningless, since any strategy is optimal – the crash cannot be avoided in the long run. In particular, deliberately crashing the plane has the same long run performance as flying it "optimally". In practice, we often actually are interested in the performance of such a model for a long, but not necessarily infinite horizon.

To this end, one could compute the step bounded variants of the objectives, but this incurs several problems: (i) choosing a sensible step bound n, (ii) computational overhead (a precise computation has a worst-case complexity of $|\Delta| \cdot n$ even for reachability), and (iii) all states reachable within n steps have to be constructed (which equals the whole state space for practically all models and reasonable choices of n). In the following, we present a different approach to this problem, again based on the idea of cores.

4.1 Finite-Horizon Cores

We introduce *finite-horizon cores*, which are completely analogous to (infinite-horizon) cores, only with a step bound attached to them.

▶ **Definition 10** (Finite-Horizon Core). *Let $\mathcal{M}$ be an MDP, $\varepsilon > 0$, and $n \in \mathbb{N}$. A set $S_{\varepsilon,n} \subseteq S$ is an* n-step ε-core *if $\mathbb{P}^{\max}[\lozenge^{\leq n}\overline{S_{\varepsilon,n}}] < \varepsilon$ and it is a* minimal n-step ε-core *if it is additionally inclusion minimal.*

As before, whenever n or ε are clear from the context, we may drop the corresponding part of the name. Again, similar properties hold and we omit the completely analogous proof.

▶ **Theorem 11.** *Let $\mathcal{M}$ be an MDP, $\varepsilon > 0$, and $n \in \mathbb{N}$. Then $S_{\varepsilon,n} \subseteq S$ is an n-step ε-core if and only if for all $R \subseteq S$ we have $0 \leq \mathbb{P}^{\max}[\lozenge^{\leq n} R] - \mathbb{P}^{\max}[\lozenge^{\leq n}(R \cap S_{\varepsilon,n}) \cap \square^{\leq n} S_{\varepsilon,n}] < \varepsilon$.*

These finite-horizon cores are much smaller than their "infinite" counterparts on some models, even for large n. For instance, in our modified running example of Figure 2, omitting the "complex" states gives an n-step core even for very large n (depending on τ). On the other hand, finding such finite cores seems to be harder in practice. Naively, one could apply the core learning approach of Algorithm 1 to a modified model where the number of steps is encoded into the state space, i.e. $S' = S \times \{0, \ldots, n\}$. Unfortunately, this yields abysmal performance, since we store and back-propagate $|S| \cdot n$ values instead of only $|S|$. Nevertheless, we can efficiently approximate them by enhancing our previous approach with further observations.

4.2 Learning a Finite Core

In Algorithm 2, we present our learning variant for the finite-horizon case. This algorithm is structurally very similar to the previous Algorithm 1. The fundamental difference is in Line 6, where the bounds are updated. One key observation is that the probability of

Algorithm 2 LearnFiniteCore.

```
Input: MDP $\mathcal{M}$, precision $\varepsilon > 0$, step bound $n$, upper bounds GetBound / UpdateBound,
    state set $S_{\varepsilon,n}$ with $s_0 \in S_{\varepsilon,n}$
Output: $S_{\varepsilon,n}$ s.t. $S_{\varepsilon,n}$ is an $n$-step $\varepsilon$-core
1: while GetBound$(s_0, n) \geq \varepsilon$ do
2:     $\varrho \leftarrow$ SamplePath$(s_0, n,$ GetBound$)$                        ▷ Generate path
3:     $S_{\varepsilon,n} \leftarrow S_{\varepsilon,n} \cup \varrho$                                      ▷ Update Core
4:     for $i \in [n-1, n-2, \dots, 0]$ do                              ▷ Back-propagate values
5:         $s \leftarrow \varrho_i$, $r \leftarrow n - i$
6:         UpdateBound $\big(s, r, \max_{a \in A(s)} \sum_{s' \in S} \Delta(s,a,s') \cdot$ GetBound$(s', r-1)\big)$
7: return $S_{\varepsilon,n}$
```

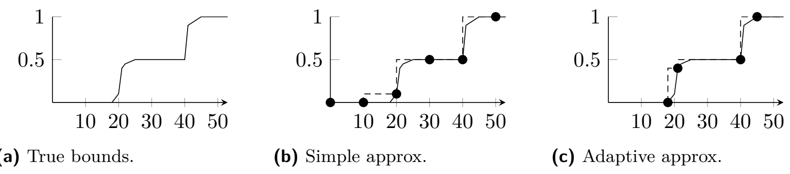


Figure 3 An example for the different approximation approaches. The graphs depict the probability of exiting the core on the y axis within a given amount of steps on the x axis by a solid line and the corresponding approximation returned by GetBounds by a dashed line. From left to right, we have example bounds, which agree with the dense representation, followed by our sparse approach, which over-approximates the bounds, but requires less memory, and finally an adaptive approach, which closely resembles the precise bounds while consuming less memory.

reaching some set R within k steps is at least as high as reaching it within $k-1$ steps, i.e. $\mathbb{P}_s^{\max}[\Diamond^{\leq k} R] < \varepsilon$ is non-decreasing in k for any s and $R \subseteq S$. Therefore, we can use function over-approximations to store upper bounds sparsely and avoid storing n values for each state. To allow for multiple implementations, we thus delegate the storage of upper bounds to an abstract function approximation, namely GetBound and UpdateBound. This approximation scheme is supposed to store and retrieve the upper bound of reaching unexplored states for each state and number of *remaining* steps. We only require it to give a *consistent* upper bound, i.e. whenever we call UpdateBound(s, r, p), GetBound(s, r') will return at least p for all $r' \geq r$. Moreover, we require the trivial result GetBound$(s, 0) = 0$ for all states s. In the following Section 4.3, we list several possible instantiations.

▶ **Theorem 12.** *Algorithm 2 is correct if* UpdateBound–GetBound *are consistent and correct w.r.t. the given state set* $S_{\varepsilon,n}$*. Further, if* UpdateBound *stores all values precisely and* SamplePath *samples any state reachable within n steps infinitely often (a.s.), the algorithm terminates (a.s.).*

Sketch. *Correctness*: As before, the upper bound function is only updated through Bellman backups, which preserve correctness.

Termination: Given that the upper bound function stores all values precisely, the algorithm is an instance of asynchronous value iteration, which is guaranteed to converge [18]. ◀

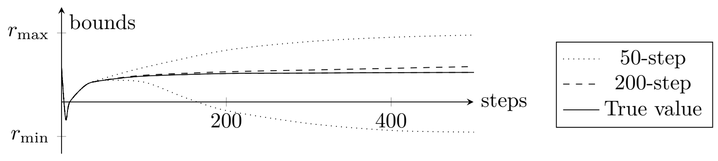


Figure 4 A schematic plot for an average reward extrapolation analysis on step bounded cores. The solid line represents the true value, while the dotted and dashed lines are the respective upper and lower bounds computed for a 50 and 200-step core. Note that the second dashed line (lower bound on the 200 core) coincides with the solid line (true value).

4.3 Implementing the function approximation

Several instances of the UPDATEBOUND–GETBOUND approach are outlined in Figure 3. The first, trivial implementation is dense storage, i.e. explicitly storing a table mapping $S \times \{0, \dots, n-1\} \to [0, 1]$. This table representation consumes an unnecessary amount of memory, since we do not need exact values in order to just guide the exploration. Hence, in our implementation, we use a simple sparse approach where we only store the value every K steps, where K manually chosen. This is depicted in Figure 3b for $K = 10$ – every black dot represents a stored value, the dashed lines represent the value returned by GETBOUNDS. The adaptive approach of Figure 3c adaptively chooses which values to store and is left for future work.

4.4 Stability and its applications

In this section, we explain the idea of a core's *stability*. Given an n-step core $S_{\varepsilon,n}$, we can easily compute the probability $\mathbb{P}^{\max}[\Diamond^{\leq N} \overline{S_{\varepsilon,n}}]$ of exiting the core within $N > n$ steps using, e.g., value iteration. The rate of increase of this exit probability intuitively gives us a measure of quality for a particular core. Should it rapidly approach 1 for increasing N, we know that the system's behaviour may change drastically within a few more steps. If instead this probability remains small even for large N, we can compute properties with a large step bound on this core with tight guarantees. We define stability as the whole function mapping the step bound N to the exit probability, since this gives a more holistic view on the system's behaviour than a singular value. In the following, we give an overview of how finite cores and the idea of stability can be used for analysis and interpretation, helping to design and understand complex systems.

As we have argued above, infinite-horizon properties may be deceiving, since (unrecoverable) errors often are bound to happen eventually. Consequently, one might be interested in a "very large"-horizon analysis instead. Unfortunately, such an analysis scales linearly both with the number of transitions and the horizon. Considering that many systems have millions of states, an analysis with a horizon of only 10,000 steps is already out of reach for existing tools. We first show how stable cores can be used for efficient extrapolation to such large horizons.

For simplicity, we consider reachability and argue how to transfer this idea to other objectives. We apply the ideas of interval iteration as used in, e.g., [10, 6], as follows. Intuitively, since we have no knowledge of the partially explored states, we simply assume the worst / best case for them, i.e. assign a lower bound of 0 and upper bound of 1. Furthermore, any explored target state is assigned a lower and upper bound of 1. By applying interval iteration, we can obtain bounds on the N-step and even unbounded reachability. Through the core property, the bounds for $N \leq n$ necessarily are smaller than ε. But, for larger N, there are no formal guarantees given by the core property. It might be the case that the core is left with probability 1 in $n+1$ steps. Nevertheless, in practice this allows us to get good approximations even for much larger bounds. Often the computation of an n-step core and subsequent approximation of the desired property is even faster than directly computing the N-step property, as shown in the evaluation.

For LTL and parity objectives, we can simply preprocess the obtained n-core by identifying the winning MECs and then applying the reachability idea, to obtain bounds on the satisfaction probability on the core. In the case of mean-payoff, we again require lower and upper bounds on the rewards $r_{\min}$ and $r_{\max}$ of the system in order to properly initialize the unknown values. Then, with the same approach, we can compute bounds on the n-step average reward by simply assign the lower and upper bounds $r_{\min}$ and $r_{\max}$ to all unexplored states instead of 0 and 1. See Figure 4 for a schematic plot of this analysis. Here, the 50-step core is too coarse for any reasonable analysis, it is unstable and can be exited with high probability. On the other hand, the 200-step core is very stable and accurately describes the system's behaviour for a longer period of time. Noticeably, it also contains a MEC guaranteeing a lower bound on the average reward, hence the lower bound actually agrees with the true value. Since the system may be significantly larger than the bounded cores or even infinitely large, this analysis potentially is much more efficient than analysis of the whole system, as shown in the experimental evaluation.

Note that we cannot use this method to obtain arbitrarily precise results. Given some n-step core and some (step bounded) property, there is a maximal precision we can achieve, depending on the property and the structure of the model. Hence, this method primarily is useful to quickly obtain an overview of a system's behaviour instead of verifying a particular property. As we have argued, one cannot avoid constructing a particular part of the state space in order to obtain an ε-precise result. Nevertheless, this may provide valuable insights in a system, quickly giving a good overview of its behaviour or potential design flaws.

We highlight that the presented algorithm can incrementally refine cores. For example, if a 100-step core does not yield a sufficiently precise extrapolation, the algorithm can reuse the computed core in order to construct a 200-step core. By applying this idea in an interactive loop, one can extract a condensed representation of the systems behaviour automatically, with the possibility for further refinements until the desired level of detail has been obtained.

5 Experimental Evaluation

In this section we give practical results for our algorithms on some examples, both the hand-crafted plane model and hand-picked models from case studies.

5.1 Implementation Details

We implemented our approach in Java, using PRISM [13] as a library for parsing its modelling language and basic computations. The implementation supports Markov chains, continuous-time Markov chains (CTMC, via embedding or uniformization [18, Ch. 11.5]) and Markov decision processes. Further, we implemented our own version of some utility classes, e.g., explicit MDP representation and MEC decomposition.

Inspired by the results of [6], we considered the following sampling heuristics. Given a state s, each heuristic first selects an action a which maximizes the expected upper bound. If there are multiple such actions, one of them is randomly selected. Then, a successor is chosen as follows: The RN (Random) heuristics samples a successors according

Table 1 Summary of our experimental results on several models and configurations for the infinite horizon core learning. The "PRISM" column shows the total number of states and construction time when explored with the explicit engine. The following columns show the size and total construction time of a 10^{-6}-core for each of the sampling heuristics.

Model	Param.	PRISM		RN		GD		MX	
`zeroconf`	100; 5; 0.1	496,291	8.4s	17,805	2.3s	1,072	0.4s	1,450	0.5s
(`N`; `K`; `loss`)	100; 10; 0.1	$3.0 \cdot 10^6$	55s	11,900	1.7s	1,006	0.3s	1,730	0.5s
	100; 15; 0.1	$4.7 \cdot 10^6$	159s	16,997	2.4s	1,067	0.4s	2,002	0.7s
`airplane`	100; `ff`	10,208	0.4s	6	0.1s	6	0.1s	6	0.1s
(`size`; `return`)	10000; `ff`	MEMOUT		6	0.1s	6	0.1s	6	0.1s
`brp`	20; 10	2,933	0.2s	2,352	0.3s	2,568	0.4s	2,675	0.5s
(`N`; `MAX`)	20; 100	26,423	0.6s	7,060	0.6s	3,421	0.4s	3,679	0.5s
	20; 1000	261,323	0.6s	7,624	0.6s	5,118	0.4s	3,912	0.5s
`wlan`	—	345,000	4.5s	344,835	52s	344,996	50s	344,997	52s

to $\Delta(s, a)$. The GD (Guided) approach samples a successor weighted by the respective upper bound, i.e. randomly select a state with probability proportional to $U(s') \cdot \Delta(s, a, s')$ or $\textsc{GetBound}(s', r) \cdot \Delta(s, a, s')$, respectively. Finally, MX (Max) samples a successor s' with probability proportional only to its upper bound $U(s')$ or $\textsc{GetBound}(s', r)$, respectively.

Recall that our algorithms can be restarted with faithful upper bounds and thus we can interleave it with other computations. In our implementation we alternate between the guided exploration of Algorithm 2 and precise computation on the currently explored set of states, guaranteeing convergence in the finite setting.

5.2 Models

In our evaluation, we considered the following models. All except the `airplane` model are taken from the PRISM case studies [16]. `airplane` is our running example from Figure 1 and Figure 2, respectively. The parameter `return` controls whether a return trip is possible, `size` quadratically influences the size of the "recovery" region. `zeroconf` [14] describes the IPv4 Zeroconf Protocol with `N` hosts, the number `K` of probes to send, and a probability of a message `loss`. `wlan` [15] is a model of two WLAN stations in a fixed network topology sending messages on the shared medium. `brp` is a DTMC modelling a file transfer of `N` chunks with bounded number `MAX` of retries per chunk. Finally, `cyclin` is a CTMC modelling the cell cycle control in eukaryotes with `N` molecules. We analyse this model using uniformization.

5.3 Results

We evaluated our implementation on an i7-4700MQ 4x2.40 GHz CPU with 16 GB RAM. We used a default precision of 10^{-6} for all experiments. The results for the infinite and finite construction are summarized in Table 1 and Table 2, respectively. We briefly discuss them in the following sections. Note that the results may vary due to the involved randomization.

5.3.1 Infinite Cores

As already explained in [6], the `zeroconf` model is very well suited for this type of analysis, since a lot of the state space is hardly reachable. In particular, most states are a result of collisions and several message losses, which is very unlikely. Consequently, a very small part

Table 2 Summary of our experimental results on several models and configurations for the finite-horizon 10^{-6}-core learning with 100 step bound; using the notation from Table 1.

Model	Param.	PRISM		RN		GD		MX	
`airplane`	100; `tt`	20,413	0.6s	11	0.3s	11	0.3s	554	0.6s
(`size`; `return`)	10000; `tt`	MEMOUT		11	0.3s	11	0.3s	603	0.6s
`wlan`	—	345,000	4.4s	36,718	15s	37,284	11s	36,825	12s
`cyclin`	4	431,101	12s	9,122	1,649s	4,380	3.9s	99,325	93s
(`N`)	5	$2.3 \cdot 10^6$	78s	26,925	8,840s	11,419	13s	817,058	1,462s

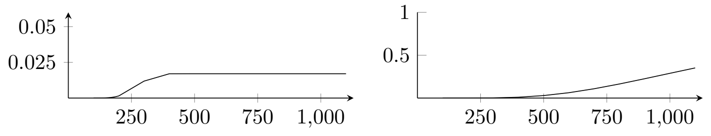


Figure 5 Stability analysis of the `wlan` (left) and `cyclin`(`N` = 4) (right) 100-step core built with the GD heuristic. The graphs show the probability of exiting the respective core within the given amount of steps. Note that the y axis of the `wlan` graph is scaled for readability.

of the model already satisfies the core property. In particular, the size of the core remains practically constant when increasing the parameter K, as only unimportant states are added to the system. Observe that the order of magnitude of explored states is very similar to the experiments from [6]. The same holds true for the `airplane` model, where a significant number of states is dedicated to recovering from an unlikely error. Hence, a small core exists independently of the total size of the model. The `brp` model shows applicability of the approach to Markov chains. In line with the other results, when scaling up the maximal number of allowed errors, the size of the core changes sub-linearly, since repeated errors are increasingly unlikely. In case of the `wlan` model, we observe that all our methods essentially construct the full model.

Comparison to [6]. We also executed the tool presented in [6] where applicable (only MDP are supported). We tested the tool both with a bogus `false` property, i.e. approximating the probability of reaching the empty set, which corresponds to constructing a core, and an actual property. We used the `MAX_DIFF` heuristic of [6], which is similar to GD. Especially on the `false` property, our tool consistently outperformed the previous one in terms of time and memory by up to several orders of magnitude. We suspect that this is mostly due to a more efficient implementation. The number of explored states was similar, as to be expected in light of Theorem 7 and its discussed consequences.

5.3.2 Finite Cores

As expected, the finite core construction yields good results on the `airplane` model, constructing only a small fraction of the state space. Interestingly, the MX heuristic explores significantly more states, which is due to this heuristic ignoring probabilities when selecting a successor and thus sampling a few paths in the recovery region. Also on the real-world models `wlan` and `cyclin`, the constructed 100-step core is significantly smaller than the whole model. For `wlan`, the construction of the respective cores unfortunately takes longer than building the whole model. We conjecture that a more fine-tuned implementation can

Method	Time		States
	model	bounds	
50 steps	0.4s	1.0s	2,137
100 steps	0.9s	3.5s	6,151
200 steps	4.0s	15.3s	22,989
Complete	8.7s	242.4s	431,101

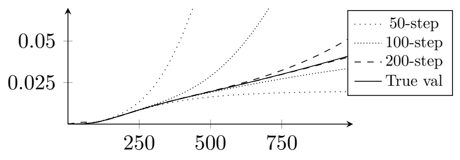


Figure 6 Overview of an extrapolation analysis for `cyclin`$(N = 4)$. We computed several step-bounded cores with precision 10^{-3}. On these, we computed bounds of a reachability query with increasing step bound. The table on the left lists the time for model construction + computation of the bounds for 1000 steps and the size of the constructed model. The plot on the right shows the upper and lower bounds computed for each core together with the true value. Observe that for growing step-size of the core, the approximation naturally gets more precise.

overcome this issue. In any case, model checking on the explored sub-system supposedly terminates significantly faster since only a much smaller state space is investigated, and the core can be re-used for more queries.

Finally, we applied the idea of stability from Section 4.2 on the `wlan` and `cyclin` models, with results outlined in Figure 5. Interestingly, for the `wlan` model, the escape probability stabilizes at roughly 0.017 and we obtain the exact same probability for *all* heuristics, even for $N = 10{,}000$. This suggests that by building the 100-step core we identified a very stable sub-system of the whole model. Additionally, we observe that at 200-400 steps, the behaviour of the system significantly changes. For the `cyclin` model, we instead observe a continuous rise of the exit probability. Nevertheless, even with 500 additional steps, the core still is only exited with a probability of roughly 6% and thus closely describes the system's behaviour.

On the `cyclin` model, we also applied our idea of extrapolation. The results are summarized in Figure 6. To show how performant this approach is, we reduced the precision of the core computation to 10^{-3}. Despite this coarse accuracy, we are able to compute accurate bounds on a 1000-step reachability query over 10 times faster by only building the 200-step core instead of constructing the full model. These results suggest that our idea of using the cores for extrapolation in order to quickly gain understanding of a model has a vast potential.

5.3.3 Heuristics

Overall, we see that the unguided, random sampling heuristic RN often is severely outperformed by the guided approaches GD and MX, both in terms of runtime and constructed states. Surprisingly, the differences between GD and MX often are small, considering that MX is significantly more “greedy” by completely ignoring the actual transition probabilities. We conjecture that this greediness is the reason for the abysmal performance of MX on the `cyclin` model, where GD seems to strike the right balance between exploration and exploitation. Altogether, the results show that a sophisticated heuristic increases performance by orders of magnitude and further research towards optimizing these heuristic may prove beneficial.

6 Conclusion

We have presented a new framework for approximate verification of probabilistic systems via partial exploration and applied it to both Markov chains and Markov decision processes. Our evaluation shows that, depending on the structure of the model, this approach can yield significant state space savings and thus reduction in model checking times. Our central idea – finding relevant sub-parts of the state space – can easily be extended to further models, e.g., stochastic games, and objectives, e.g., mean payoff. We have also shown how this idea can be transferred to the step-bounded setting and derived the notion of stability. This in turn allows for an efficient analysis of long-run properties and strongly connected systems.

Future work includes implementing a more sophisticated function approximation for the step-bounded case, e.g., as depicted in Figure 3c. Here, an adaptive method could yield further insight in the model by deriving points of interest, i.e. an interval of remaining steps where the exit probability significantly changes. These breakpoints might indicate a significant change in the systems behaviour, e.g., the probability of some error occurring not being negligible any more, yielding interesting insights into the structure of a particular model. For example, in the bounds of Figure 3, the regions around 20 and 40 steps, respectively, seems to be of significance.

Moreover, a more sophisticated sampling heuristic to be used in SamplePath could be of interest. For example, one could apply an advanced machine learning technique here, which also considers state labels or previous decisions and their outcomes.

In the spirit of [6], our approach also could be extended to a PAC algorithm for black-box systems. Extensions to stochastic games and continuous time systems are also possible.

Further interesting variations are cores for discounted objectives [20] or cost-bounded cores, a set of states which is left with probability smaller than ε given that at most k cost is incurred. This generalizes both the infinite (all edges have cost 0) and the step bounded cores (all edges have cost 1) and allows for a wider range of analysis.

References

1 Pranav Ashok, Yuliya Butkova, Holger Hermanns, and Jan Křetínskỳ. Continuous-Time Markov Decisions Based on Partial Exploration. In *ATVA*, pages 317–334. Springer, 2018.

2 Pranav Ashok, Krishnendu Chatterjee, Przemyslaw Daca, Jan Kretínský, and Tobias Meggendorfer. Value Iteration for Long-Run Average Reward in Markov Decision Processes. In *CAV*, pages 201–221, 2017. doi:10.1007/978-3-319-63387-9_10.

3 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

4 Richard Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

5 Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II: Approximate Dynamic Programming*. Athena Scientific, 2012.

6 Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Křetínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of Markov Decision Processes Using Learning Algorithms. In *ATVA*, pages 98–114. Springer, 2014. doi:10.1007/978-3-319-11936-6_8.

7 Costas Courcoubetis and Mihalis Yannakakis. The Complexity of Probabilistic Verification. *J. ACM*, 42(4):857–907, 1995. doi:10.1145/210332.210339.

8 Pedro R. D'Argenio, Bertrand Jeannet, Henrik Ejersbo Jensen, and Kim Guldstrand Larsen. Reduction and Refinement Strategies for Probabilistic Analysis. In *PAPM-PROBMIV*, pages 57–76. Springer, 2002.

9 Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *CAV*, pages 592–600. Springer, 2017.

10 Serge Haddad and Benjamin Monmege. Reachability in MDPs: Refining convergence of value iteration. In *International Workshop on Reachability Problems*, pages 125–137. Springer, 2014.

11 Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. PASS: abstraction refinement for infinite probabilistic models. In *TACAS*, pages 353–357. Springer, 2010.

12 Jan Křetínský and Tobias Meggendorfer. Of Cores: A Partial-Exploration Framework for Markov Decision Processes. *arXiv e-prints*, June 2019. `arXiv:1906.06931`.

13 M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *TOOLS*, pages 200–204, 2002.

14 Marta Kwiatkowska, Gethin Norman, David Parker, and Jeremy Sproston. Performance analysis of probabilistic timed automata using digital clocks. *FMSD*, 29(1):33–78, 2006.

15 Marta Kwiatkowska, Gethin Norman, and Jeremy Sproston. Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In *Process Algebra and Probabilistic Methods: Performance Modeling and Verification*, pages 169–187. Springer, 2002.

16 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. The PRISM Benchmark Suite. In *QEST*, pages 203–204. IEEE Computer Society, 2012. The models are accessible at `http://www.prismmodelchecker.org/casestudies/`.

17 H Brendan McMahan, Maxim Likhachev, and Geoffrey J Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML*, pages 569–576. ACM, 2005.

18 M.L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley and Sons, 1994.

19 Tim Quatmann and Joost-Pieter Katoen. Sound Value Iteration. In *CAV (1)*, volume 10981 of *LNCS*, pages 643–661. Springer, 2018.

20 Aaron Sidford, Mengdi Wang, Xian Wu, and Yinyu Ye. Variance Reduced Value Iteration and Faster Algorithms for Solving Markov Decision Processes. In *SODA*, pages 770–787. SIAM, 2018.

F Conditional Value-at-Risk for Reachability and Mean Payoff in Markov Decision Processes. LICS 2018

This section has been published as **peer-reviewed conference paper**.

> Jan Křetínský and Tobias Meggendorfer. ‘Conditional Value-at-Risk for Reachability and Mean Payoff in Markov Decision Processes’. In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*. Ed. by Anuj Dawar and Erich Grädel. ACM, 2018, pp. 609–618. DOI: `10.1145/3209108.3209176`. URL: `https://doi.org/10.1145/3209108.3209176`

Synopsis Humans are, by nature, risk-aware and -averse (to an extent). This is reflected in science by extensive research into risk and risk-influenced behaviour in areas like psychology, finance, operations research, and many more. Yet, the verification community hardly dealt with this issue at all. Typically, probabilistic outcomes are aggregated and optimized w.r.t. expectation, which is completely oblivious to risk. A first small step towards risk-analysis was done by considering worst-case analysis, which however is extremely prohibitive—after all, we want to be able to take controlled risks. In this work, we provide a significant contribution towards risk quantification. We introduce the established notion of *conditional value-at-risk* to the area of verification. We provide precise bounds on the computational complexity and structure of optimal strategies. In particular, we show that synthesizing risk-aware behaviour is not more costly than pure expectation optimization, motivating further research in this direction.

Contributions of the thesis author Composition of the manuscript except abstract and introduction. Discussion and revision of the entire manuscript. Sole contribution of all results and proofs presented in the paper, with the exception of the Proof sketches for Theorems 6.8 and 6.9.

Conditional Value-at-Risk for Reachability and Mean Payoff in Markov Decision Processes

Jan Křetínský
Institut für Informatik (I7)
Technische Universität München
Garching bei München, Bavaria, Germany
jan.kretinsky@in.tum.de

Tobias Meggendorfer
Institut für Informatik (I7)
Technische Universität München
Garching bei München, Bavaria, Germany
tobias.meggendorfer@in.tum.de

Abstract

We present the *conditional value-at-risk (CVaR)* in the context of Markov chains and Markov decision processes with reachability and mean-payoff objectives. CVaR quantifies risk by means of the expectation of the worst p-quantile. As such it can be used to design risk-averse systems. We consider not only CVaR constraints, but also introduce their conjunction with expectation constraints and quantile constraints (value-at-risk, VaR). We derive lower and upper bounds on the computational complexity of the respective decision problems and characterize the structure of the strategies in terms of memory and randomization.

CCS Concepts • **Theory of computation → Verification by model checking**;

ACM Reference Format:
Jan Křetínský and Tobias Meggendorfer. 2018. Conditional Value-at-Risk for Reachability and Mean Payoff in Markov Decision Processes. In *LICS '18: 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, July 9–12, 2018, Oxford, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3209108.3209176

1 Introduction

Markov decision processes (MDP) are a standard formalism for modelling stochastic systems featuring non-determinism. The fundamental problem is to design a strategy resolving the non-deterministic choices so that the systems' behaviour is optimized with respect to a given objective function, or, in the case of multi-objective optimization, to obtain the desired trade-off. The objective function (in the optimization phrasing) or the query (in the decision-problem phrasing) consists of two parts. First, a payoff is a measurable function assigning an outcome to each run of the system. It can be real-valued, such as the *long-run average reward* (also called *mean payoff*), or a two-valued predicate, such as *reachability*. Second, the payoffs for single runs are combined into an overall outcome of the strategy, typically in terms of *expectation*. The resulting objective function is then for instance the expected long-run average reward, or the probability to reach a given target state.

Risk-averse control aims to overcome one of the main disadvantages of the expectation operator, namely its ignorance towards the incurred risks, intuitively phrased as a question *"How bad are the bad cases?"* While the standard deviation (or variance) quantifies the spread of the distribution, it does not focus on the bad cases and thus fails to capture the risk. There are a number of quantities used to deal with this issue:

- The *worst-case* analysis (in the financial context known as discounted maximum loss) looks at the payoff of the worst possible run. While this makes sense in a fully non-deterministic environment and lies at the heart of verification, in the probabilistic setting it is typically unreasonably pessimistic, taking into account events happening with probability 0, e.g., never tossing head on a fair coin.
- The *value-at-risk* (VaR) denotes the worst p-quantile for some $p \in [0, 1]$. For instance, the value at the 0.5-quantile is the median, the 0.05-quantile (the *vigintile* or *ventile*) is the value of the best run among the 5% worst ones. As such it captures the "reasonably possible" worst-case. See Fig. 1 for an example of VaR for two given probability density functions. There has been an extensive effort spent recently on the analysis of MDP with respect to VaR and the re-formulated notions of quantiles, percentiles, thresholds, satisfaction view etc., see below. Although VaR is more realistic, it tends to ignore outliers too much, as seen in Fig. 1 on the right. VaR has been characterized as *"seductive, but dangerous"* and *"not sufficient to control risk"* [8].
- The *conditional value-at-risk* (average value-at-risk, expected shortfall, expected tail loss) answers the question *"What to expect in the bad cases?"* It is defined as the expectation over all events worse than the value-at-risk, see Fig. 1. As such it describes the lossy tail, taking outliers into account, weighted respectively. In the degenerate cases, CVaR for $p = 1$ is the expectation and for $p = 0$ the (probabilistic) worst case. It is an established risk metric in finance, optimization and operations research, e.g. [1, 33], and *"is considered to be a more consistent measure of risk"* [33]. Recently, it started permeating to areas closer to verification, e.g. robotics [13].

Our contribution In this paper, we investigate optimization of MDP with respect to CVaR as well as the respective trade-offs with expectation and VaR. We study the VaR and CVaR operators for the first time with the payoff functions of weighted reachability and

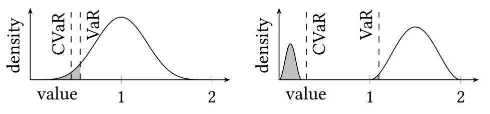


Figure 1. Illustration of VaR and CVaR for some random variables.

mean payoff, which are fundamental in verification. Moreover, we cover both the single-dimensional and the multi-dimensional case.

Particularly, we define CVaR for MDP and show the peculiarities of the concept. Then we study the computational complexity and the strategy complexity for various settings, proving the following:

- The single dimensional case can be solved in polynomial time through linear programming, see Section 5.
- The multi-dimensional case is NP-hard, even for CVaR-only constraints. Weighted reachability is NP-complete and we give PSPACE and EXPSPACE upper bounds for mean payoff with CVaR and expectation constraints, and with additional VaR constraints, respectively, see Section 6. (Note that already for the sole VaR constraints only an exponential algorithm is known; the complexity is an open question and not even NP-hardness is known [15, 32].)
- We characterize the strategy requirements, both in terms of memory, ranging from memoryless, over constant-size to infinite memory, and the required degree of randomization, ranging from fully deterministic strategies to randomizing strategies with stochastic memory update.

While dealing with the CVaR operator, we encountered surprising behaviour, preventing us to trivially adapt the solutions to the expectation and VaR problems:

- Compared to, e.g., expectation and VaR, CVaR does not behave linearly w.r.t. stochastic combination of strategies.
- A conjunction of CVaR constraints already is NP-hard, since it can force a strategy to play deterministically.

1.1 Related work

Worst case Risk-averse approaches optimizing the worst case together with expectation have been considered in beyond-worst-case and beyond-almost-sure analysis investigated in both the single-dimensional [11] and in the multi-dimensional [17] setup.

Quantiles The decision problem related to VaR has been phrased in probabilistic verification mostly in the form *"Is the probability that the payoff is higher than a given value threshold more than a given probability threshold?"* The total reward gained attention both in the verification community [6, 24, 35] and recently in the AI community [23, 29]. Multi-dimensional percentile queries are considered for various objectives, such as mean-payoff, limsup, liminf, shortest path in [32]; for the specifics of two-dimensional case and their interplay, see [3]. Quantile queries for more complex constraints have also been considered, namely their conjunctions [9, 20], conjunctions with expectations [15] or generally Boolean expressions [25]. Some of these approaches have already been practically applied and found useful by domain experts [4, 5].

CVaR There is a body of work that optimizes CVaR in MDP. However, to the best of our knowledge, all the approaches (1) focus on the single-dimensional case, (2) disregard the expectation, and (3) treat neither reachability nor mean payoff. They focus on the discounted [7], total [13], or immediate [27] reward, as well as extend the results to continuous-time models [26, 30]. This work comes from the area of optimization and operations research, with the notable exception of [13], which focuses on the total reward. Since the total reward generalizes weighted reachability, [13] is related to our work the most. However, it provides only an approximation solution for the one-dimensional case, neglecting expectation and the respective trade-offs.

Further, CVaR is a topic of high interest in finance, e.g., [8, 33]. The central difference is that there variations of portfolios (i.e. the objective functions) are considered while leaving the underlying random process (the market) unchanged. This is dual to our problem, since we fix the objective function and now search for an optimal random process (or the respective strategy).

Multi-objective expectation In the last decade, MDP have been extensively studied generally in the setting of multiple objectives, which provides some of the necessary tools for our trade-off analysis. Multiple objectives have been considered for both qualitative payoffs, such as reachability and LTL [19], as well as quantitative payoffs, such as mean payoff [9], discounted sum [14], or total reward [22]. Variance has been introduced to the landscape in [10].

2 Preliminaries

Due to space constraints, some proofs and explanations are shortened or omitted when clear and can be found in [28].

2.1 Basic definitions

We mostly follow the definitions of [9, 15]. $\mathbb{N}, \mathbb{Q}, \mathbb{R}$ are used to denote the sets of positive integers, rational and real numbers, respectively. For $n \in \mathbb{N}$, let $[n] = \{1, \dots, n\}$. Further, k_j refers to $k \cdot e_j$, where e_j is the unit vector in dimension j.

We assume familiarity with basic notions of probability theory, e.g., *probability space* $(\Omega, \mathcal{F}, \mu)$, *random variable* F, or *expected value* $\mathbb{E}$. The set of all distributions over a countable set C is denoted by $\mathcal{D}(C)$. Further, $d \in \mathcal{D}(C)$ is Dirac if $d(c) = 1$ for some $c \in C$. To ease notation, for functions yielding a distribution over some set C, we may write $f(\cdot, c)$ instead of $f(\cdot)(c)$ for $c \in C$.

Markov chains A *Markov chain* (MC) is a tuple $\mathsf{M} = (S, \delta, \mu_0)$, where S is a countable set of states[1], $\delta : S \to \mathcal{D}(S)$ is a probabilistic transition function, and $\mu_0 \in \mathcal{D}(S)$ is the initial probability distribution. The SCCs and BSCCs of a MC are denoted by SCC and BSCC, respectively [31].

A *run* in M is an infinite sequence $\rho = s_1 s_2 \cdots$ of states, we write ρ_i to refer to the i-th state s_i. A *path* ϱ in M is a finite prefix of a run ρ. Each path ϱ in M determines the set $\mathsf{Cone}(\varrho)$ consisting of all runs that start with ϱ. To M, we associate the usual probability space $(\Omega, \mathcal{F}, \mathbb{P})$, where Ω is the set of all runs in M, $\mathcal{F}$ is the σ-field generated by all $\mathsf{Cone}(\varrho)$, and $\mathbb{P}$ is the unique probability measure such that $\mathbb{P}(\mathsf{Cone}(s_1 \cdots s_k)) = \mu_0(s_1) \cdot \prod_{i=1}^{k-1} \delta(s_i, s_{i+1})$. Furthermore, $\Diamond B$ ($\Diamond\Box B$) denotes the set of runs which eventually reach (eventually remain in) the set $B \subseteq S$, i.e. all runs where $\rho_i \in B$ for some i (there exists an i_0 such that $\rho_i \in B$ for all $i \geq i_0$).

Markov decision processes A *Markov decision process* (MDP) is a tuple $\mathcal{M} = (S, \mathsf{A}, \mathsf{Av}, \Delta, s_0)$ where S is a finite set of states, A is a finite set of actions, $\mathsf{Av} : S \to 2^{\mathsf{A}} \setminus \{\emptyset\}$ assigns to each state s the set $\mathsf{Av}(s)$ of actions enabled in s so that $\{\mathsf{Av}(s) \mid s \in S\}$ is a partitioning of A[2], $\Delta : \mathsf{A} \to \mathcal{D}(S)$ is a probabilistic transition function that given an action a yields a probability distribution over the successor states, and s_0 is the initial state of the system.

[1]We allow the state set to be countable for the formal definition of strategies on MDP. When dealing with Markov Chains in queries, we only consider finite state sets.
[2]In other words, each action is associated with exactly one state.

A *run* ρ of $\mathcal{M}$ is an infinite alternating sequence of states and actions $\rho = s_1 a_1 s_2 a_2 \cdots$ such that for all $i \geq 1$, we have $a_i \in \mathrm{Av}(s_i)$ and $\Delta(a_i, s_{i+1}) > 0$. Again, ρ_i refers to the i-th state visited by this particular run. A *path* of length k in $\mathcal{M}$ is a finite prefix $\varrho = s_1 a_1 \cdots a_{k-1} s_k$ of a run in G.

Strategies and plays. Intuitively, a strategy in an MDP $\mathcal{M}$ is a "recipe" to choose actions based on the observed events. Usually, a strategy is defined as a function $\sigma : (SA)^* S \to \mathcal{D}(A)$ that given a finite path ϱ, representing the history of a play, gives a probability distribution over the actions enabled in the last state. We adopt the slightly different, though equivalent [9, Sec. 6] definition from [15], which is more convenient for our setting.

Let M be a countable set of *memory elements*. A *strategy* is a triple $\sigma = (\sigma_u, \sigma_n, \alpha)$, where $\sigma_u : \mathsf{A} \times S \times \mathsf{M} \to \mathcal{D}(\mathsf{M})$ and $\sigma_n : S \times \mathsf{M} \to \mathcal{D}(\mathsf{A})$ are *memory update* and *next move* functions, respectively, and $\alpha \in \mathcal{D}(\mathsf{M})$ is the initial memory distribution. We require that, for all $(s, m) \in S \times \mathsf{M}$, the distribution $\sigma_n(s, m)$ assigns positive values only to actions available at s, i.e. $\mathrm{supp}\, \sigma_n(s, m) \subseteq \mathrm{Av}(s)$.

A *play* of $\mathcal{M}$ determined by a strategy σ is a Markov chain $\mathcal{M}^\sigma = (S^\sigma, \delta^\sigma, \mu_0^\sigma)$, where the set of states is $S^\sigma = S \times \mathsf{M} \times \mathsf{A}$, the initial distribution μ_0 is zero except for $\mu_0^\sigma(s_0, m, a) = \alpha(m) \cdot \sigma_n(s_0, m, a)$, and the transition probability from $s^\sigma = (s, m, a)$ to $s'^\sigma = (s', m', a')$ is $\delta^\sigma(s^\sigma, s'^\sigma) = \Delta(a, s') \cdot \sigma_u(a, s', m, m') \cdot \sigma_n(s', m', a')$. Hence, $\mathcal{M}^\sigma$ starts in a location chosen randomly according to α and σ_n. In state (s, m, a) the next action to be performed is a, hence the probability of entering s' is $\Delta(a, s')$. The probability of updating the memory to m' is $\sigma_u(a, s', m, m')$, and the probability of selecting a' as the next action is $\sigma_n(s', m', a')$. Since these choices are independent, and thus we obtain the product above.

Technically, $\mathcal{M}^\sigma$ induces a probability measure $\mathbb{P}^\sigma$ on S^σ. Since we mostly work with the corresponding runs in the original MDP, we overload $\mathbb{P}^\sigma$ to also refer to the probability measure obtained by projecting onto S. Further, "almost surely" etc. refers to happening with probability 1 according to $\mathbb{P}^\sigma$. The expected value of a random variable $X : \Omega \to \mathbb{R}$ is $\mathbb{E}^\sigma[X] = \int_\Omega X \, d\mathbb{P}^\sigma$.

A convex combinations of two strategies σ_1 and σ_2, written as $\sigma_\lambda = \lambda\sigma_1 + (1 - \lambda)\sigma_2$, can be obtained by defining the memory as $\mathsf{M}_\lambda = \{1\} \times \mathsf{M}_1 \cup \{2\} \times \mathsf{M}_2$, randomly choosing one of the two strategies via the initial memory distribution α_λ and then following the chosen strategy. Clearly, we have that $\mathbb{P}^{\sigma_\lambda} = \lambda\mathbb{P}^{\sigma_1} + (1 - \lambda)\mathbb{P}^{\sigma_2}$.

Strategy types. A strategy σ may use infinite memory M, and both σ_u and σ_n may randomize. The strategy σ is

- *deterministic-update*, if α is Dirac and the memory update function σ_u gives a Dirac distribution for every argument;
- *deterministic*, if it is deterministic-update and the next move function σ_n gives a Dirac distribution for every argument.

A *stochastic-update* strategy is a strategy that is not necessarily deterministic-update and *randomized* strategy is a strategy that is not necessarily deterministic. We also classify the strategies according to the size of memory they use. Important subclasses are *memoryless* strategies, in which M is a singleton, *n-memory* strategies, in which M has exactly n elements, and *finite-memory* strategies, in which M is finite.

End components. A tuple (T, B) where $\emptyset \neq T \subseteq S$ and $\emptyset \neq B \subseteq \bigcup_{t \in T} \mathrm{Av}(t)$ is an *end component* of the MDP $\mathcal{M}$ if (i) for all actions $a \in B$, $\Delta(a, s') > 0$ implies $s' \in T$; and (ii) for all states $s, t \in T$ there is a path $\varrho = s_1 a_1 \cdots a_{k-1} s_k \in (TB)^{k-1} T$ with $s_1 = s$, $s_k = t$. An end component (T, B) is a *maximal end component (MEC)* if T and B are maximal with respect to subset ordering. Given an MDP, the set of MECs is denoted by MEC. By abuse of notation, $s \in M$ refers to all states of a MEC M, while $a \in M$ refers to the actions.

Remark 1. *Computing the maximal end component (MEC) decomposition of an MDP, i.e. the computation of* MEC*, is in P [18].*

Remark 2. *For any MDP $\mathcal{M}$ and strategy σ, a run almost surely eventually stays in one MEC, i.e. $\mathbb{P}^\sigma[\bigcup_{M_i \in \mathrm{MEC}} \Diamond\Box M_i] = 1$ [31].*

2.2 Random variables on Runs

We introduce two standard random variables, assigning a value to each run of a Markov Chain or Markov Decision Process.

Weighted reachability. Let $T \subseteq S$ be a set of target states and $\mathbf{r} : T \mapsto \mathbb{Q}$ be a reward function. Define the random variable R^{r} as $R^{\mathrm{r}}(\rho) = \mathbf{r}(\min_i\{\rho_i \mid \rho_i \in T\})$, if such an i exists, and 0 otherwise. Informally, R^{r} assigns to each run the value of the first visited target state, or 0 if none. R^{r} is measurable and discrete, as S is finite [31]. Whenever we are dealing with weighted reachability, we assume w.l.o.g. that all target states are absorbing, i.e. for any $s \in T$ we have $\delta(s, s) = 1$ for MC and $\Delta(a, s) = 1$ for all $a \in \mathrm{Av}(s)$ for MDP.

Mean payoff (also known as *long-run average reward*, and *limit average reward*). Again, let $\mathbf{r} : S \mapsto \mathbb{Q}$ be a reward function. The mean payoff of a run ρ is the average reward obtained per step, i.e. $R^{\mathrm{m}}(\rho) = \liminf_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} \mathbf{r}(\rho_i)$. The lim inf is necessary, since lim may not be defined in general. Further, R^{m} is measurable [31].

Remark 3. *There are several distinct definitions of "weighted reachability". The one chosen here primarily serves as foundation for the more general mean payoff.*

3 Introducing the Conditional Value-at-risk

In order to define our problem, we first introduce the general concept of *conditional value-at-risk* (CVaR), also known as *average value-at-risk*, *expected shortfall*, and *expected tail loss*. As already hinted, the CVaR of some real-valued random variable X and probability $p \in [0, 1]$ intuitively is the expectation below the worst p-quantile of X.

Let $X : \Omega \to \mathbb{R}$ be a random variable over the probability space $(\Omega, \mathcal{F}, \mathbb{P})$. The associated *cumulative density function* (CDF) $F_X : \mathbb{R} \to [0, 1]$ of X yields the probability of X being less than or equal to the given value r, i.e. $F_X(r) = \mathbb{P}(\{X(\omega) \leq r\})$. F is non-decreasing and right continuous with left limits (*càdlàg*).

The *value-at-risk* VaR_p is the worst p-quantile, i.e. a value v s.t. the probability of X attaining a value less than or equal to v is p:[3]

$$\mathrm{VaR}_p(X) := \sup\{r \in \mathbb{R} \mid F_X(r) \leq p\} \quad (\mathrm{VaR}_1(X) = \infty)$$

Then, with $v = \mathrm{VaR}_p(X)$, CVaR can be defined as [33]

$$\mathrm{CVaR}_p(X) := \mathbb{E}[X \mid X \leq v] = \frac{1}{p} \int_{(-\infty, v]} x \, dF_X,$$

with the corner cases $\mathrm{CVaR}_0 := \mathrm{VaR}_0$ and $\mathrm{CVaR}_1 = \mathbb{E}$.

Unfortunately, this definition only works as intended for continuous X, as shown by the following example.

[3]An often used, mostly equivalent definition is $\inf\{r \in \mathbb{R} \mid F_X(r) \geq p\}$. Unfortunately, this would lead to some complications later on. See [28, Sec. A.1] for details.

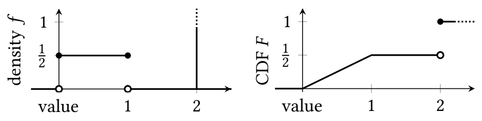


Figure 2. Distribution showing peculiarities of CVaR

Example 3.1. Consider a random variable X with a distribution as outlined in Fig. 2. For $p < \frac{1}{2}$, we certainly have $\mathrm{VaR}_p = 2p$. On the other hand, for any $p \in (\frac{1}{2}, 1)$, we get $\mathrm{VaR}_p = 2$. Consequently, the integral remains constant and CVaR_p would actually *decrease* for increasing p, not matching the intuition. △

General definition. As seen in Ex. 3.1, the previous definition breaks down when F_X is not continuous at the p-quantile and consequently $F_X(\mathrm{VaR}_p(X)) > p$. Thus, we handle the values at the threshold separately, similar to [34].

Definition 3.2. Let X be some random variable and $p \in [0, 1]$. With $v = \mathrm{VaR}_p(X)$, the CVaR of X is defined as

$$\mathrm{CVaR}_p(X) := \frac{1}{p}\left(\int_{(-\infty,v)} x\, dF_X + (p - \mathbb{P}[X < v]) \cdot v\right),$$

which can be rewritten as

$$\mathrm{CVaR}_p(X) = \tfrac{1}{p}\big(\mathbb{P}[X < v] \cdot \mathbb{E}[X \mid X < v] + (p - \mathbb{P}[X < v]) \cdot v\big).$$

The corner cases again are $\mathrm{CVaR}_0 := \mathrm{VaR}_0$, and $\mathrm{CVaR}_1 = \mathbb{E}$.

Since the degenerate cases of $p = 0$ and $p = 1$ reduce to already known problems, we exclude them in the following.

We demonstrate this definition on the previous example.

Example 3.3. Again, consider the random variable X from Ex. 3.1. For $\frac{1}{2} < p < 1$ we have that $\mathbb{P}[X < \mathrm{VaR}_p(X)] = \mathbb{P}[X < 2] = \frac{1}{2}$. The right hand side of the definition $(p - \mathbb{P}[X < \mathrm{VaR}_p(X)]) = p - \frac{1}{2}$ captures the remaining discrete probability mass which we have to handle separately. Together with $\int_{(-\infty,2)} x\, dF_X = \frac{1}{4}$ we get $\mathrm{CVaR}_p(X) = \frac{1}{p}(\frac{1}{4} + (p - \frac{1}{2}) \cdot 2) = 2 - \frac{3}{4p}$. For example, with $p = \frac{3}{4}$, this yields the expected result $\mathrm{CVaR}_p(X) = 1$. △

Remark 4. *Recall that* $\mathbb{P}[X < r]$ *can be expressed as the left limit of* F_X*, namely* $\mathbb{P}[X < r] = \lim_{r' \to^- r} F_X(r')$*. Hence,* $\mathrm{CVaR}_p(X)$ *solely depends on the CDF of* X *and thus random variables with the same CDF also have the same CVaR.*

We say that F_1 *stochastically dominates* F_2 for two CDF F_1 and F_2, if $F_1(r) \leq F_2(r)$ for all r. Intuitively, this means that a sample drawn from F_2 is likely to be larger or equal to a sample from F_1. All three investigated operators ($\mathbb{E}$, CVaR, and VaR) are monotone w.r.t. stochastic dominance [28, Sec. A.1].

4 CVaR in MC and MDP: Problem statement

Now, we are ready to define our problem framework. First, we explain the types of building blocks for our queries, namely lower bounds on expectation, CVaR, and VaR. Formally, we consider the following types of constraints.

$$\mathsf{e} \leq \mathbb{E}(X) \qquad \mathsf{c} \leq \mathrm{CVaR}_{\mathsf{p}}(X) \qquad \mathsf{v} \leq \mathrm{VaR}_{\mathsf{q}}(X)$$

X is some real-valued random variable, assigning a payoff to each run. With these constraints, the classes of queries are denoted by

$$\mathsf{MDP}^{\mathbf{crit}}_{\mathbf{obj},\mathbf{dim}}$$

- $\mathbf{crit} \subseteq \{\mathbb{E}, \mathrm{CVaR}, \mathrm{VaR}\}$ are the types of constraints,
- $\mathbf{obj} \in \{\mathsf{r}, \mathsf{m}\}$ is the type of the objective function, either weighted reachability r or mean payoff m, and
- $\mathbf{dim} \in \{\mathsf{single}, \mathsf{multi}\}$ is the dimensionality of the query.

We use d to denote the dimensions of the problem, $d = 1$ iff $\mathbf{dim} = \mathsf{single}$. As usual, we assume that all quantities of the input, e.g., probabilities of distributions, are rational.

An instance of these queries is specified by an MDP $\mathcal{M}$, a d-dimensional reward function $\mathbf{r} : S \to \mathbb{Q}^d$, and constraints from $\mathbf{crit}$, given by vectors $\mathbf{e}, \mathbf{c}, \mathbf{v} \in (\mathbb{Q} \cup \{\bot\})^d$ and $\mathbf{p}, \mathbf{q} \in (0, 1)^d$. This implies that in each dimension there is at most one constraint per type. The presented methods can easily be extended to the more general setting of multiple constraints of a particular type in one dimension. The decision problem is to determine whether there exists a strategy σ such that *all* constraints are met.

Technically, this is defined as follows. Let X be the d-dimensional random variable induced by the objective $\mathbf{obj}$ and reward function $\mathbf{r}$, operating on the probability space of $\mathcal{M}^\sigma$. The strategy σ is a witness to the query iff for each dimension $j \in [d]$ we have that $\mathbb{E}[X_j] \geq \mathbf{e}_j$, $\mathrm{CVaR}_{\mathbf{p}_j}(X_j) \geq \mathbf{c}_j$, and $\mathrm{VaR}_{\mathbf{q}_j}(X_j) \geq \mathbf{v}_j$. Moreover, $\bot$ constraints are trivially satisfied.

For completeness sake, we also consider $\mathsf{MC}^{\mathbf{crit}}_{\mathbf{obj},\mathbf{dim}}$ queries, i.e. the corresponding problem on (finite state) Markov chains.

Notation. We introduce the following abbreviations. When dealing with an MDP $\mathcal{M}$, CVaR^σ_p denotes CVaR_p relative to the probability space over runs induced by the strategy σ. When additionally the random variable X (e.g., mean payoff) is clear from the context, we may write CVaR_p and CVaR^σ_p instead of $\mathrm{CVaR}_p(X)$ and $\mathrm{CVaR}^\sigma_p(X)$, respectively. We also define analogous abbreviations for VaR.

5 Single dimension

We show that all queries in one dimension are in P. Furthermore, our LP-based decision procedures directly yield a description of a witness strategy and allow for optimization objectives. We refer to the input constraints by e for expectation, (p, c) for CVaR, and (q, v) for VaR. Further, we use i for indices related to SCCs / MECs.

5.1 Weighted reachability

First, we show the simple result for Markov Chains, providing some insight in the techniques used in the MDP case.

Theorem 5.1. $\mathsf{MC}^{\{\mathbb{E},\mathrm{CVaR},\mathrm{VaR}\}}_{\mathsf{r},\mathsf{single}}$ *is in P.*

Proof. Let M be a *finite-state* Markov chain, $\mathbf{r}$ a reward function, and $T = \{b_1, \ldots, b_n\}$ the target set. Recall that all b_i are absorbing, hence single-state BSCCs. We obtain the stationary distribution p of M' in polynomial time by, e.g., solving a linear equation system [31]. With p, we can directly compute the CDF of R^{r} as $F_{R^{\mathsf{r}}}(v) = \sum_{b_i : \mathbf{r}(b_i) \leq v} p(b_i)$ and immediately decide the query. □

Let us consider the more complex case of MDP. We show a lower bound on the type of strategies necessary to realize $\mathbf{obj} = \mathsf{r}$ queries with constraints on expectation and one of VaR or CVaR. We then continue to prove that this class of strategies is optimal. This characterization is used to derive a polynomial time decision procedure based on a linear program (LP) which immediately yields a witness strategy. Finally, when we deal with the mean payoff case in Sec. 5.2, we make use of the reasoning presented in this section.

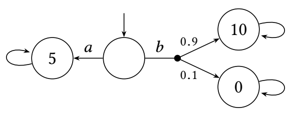


Figure 3. MDP used to show various difficulties of CVaR

Randomization is necessary for weighted reachability. In the following example, we present a simple MDP on which all deterministic strategies fail to satisfy specific constraints, while a straightforward randomizing one succeeds in doing so.

Example 5.2. Consider the MDP outlined in Fig. 3. The only non-determinism is given by the choice in the initial state s_0. Hence, any strategy is characterised by the choice in that particular state. Let now σ_a and σ_b denote the deterministic strategies playing a and b in s_0, respectively. Clearly, σ_a achieves an expectation, $\mathrm{CVaR}_{0.05}^{\sigma_a}$, and $\mathrm{VaR}_{0.05}^{\sigma_a}$ of 5. On the other hand, σ_b obtains an expectation of 9 with $\mathrm{CVaR}_{0.05}^{\sigma_b}$ and $\mathrm{VaR}_{0.05}^{\sigma_b}$ equal to 0.

Thus, neither strategy satisfies the constraints $\mathrm{q} = \mathrm{p} = 0.05$, $\mathrm{e} = 6$, and $\mathrm{c} = 2$ (or $\mathrm{v} = 5$). This is the case even when the strategy has arbitrary (deterministic) memory at its disposal, since in the first step there is nothing to remember. Yet, $\sigma = \frac{3}{4}\sigma_a + \frac{1}{4}\sigma_b$ achieves $\mathbb{E} = \frac{3}{4}5 + \frac{1}{4}9 = 6 \geq \mathrm{e}$, $\mathrm{CVaR}_\mathrm{p} = 2.5 \geq \mathrm{c}$, and $\mathrm{VaR}_\mathrm{q} = 5 \geq \mathrm{v}$. △

Hence strategies satisfying an expectation constraint together with either a CVaR *or* VaR constraint may necessarily involve randomization in general. We prove that (i) under mild assumptions randomization actually is sufficient, i.e. no memory is required, and (ii) fixed memory may additionally be required in general.

Definition 5.3. Let $\mathcal{M}$ be an MDP with target set T and reward function $\mathbf{r}$. We say that $\mathcal{M}$ satisfies the *attraction assumption* if **A1**) the target set T is reached almost surely for any strategy, or **A2**) for all target state $s \in T$ we have $\mathbf{r}(s) \geq 0$.

Essentially, this definition implies that an optimal strategy never remains in a non-target MEC. This allows us to design memoryless strategies for the weighted reachability problem.

Theorem 5.4. *Memoryless randomizing strategies are sufficient for* $\mathsf{MDP}_{\mathbf{r},\mathrm{single}}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}$ *under the attraction assumption.*

Proof. Fix an MDP $\mathcal{M}$ and reward function $\mathbf{r}$. We prove that for any strategy σ there exists a memoryless, randomizing strategy σ' achieving at least the expectation, VaR, and CVaR of σ.

All target states $t_i \in T$ form single-state MECs, as we assumed that all target states are absorbing. Consequently, σ naturally induces a distribution over these s_i. Now, we apply [19, Theorem 3.2] to obtain a strategy σ' with $\mathbb{P}^{\sigma'}[\Diamond s_i] \geq \mathbb{P}^{\sigma}[\Diamond s_i]$ for all i.

With **A1**), we have $\sum p_i = 1$ and thus $\mathbb{P}^{\sigma'}[\Diamond t_i] = \mathbb{P}^{\sigma}[\Diamond t_i]$. Hence, σ' obtains the same CDF for the weighted reachability objective. Under **A2**), the CDF F' of strategy σ' stochastically dominates the CDF F of the original strategy σ, concluding the proof. □

Theorem 5.5. *Two-memory stochastic strategies (i.e. with both randomization and stochastic update) are sufficient for* $\mathsf{MDP}_{\mathbf{r},\mathrm{single}}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}$.

The proof is a simple application of the following Thm. 5.10, as weighted reachability is a special case of mean payoff. Together with an example for the lower bound it can be found in [28, Sec. A.2].

(1) All variables y_a, x_s, $\underline{x}_s$ are non-negative.

(2) Transient flow for $s \in S$:

$$\mathbb{1}_{s_0}(s) + \sum\nolimits_{a \in \mathrm{A}} y_a \Delta(a, s) = \sum\nolimits_{a \in \mathrm{Av}(s)} y_a + x_s$$

(3) Switching to recurrent behaviour:

$$\sum\nolimits_{s \in T} x_s = 1$$

(4) VaR-consistent split:

$$\underline{x}_s = x_s \text{ for } s \in T_< \qquad \underline{x}_s \leq x_s \text{ for } s \in T_=$$

(5) Probability-consistent split:

$$\sum\nolimits_{s \in T_\leq} \underline{x}_s = \mathrm{p}$$

(6) CVaR and expectation satisfaction:

$$\sum\nolimits_{s \in T_\leq} \underline{x}_s \cdot \mathbf{r}(s) \geq \mathrm{p} \cdot \mathrm{c} \qquad \sum\nolimits_{s \in T} x_s \cdot \mathbf{r}(s) \geq \mathrm{e}$$

Figure 4. LP used to decide weighted reachability queries given a guess t of VaR_p. $T_\sim := \{s \in T \mid \mathbf{r}(s) \sim t\}$, $\sim \in \{<, =, \leq\}$.

Inspired by [15, Fig. 3], we use the optimality result from Thm. 5.4 to derive a decision procedure for weighted reachability queries under the attraction assumptions based on the LP in Fig. 4.

To simplify the LP, we make further assumptions – see [28, Sec. A.2] for details. First, all MECs, including non-target ones, consist of a single state. Second, all MECs from which T is not reachable are considered part of T and have $\mathbf{r} = 0$ (similar to the "cleaned-up MDP" from [19]). Finally, we assume that the quantile-probabilities are equal, i.e. $\mathrm{p} = \mathrm{q}$. The LP can easily be extended to account for different values by duplicating the $\underline{x}_s$ variables and adding according constraints.

The central idea is to characterize randomizing strategies by the "flow" they achieve. To this end, Equality (2) essentially models Kirchhoff's law, i.e. inflow and outflow of a state have to be equal. In particular, y_a expresses the transient flow of the strategy as the expected total number of uses of action a. Similarly, x_s models the recurrent flow, which under our absorption assumption equals the probability of reaching s. Equality (3) ensures that all transient behaviour eventually changes into recurrent one.

In order to deal with our query constraints, Constraints (4) and (5) extract the worst p fraction of the recurrent flow, ensuring that the VaR_p is at least t. Note that equality is not guaranteed by the LP; if $\underline{x}_s = x_s$ for all $s \in T_\leq$, we have $\mathrm{VaR}_\mathrm{p} > t$. Finally, Inequality (6) enforces satisfaction of the constraints.

Theorem 5.6. *Let $\mathcal{M}$ be an MDP with target states T and reward function* $\mathbf{r}$, *satisfying the attraction assumption. Fix the constraint probability* $\mathrm{p} \in (0, 1)$ *and thresholds* $\mathrm{e}, \mathrm{c} \in \mathbb{Q}$*. Then, we have that*

1. *for any strategy σ satisfying the constraints, there is a $t \in \mathbf{r}(S)$ such that the LP in Fig. 4 is feasible, and*
2. *for any threshold $t \in \mathbf{r}(S)$, a solution of the LP in Fig. 4 induces a memoryless, randomizing strategy σ satisfying the constraints and* $\mathrm{VaR}_\mathrm{p}^\sigma \geq t$.

Proof. First, we prove for a strategy σ satisfying the constraints that there exists a $t \in \mathbf{r}(S)$ such that the LP is feasible. By Thm. 5.4, we may assume that σ is a memoryless randomizing strategy. From [19, Theorem 3.2], we get an assignment to the y_a's and x_s's satisfying Equalities (1), (2), and (3) such that $\mathbb{P}^\sigma[\Diamond s] = x_s$ for all target states

$s \in T$. Further, let $v = \mathrm{VaR}_{\mathrm{p}}^{\sigma}$ be the value-at-risk of the strategy. By definition of VaR, we have that $\mathbb{P}^{\sigma}[X < v] \leq p$.

Assume for now that $\mathbb{P}^{\sigma}[X < v] = p$, i.e. the probability of obtaining a value strictly smaller than v is exactly p. In this case, choose t to be the next smaller reward, i.e. $t = \max\{\mathbf{r}(s) < v\}$. We set $\underline{x}_s = x_s$ for all $s \in T_{\leq}$, satisfying Constraints (4) and (5).

Otherwise, we have $\mathbb{P}^{\sigma}[X < v] < p$. Now, some non-zero fraction of the probability mass at v contributes to the CVaR. Again, we set the values for $\underline{x}_s$ according to Constraint (4). The only degree of freedom are the values of $\underline{x}_s$ where $\mathbf{r}(s) = t$. There, we assign the values so that $\sum_{s \in T_{=}} \underline{x}_s = p - \sum_{s \in T_{<}} \underline{x}_s$, satisfying Equality (5).

It remains to check Inequality (6). For expectation, we have $\sum_{s \in T} x_s \cdot \mathbf{r}(s) = \sum_{s \in T} \mathbb{P}^{\sigma}[\Diamond s] \cdot \mathbf{r}(s) = \mathbb{E}^{\sigma}[R^{\mathrm{r}}] \geq \mathrm{e}$. For CVaR, notice that, due to the already proven Constraints (4) and (5), the side of Inequality (6) is equal to $\mathrm{CVaR}_{\mathrm{p}}^{\sigma}$ and thus at least c.

Second, we prove that a solution to the LP induces the desired strategy σ. Again by [19, Theorem 3.2], we get a memoryless randomizing strategy σ such that $\mathbb{P}^{\sigma}[\Diamond s] = x_s$ for all states $s \in T$. Then $\mathbb{E}^{\sigma}[R^{\mathrm{r}}] = \sum_{s \in T} \mathbb{P}^{\sigma}[\Diamond s] \cdot \mathbf{r}(s) = \sum_{s \in T} x_s \cdot \mathbf{r}(s) \geq \mathrm{e}$. Further,

$$\mathrm{CVaR}_p(R^{\mathrm{r}}) = \frac{1}{\mathrm{p}} \left(\sum\nolimits_{s:\mathbf{r}(s)<v} x_s \cdot \mathbf{r}(s) + (\mathrm{p} - \sum\nolimits_{s:\mathbf{r}(s)<v} x_s) \cdot v \right)$$

by definition. Now, we make a case distinction on $\underline{x}_s = x_s$ for all $s \in T_{=}$. If this is true, we have $v = \mathrm{VaR}_{\mathrm{p}}^{\sigma} = \min\{r \in \mathbf{r}(S) \mid r > t\}$, but $\mathbb{P}^{\sigma}[X < v] = p$. Consequently, $T_{\leq} = \{s \in T : \mathbf{r}(s) < v\}$ and $\sum_{s:\mathbf{r}(s)<v} x_s = p$. Otherwise, we have $v = t$ and consequently $T_{<} = \{s \mid \mathbf{r}(s) < v\}$. Inserting in the above equation immediately gives the result $\mathrm{CVaR}_p(R^{\mathrm{r}}) = \frac{1}{\mathrm{p}} \sum_{s \in T_{\leq}} \mathbf{r}(s) \cdot \underline{x}_s$. □

The linear program requires to know the VaR_p^{σ} beforehand, which in turn clearly depends on the chosen strategy. Yet, there are only linearly many values the random variable R^{r} attains. Thus we can simply try to find a solution for all potential values of VaR_p^{σ}, i.e. $\{r \in \mathbf{r}(S)\}$, yielding a polynomial time solution.

Corollary 5.7. $\mathrm{MDP}_{\mathrm{r,single}}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}$ *is in P.*

Proof. Under the attraction assumption, this follows directly from Thm. 5.6. In general, the reduction to mean payoff used by Thm. 5.5 and the respective result from Cor. 5.11 show the result. □

5.2 Mean payoff

In this section, we investigate the case of $\mathbf{obj} = \mathrm{m}$. Again, the construction for MC is considerably simple, yet instructive for the following MDP case.

Theorem 5.8. $\mathrm{MC}_{\mathrm{m,single}}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}$ *is in P.*

Proof sketch. For each BSCC B_i, we obtain its expected mean payoff $r_i = \mathbb{E}[R^{\mathrm{m}} \mid B_i]$ through, e.g., a linear equation system [31]. Almost all runs in B_i achieve this mean payoff and thus the corresponding random variable is discrete. We reduce the problem to weighted reachability by using the known reformulation

$$\mathbb{P}[R^{\mathrm{m}} = c] = \sum\nolimits_{B_i : r_i = c} P[\Diamond B_i].$$

We replace each of these BSCCs by a representative b_i to obtain M'. Define the set of target states $T = \{b_i\}$ and the reachability reward function $\mathbf{r}'(b_i) = r_i$. By applying the approach of Thm. 5.1, we obtain the expectation, VaR, and CVaR for reachability in M' which by construction coincides with the respective values for mean payoff in M. □

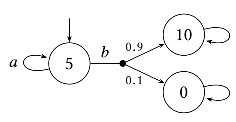


Figure 5. Memory is necessary for mean payoff queries

For the MDP case, recall that simple expectation maximization of mean payoff can be reduced to weighted reachability [2] and deterministic, memoryless strategies are optimal [31]. Yet, solving a conjunctive query involving either VaR or CVaR needs more powerful strategies than in the weighted reachability case of Thm. 5.4. Nevertheless, we show how to decide these queries in P.

Randomization and memory is necessary for mean payoff. A simple modification of the MDP in Fig. 3 yields an MDP where both randomization and memory is required to satisfy the constraints of the following example.

Example 5.9. Consider the MDP presented in Fig. 5. There, the same constraints as before, i.e. $\mathrm{q} = \mathrm{p} = 0.05$, $\mathrm{e} = 6$, and $\mathrm{c} = 2$ (or $\mathrm{v} = 5$), can only be satisfied by strategies with both memory and randomization. Clearly, a pure strategy can only satisfy either of the two constraints again. But now a memoryless randomizing strategy also is insufficient, too, since any non-zero probability on action b leads to almost all runs ending up on the right side of the MDP, hence yielding a $\mathrm{CVaR}_{\mathrm{p}}$ and $\mathrm{VaR}_{\mathrm{q}}$ of 0. Instead, a stochastic strategy with $\mathrm{M} = \{a, b\}$ can simply choose $\alpha = \{a \mapsto \frac{3}{4}, b \mapsto \frac{1}{4}\}$ and play the corresponding action indefinitely, satisfying the constraints. △

We prove that this bound actually is tight, i.e. that, given stochastic memory update, two memory elements are sufficient.

Theorem 5.10. *Two-memory stochastic strategies (i.e. with both randomization and stochastic update) are sufficient for* $\mathrm{MDP}_{\mathrm{m,single}}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}$.

Proof. Let σ be a strategy on an MDP $\mathcal{M}$ with reward function $\mathbf{r}$. We construct a two-memory stochastic strategy σ' achieving at least the expectation, VaR, and CVaR of σ.

First, we obtain a memoryless deterministic strategy σ_{opt} which obtains the maximal possible mean payoff in each MEC [31]. We then apply the construction of [9, Proposition 4.2] (see also [15, Lemma 5.7]), where the ξ is our σ_{opt}. (Technically, this can be ensured by choosing the constraints of the LP L according to σ_{opt}.)

Intuitively, this constructs a two-memory strategy σ' on $\mathcal{M}$ behaving as follows. Initially, σ' remains in each MEC with the same probability as σ, i.e. $\mathbb{P}^{\sigma'}[\Diamond\Box M_i] = \mathbb{P}^{\sigma}[\Diamond\Box M_i]$ by following a memoryless "searching" strategy and stochastically switching its memory state to "remain". Once in the "remain" state, the behaviour of the optimal strategy σ_{opt} is implemented.

Clearly, (i) both strategies remain in a particular MEC with the same probability, and (ii) σ' obtains as least as much value in each MEC as σ. Hence the CDF induced by σ' stochastically dominates the one of σ, concluding the proof. □

This immediately gives us a polynomial time decision procedure.

Corollary 5.11. $\mathrm{MDP}_{\mathrm{m,single}}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}$ *is in P.*

Furthermore, we can use results of [15, Lemma 16] to trade the stochastic update for more memory.

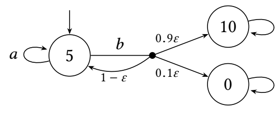


Figure 6. Exponential memory is necessary for mean payoff when only deterministic update is allowed.

Corollary 5.12. *Stochastic strategies with finite, deterministic memory are sufficient for* $\mathsf{MDP}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}_{\mathsf{m},\mathsf{single}}$.

Deterministic strategies may require exponential memory. As sources of randomness are not always available, one might ask what can be hoped for when only determinism is allowed. As already shown in Ex. 5.2, randomization is required in general. But even if some deterministic strategy is sufficient, it may require memory exponential in the size of the input, even in an MDP with only 3 states. We show this in the following example.

Example 5.13. Consider the MDP outlined in Fig. 6 together with the constraints $\mathsf{q} = \mathsf{p} = 0.05$, $\mathsf{e} = 6$, and $\mathsf{c} = 2$ (or $\mathsf{v} = 5$). Again, any optimal strategy needs a significant part of runs to go to the right side in order to satisfy the expectation constraint. Yet, any strategy can only "move" a small fraction of the runs there in each step. In particular, after k steps, the right side is only reached with probability at most $1 - (1 - \varepsilon)^k$. When choosing $\varepsilon = 2^{-n}$, which needs $\Theta(n)$ bits to encode, a deterministic strategy requires $k \geq c/\log(1-2^{-n}) \in O(2^n)$ memory elements to count the number of steps. The same holds true for any deterministic-update strategy.

On the other hand, a strategy with stochastic memory update can encode this counting by switching its state with a small probability after each step. For example, a strategy switching with probability $p = 3\varepsilon$ from "play b" to "play a" satisfies the constraint. △

5.3 Single constraint queries

In this section, we discuss an important sub-case of the single-dimensional case, namely queries with only a single constraint, i.e. $|\mathbf{crit}| = 1$. We show that deterministic memoryless strategies are sufficient in this case.

One might be tempted to use standard arguments and directly conclude this from the results of Thm. 5.4 as follows. Recall that this theorem shows that memoryless, randomizing strategies are sufficient; and that any such strategy can be written as finite convex combination of memoryless, deterministic strategies. Most constraints, for example expectation or reachability, behave linearly under convex combination of strategies, e.g., $\mathbb{E}^{\sigma_\lambda}(X) = \lambda\mathbb{E}^{\sigma_1}[X] + (1-\lambda)\mathbb{E}^{\sigma_2}[X]$. Consequently, for an optimal memoryless strategy, there is a deterministic witness, which in turn also is optimal.

Surprisingly, this assumption is not true for CVaR. On the contrary, the CVaR of a convex combination of strategies might be strictly worse than the CVaRs of either strategy, as shown in the following example. We prove a slightly weaker property of CVaR which eventually allows us to apply similar reasoning.

Example 5.14. Recall the MDP in Fig. 3 and let $p = 0.05$. As previously shown, $\mathrm{CVaR}_p^{\sigma_a} = 5$ and $\mathrm{CVaR}_p^{\sigma_b} = 0$, but the mixed strategy $\sigma_\lambda = \frac{1}{2}\sigma_a + \frac{1}{2}\sigma_b$ achieves $\mathrm{CVaR}_p^{\sigma_\lambda} = 0$ instead of the convex combination $\frac{1}{2}5 + \frac{1}{2}0 = 2.5$.

For $p = 0.2$, we have $\mathrm{CVaR}_p^{\sigma_a} = \mathrm{CVaR}_p^{\sigma_b} = 5$. Yet, *any* non-trivial convex combination of the two strategies yields a CVaR_p less than 5. See [28, Sec. A.1] for more details. With according constraints, this effectively can force an optimal strategy to choose between a or b. This observation is further exploited in the NP-hardness proof of the multi-dimensional case in Sec. 6. △

Since CVaR considers the worst events, the CVaR of a combination intuitively cannot be better than the combination of the respective CVaRs. We prove this intuition in the general setting, where instead of a convex combination of strategies we consider a mixture of two random variables.

Lemma 5.15. $\mathrm{CVaR}_p(X)$ *is convex in* X *for fixed* $p \in (0,1)$, *i.e. for random variables* X_1, X_2 *and* $\lambda \in [0,1]$

$$\mathrm{CVaR}_p(\lambda X_1 + (1-\lambda)X_2) \leq \lambda\,\mathrm{CVaR}_p(X_1) + (1-\lambda)\,\mathrm{CVaR}_p(X_2).$$

The proof can be found in [28, Sec. A.1]. This result allows us to apply the ideas outlined in the beginning of the section.

Theorem 5.16. *For any* $\mathbf{obj} \in \{\mathsf{r},\mathsf{m}\}$, *deterministic memoryless strategies are sufficient for* $\mathsf{MDP}^{\mathbf{crit}}_{\mathbf{obj},\mathsf{single}}$ *when* $|\mathbf{crit}| = 1$.

Proof. This is known for $\mathbf{crit} = \{\mathbb{E}\}$ [31] and $\mathbf{crit} = \{\mathrm{VaR}\}$ [21].

For CVaR, observe that the convex combination of deterministic strategies cannot achieve a better CVaR than the best strategy involved in the combination (see Lem. 5.15). This immediately yields the result for $\mathbf{obj} = \mathsf{r}$ through Thm. 5.4. For $\mathbf{obj} = \mathsf{m}$, we exploit the approach of Thm. 5.10. Recall that there we obtained a two-memory strategy σ'. Both randomization and stochastic update are used solely to distribute the runs over all MECs accordingly. By the above reasoning, for each MEC it is sufficient to either almost surely remain there or leave it. This behaviour can be implemented by a deterministic memoryless strategy on the original MDP. □

6 Multiple Dimensions

In this section, we deal with multi-dimensional queries. We continue to use i for indices related to MECs and further use j for dimension indices. First, we show that the Markov Chain case does not significantly change.

Theorem 6.1. *For any* $\mathbf{obj} \in \{\mathsf{r},\mathsf{m}\}$, $\mathsf{MC}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}_{\mathbf{obj},\mathsf{multi}}$ *is in P.*

Proof. Similarly to the single-dimensional case, we decide each constraint in each dimension separately, using our previous results. The query is satisfied iff each of the constraints is satisfied. □

6.1 NP-Hardness of reachability and mean payoff

For the MDP on the other hand, multiple dimensions add significant complexity. In the following, we show that already the weighted reachability problem with multiple dimensions and only CVaR constraints, i.e. $\mathsf{MDP}^{\{\mathrm{CVaR}\}}_{\mathsf{r},\mathsf{multi}}$, is NP-hard. This result directly transfers to mean payoff, i.e. $\mathbf{obj} = \mathsf{m}$. Recall that in contrast $\mathsf{MDP}^{\{\mathbb{E}\}}_{\mathsf{r},\mathsf{multi}}$ and even $\mathsf{MDP}^{\{\mathbb{E},\mathrm{VaR}_0\}}_{\mathsf{r},\mathsf{multi}}$, i.e. constraints on the expectation and ensuring that almost all runs achieve a given threshold, are in P [15].

Theorem 6.2. *For any* $\mathbf{obj} \in \{\mathsf{r},\mathsf{m}\}$, $\mathsf{MDP}^{\{\mathrm{CVaR}\}}_{\mathbf{obj},\mathsf{multi}}$ *is NP-hard (when the dimension* d *is a part of the input).*

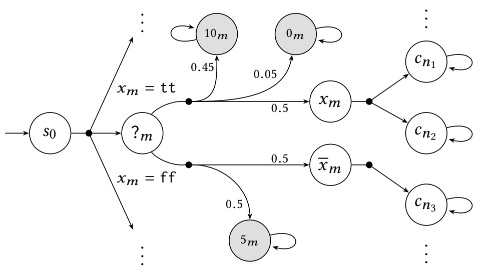


Figure 7. Gadget for variable x_m. Uniform transition probabilities are omitted for readability.

Proof. We prove hardness by reduction from 3-SAT. The core idea is to utilize observations from Fig. 3 and Ex. 5.14, namely that CVaR constraints can be used to enforce a deterministic choice.

Let $\{C_n\}$ be a set of N clauses with M variables x_m and set the dimensions $d = N+M$. By abuse of notation, n refers to the dimension of clause C_n and m to the one of variable x_m, respectively.

The gadget for the reduction is outlined in Fig. 7. Observe that, due to the structure of the MDP, we have that $R^{\mathsf{r}} = R^{\mathsf{m}}$.

Overall, the reduction works as follows. Initially, a state $?_m$, representing the variable x_m, is chosen uniformly. In this state, the strategy is asked to give the valuation of x_m through the actions "$x_m = \texttt{tt}$" or "$x_m = \texttt{ff}$". As seen in Ex. 5.14, the structure of the shaded states can be used to enforces a deterministic choice between the two actions. Particularly, in dimension m we require $\mathrm{CVaR}_p \geq 5$ for $p = \frac{M-1}{M} + \frac{1}{M} \cdot 0.5 \cdot 0.2$. Since all other gadgets yield 0 in dimension m and only half of the runs going through $?_m$ end up in the shaded area, this corresponds to Ex. 5.14, where $p = 0.2$.

Once in either state x_m or $\overline{x}_m$, a state c_n corresponding to a clause C_n satisfied by this assignment is chosen uniformly. In the example gadget, we would have $x_m \in C_{n_1} \cap C_{n_2}$, and $\overline{x}_m \in C_{n_3}$. We set the reward of c_n to 1_n. Then a clause c_n is satisfied under the assignment if the state c_n is visited with positive probability, e.g. if $\mathrm{CVaR}_1 \geq \frac{1}{M} \cdot 0.5 \cdot \frac{1}{N}$. Clearly, a satisfying assignment exists iff a strategy satisfying these constraints exists. □

6.2 NP-completeness and strategies for reachability

For weighted reachability, we prove that the previously presented bound is tight, i.e. that the weighted reachability problem with multiple dimensions and CVaR constraints is NP-complete when d is part of the input and P otherwise. First, we show that the strategy bounds of the single dimensional case directly transfer. Intuitively, this is the case since only the steady state distribution over the target set T is relevant, independent of the dimensionality.

Theorem 6.3. *Two-memory stochastic strategies (i.e. with both randomization and stochastic update) are sufficient for* $\mathsf{MDP}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}_{\mathsf{r},\mathsf{multi}}$. *Moreover, if* $\mathbf{r}_j(s) \geq 0$ *for all* $s \in T$ *and* $j \in [d]$, *then memoryless randomizing strategies are sufficient.*

Proof. Follows directly from the reasoning used in the proofs of Thm. 5.10 and Thm. 5.4. □

(1) All variables y_a, x_s, $\underline{x}^j_s$ are non-negative.
(4) VaR-consistent split for $j \in [d]$:

$$\underline{x}^j_s = x_s \text{ for } s \in T^j_< \qquad \underline{x}^j_s \leq x_s \text{ for } s \in T^j_=$$

(5) Probability-consistent split for $j \in [d]$:

$$\sum\nolimits_{s \in T^j_\leq} \underline{x}^j_s = \mathbf{p}_j$$

(6) CVaR and expectation satisfaction for $j \in [d]$:

$$\sum\nolimits_{s \in T^j_\leq} \underline{x}_s \cdot \mathbf{r}(s) \geq \mathrm{p}_j \cdot \mathbf{c}_j \qquad \sum\nolimits_{s \in T} x_s \cdot \mathbf{r}_j(s) \geq \mathbf{e}_j$$

Figure 8. LP used to decide multi-dimensional weighted reachability queries given a guess $\mathbf{t}$ of $\mathrm{VaR}_{\mathrm{p}_j}$. Equalities (2) and (3) are as in Fig. 4, $T^j_\sim := \{s \in T \mid \mathbf{r}_j(s) \sim \mathbf{t}_j\}$, $\sim \in \{<, =, \leq\}$.

Theorem 6.4. $\mathsf{MDP}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}_{\mathsf{r},\mathsf{multi}}$ *is in NP if d is a part of the input; moreover, it is in P for any fixed d.*

Proof sketch. To prove containment, we guess the VaR threshold vector $\mathbf{t}$ out of the set of potential ones, namely $\{r \mid \exists i \in [d], s \in T.\mathbf{r}_i(s) = r\}^d$ and use an LP to verify the solution. We again assume that each MEC can reach the target set and is single-state, as we did for Fig. 4. The arguments used to resolve this assumption are still applicable in the multi-dimensional setting. The LP consists of the flow Equalities (2) and (3) from the LP in Fig. 4 together with the modified (In)Equalities (4)-(6) as shown in Fig. 8.

The difference is that we extract the worst fraction of the flow *in each dimension.* Consequently, we have d instances of each $\underline{x}_s$ variable, namely $\underline{x}^j_s$. The number of possible guesses $\mathbf{t}$ is bounded by $|T|^d$ and thus the guess is of polynomial length. For a fixed d the bound itself is polynomial and hence, as previously, we can try out all vectors. □

6.3 Upper bounds of mean payoff

In this section, we provide an upper bound on the complexity of mean-payoff queries. Strategies in this context are known to have higher complexity.

Proposition 6.5 ([9]). *Infinite memory is necessary for* $\mathsf{MDP}^{\{\mathbb{E}\}}_{\mathsf{m},\mathsf{multi}}$.

Note that this directly transfers to $\mathsf{MDP}^{\{\mathrm{CVaR}\}}_{\mathsf{m},\mathsf{multi}}$, as $\mathrm{CVaR}_1 = \mathbb{E}$. However, closing gaps between lower and upper bounds for the mean payoff objective is notoriously more difficult. For instance, $\mathsf{MDP}^{\{\mathrm{VaR}\}}_{\mathsf{m},\mathsf{multi}}$ is known to be in EXP, but not even known to be NP-hard, neither is $\mathsf{MDP}^{\{\mathbb{E},\mathrm{VaR}\}}_{\mathsf{m},\mathsf{multi}}$. Since we have proven that $\mathsf{MDP}^{\{\mathrm{CVaR}\}}_{\mathsf{m},\mathsf{multi}}$ is NP-hard, we can expect that obtaining the matching NP upper bound will be yet more difficult. The fundamental difference of the multi-dimensional mean-payoff case is that the solutions within MECs cannot be pre-computed, rather non-trivial trade-offs must be considered. Moreover, the trade-offs are not "local" and must be synchronized over all the MECs, see [15] for details.

We now observe that, as opposed to quantile queries, i.e. VaR constraints, the behaviour inside each MEC can be assumed to be quite simple. Our results primarily rely on [16] and use a similar notation. In particular, given a run ρ, $\mathrm{Freq}_a(\rho)$ yields the average frequency of action a, i.e. $\mathrm{Freq}_a(\rho) := \liminf_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} \mathbb{1}_a(a_t)$, where a_t refers to the action taken by ρ in step t.

Definition 6.6. A strategy σ is *MEC-constant* if for all $M_i \in \mathsf{MEC}$ with $\mathbb{P}^\sigma[\Diamond\Box M_i] > 0$ and all $j \in [d]$ there is a $v \in \mathbb{R}$ such that $\mathbb{P}^\sigma[R_j^{\mathrm{m}} = v \mid \Diamond\Box M_i] = 1$.

Lemma 6.7. *MEC-constant strategies are sufficient for* $\mathsf{MDP}^{\{\mathbb{E},\mathrm{CVaR}\}}_{\mathrm{m,multi}}$.

Proof. Fix an MDP $\mathcal{M}$ with MECs $\mathsf{MEC} = \{M_1, \dots, M_n\}$, reward function $\mathbf{r}$ and a strategy σ. Further, define $p_i = \mathbb{P}^\sigma[\Diamond\Box M_i]$. We construct a strategy σ' so that (i) $\mathbb{P}^{\sigma'}[\Diamond\Box M_i] = p_i$ for all M_i, and (ii) all behaviours of σ on a MEC M_i are "mixed" into each run on M_i, making it MEC-constant.

We first define the mixing strategies σ_i, achieving point (ii). By [16, Sec. 4.1], there are frequencies $(x_a)_{a\in\mathsf{A}}$ which

- satisfy $\sum_{a\in A} x_a \cdot \Delta(a,s) = \sum_{a\in\mathsf{Av}(s)} x_a$ for all $s \in S$,
- for each action a we have $\mathbb{E}^\sigma[\mathrm{Freq}_a] \le x_a$, and
- $\sum_{a\in\mathsf{A}\cap M_i} x_a = p_i$.

By [16, Cor. 5.5], there is a (Markov) strategy σ_i on M_i where

$$\mathbb{P}^{\sigma_i}\left[\mathrm{Freq}_a = x_a/p_i\right] = 1.$$

Consequently, σ_i is almost surely constant on M_i w.r.t. R^{m}. We apply the reasoning used in the proof of Thm. 5.10 to obtain the combined strategy σ' which achieves point (i) and switches to σ_i upon remaining in M_i.

Now, fix any $j \in [d]$, $M_i \in \mathsf{MEC}$, and $p, q \in (0,1)$. We have that $\mathbb{E}^{\sigma_i}[\mathrm{Freq}_a \mid \Diamond\Box M_i] \ge \mathbb{E}^\sigma[\mathrm{Freq}_a \mid \Diamond\Box M_i]$ by construction. Consequently, $\mathbb{E}^{\sigma'}(R_j^{\mathrm{m}}) \ge \mathbb{E}^\sigma(R_j^{\mathrm{m}})$.

Since σ' is MEC-constant, we have $\mathrm{CVaR}_p^{\sigma'}(R_j^{\mathrm{m}} \mid \Diamond\Box M_i) = \mathbb{E}^{\sigma'}[R_j^{\mathrm{m}} \mid \Diamond\Box M_i]$. Further, by $\mathbb{E}^\sigma[\mathrm{Freq}_a \mid \Diamond\Box M_i] \cdot p_i \le \mathbb{E}^{\sigma_i}[\mathrm{Freq}_a]$ for all a, we get $\mathbb{E}^\sigma[R_j^{\mathrm{m}} \mid \Diamond\Box M_i] \le \mathbb{E}^{\sigma_i}[R_j^{\mathrm{m}}]$. So, $\mathrm{CVaR}_p^{\sigma_i}(R_j^{\mathrm{m}}) = \mathbb{E}^{\sigma_i}[R_j^{\mathrm{m}}] \ge \mathbb{E}^\sigma[R_j^{\mathrm{m}} \mid \Diamond\Box M_i] \ge \mathrm{CVaR}_q^\sigma(R_j^{\mathrm{m}} \mid \Diamond\Box M_i)$, as $\mathrm{CVaR} \le \mathbb{E}$.

Finally, we apply this inequality together with property (i), obtaining $\mathrm{CVaR}_p^\sigma(R_j^{\mathrm{m}}) \le \mathrm{CVaR}_p^{\sigma'}(R_j^{\mathrm{m}})$ by [28, Thm. A.4] □

We utilize this structural property to design a linear program for these constraints. However, similarly to the previously considered LPs, it relies on knowing the VaR for each $\mathrm{CVaR}_{\mathrm{p}}$ constraint. Due to the non-linear behaviour of CVaR, the classical techniques do not allow us to conclude that VaR is polynomially sized and thus we do not present the "matching" NP upper bound, but a PSPACE upper bound, which we achieve as follows.

Theorem 6.8. $\mathsf{MDP}^{\{\mathbb{E},\mathrm{CVaR}\}}_{\mathrm{m,multi}}$ *is in PSPACE.*

Proof sketch. We use the existential theory of the reals, which is NP-hard and in PSPACE [12], to encode our problem. The VaR vector $\mathbf{t}$ is existentially quantified and the formula is a polynomially sized program with constraints linear in VaR's and linear in the remaining variables. This shows the complexity result.

The details of the procedure are as follows. For each $j \in [d]$, we use the existential theory of reals to guess the achieved VaR $\mathbf{t} = \mathrm{VaR}_{\mathbf{p}_j}$. Further, we non-deterministically obtain the following polynomially-sized information (or deterministically try out all options in PSPACE). For each $j \in [d]$ and for each MEC M_i, we guess if the value achieved in M_i is at most (denoted $M_i \in \mathsf{MEC}^j_{\le}$) or above (denoted $M_i \in \mathsf{MEC}^j_{>}$) the respective $\mathbf{t}_j$, and exactly one MEC $M^j_{=}$, which achieves a value equal to it. Given these guesses, we check whether the LP in Fig. 9 has a solution.

(1) All variables y_a, y_s, x_a, x_s are non-negative.

(2) Transient flow for $s \in S$:

$$\mathbb{1}_{s_0}(s) + \sum\nolimits_{a\in\mathsf{A}} y_a \cdot \Delta(a,s) = \sum\nolimits_{a\in\mathsf{Av}(s)} y_a + y_s$$

(3) Probability of switching in a MEC is the frequency of using its actions for $M_i \in \mathsf{MEC}$:

$$\sum\nolimits_{s\in M_i} y_s = \sum\nolimits_{a\in M_i} x_a$$

(4) Recurrent flow for $s \in S$:

$$x_s = \sum\nolimits_{a\in\mathsf{A}} x_a \cdot \Delta(a,s) = \sum\nolimits_{a\in\mathsf{Av}(s)} x_a$$

(5) CVaR and expectation satisfaction for $j \in [d]$:

$$\sum\nolimits_{s\in S^j_{\le}} x_s \cdot \mathbf{r}_j(s) + \left(\mathbf{p}_j - \sum\nolimits_{s\in S^j_{\le}} x_s\right) \cdot \mathbf{t}_j \ge \mathbf{p}_j \cdot \mathbf{c}_j$$
$$\sum\nolimits_{s\in S} x_s \cdot \mathbf{r}_j(s) \ge \mathbf{e}_j$$

(6) Verify MEC classification guess for $j \in [d]$:

$$\sum\nolimits_{s\in M^j_{\le}} x_s \cdot \mathbf{r}_j(s) \le \mathbf{t}_j \quad \text{for } M^j_{\le} \in \mathsf{MEC}^j_{\le} \cup \{M^j_{=}\}$$
$$\sum\nolimits_{s\in M^j_{\ge}} x_s \cdot \mathbf{r}_j(s) \ge \mathbf{t}_j \quad \text{for } M^j_{\ge} \in \mathsf{MEC}^j_{>} \cup \{M^j_{=}\}$$

(7) Verify VaR guess for $j \in [d]$:

$$\sum\nolimits_{s\in S^j_{\le}} x_s \le \mathbf{p}_j \qquad \sum\nolimits_{s\in S^j_{\le}\cup M^j_{=}} x_s \ge \mathbf{p}_j$$

Figure 9. LP used to decide multi-dimensional mean-payoff queries given a guess $\mathbf{t}$ of $\mathrm{VaR}_{\mathbf{p}_j}$ and MEC classification $\mathsf{MEC}^j_{\le}$, $M^j_{=}$, and $\mathsf{MEC}^j_{>}$. $S^j_{\sim} := \{s \in S \mid s \in M \text{ and } M \in \mathsf{MEC}^j_{\sim}\}$, $\sim \in \{\le, >\}$.

Equations (1)-(4) describe the transient flow like the previous LP's and, additionally, the recurrent flow like in [31, Sec. 9.3] or [9, 16, 19]. This addition is needed, since now our MECs are not trivial, i.e. single state. Again, Inequalities (5) verify that the CVaR and expectation constraints are satisfied. Finally, Inequalities (6) and (7) verify the previously guessed information, i.e. the VaR vector and the MEC classification.

Using the very same techniques, it is easy to prove that solutions to the LP correspond to satisfying strategies and vice versa. In particular, Inequalities (6) and (7) directly make use of the MEC-constant property of Lem. 6.7. □

While MEC-constant strategies are sufficient for $\mathbb{E}$ with CVaR, in contrast, they are not even for just $\mathsf{MDP}^{\{\mathrm{VaR}\}}_{\mathrm{m,multi}}$ [15, Ex.22]. Consequently, only an exponentially large LP is known for $\mathsf{MDP}^{\{\mathrm{VaR}\}}_{\mathrm{m,multi}}$. We can combine all the objective functions together as follows:

Theorem 6.9. $\mathsf{MDP}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}_{\mathrm{m,multi}}$ *is in EXPSPACE.*

Proof sketch. We proceed exactly as in the previous case, but now the flows in Equality (4) are split into exponentially many flows, depending on the set of dimensions where they achieve the given VaR threshold, see LP L in [15, Fig. 4]. The resulting size of the program is polynomial in the size of the system and exponential in d. Hence the call to the decision procedure of the existential theory of reals results in the EXPSPACE upper bound. □

Table 1. Schematic summary of known and new results. Strategies are abbreviated by "C/*n*-M", where C is either *D*eterministic or *R*andomizing, *n* is the size of the memory, and M is either *D*etereministic or *S*tochastic *MEM*ory.

dim	single		multi				
obj	any		r	m			
crit	$\lvert\textbf{crit}\rvert = 1$	$\lvert\textbf{crit}\rvert \geq 2$	CVaR $\in$ **crit**	$\{\mathbb{E}, \mathrm{VaR}_0\}$	$\{\mathrm{VaR}\}$	$\{\mathrm{CVaR}\}$, $\{\mathrm{CVaR}, \mathbb{E}\}$	$\{\mathbb{E}, \mathrm{CVaR}, \mathrm{VaR}\}$
Complex.	P		NP-c., P for fixed *d*	P	EXP	NP-h., PSPACE	NP-h., EXPSPACE
Strat.	D/1-MEM	R/2-SMEM	R/2-SMEM			R/∞-DMEM	

7 Conclusion

We introduced the conditional value-at-risk for Markov decision processes in the setting of classical verification objectives of reachability and mean payoff. We observed that in the single dimensional case the additional CVaR constraints do not increase the computational complexity of the problems. As such they provide a useful means for designing risk-averse strategies, at no additional cost. In the multidimensional case, the problems become NP-hard. Nevertheless, this may not necessarily hinder the practical usability. Our results are summarized in Table 1.

We conjecture that the VaR's for given CVaR constraints are polynomially large numbers. In that case, the provided algorithms would yield NP-completeness for $\mathsf{MDP}^{\{\mathrm{CVaR}\}}_{\mathsf{m},\mathsf{multi}}$ and EXPTIME-containment for $\mathsf{MDP}^{\{\mathbb{E},\mathrm{VaR},\mathrm{CVaR}\}}_{\mathsf{m},\mathsf{multi}}$, where the exponential dependency is only on the dimension, not the size of the system.

Acknowledgments

This research has been partially supported by the Czech Science Foundation grant No. 18-11193S and the German Research Foundation (DFG) project KR 4890/2 "Statistical Unbounded Verification" (383882557). We thank Vojtěch Forejt for bringing up the topic of CVaR and the initial discussions with Jan Krčál and wish them both happy life in industry. We also thank Michael Luttenberger and the anonymous reviewers for insightful comments and valuable suggestions.

References

[1] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. 1999. Coherent Measures of Risk. *Mathematical Finance* 9, 3 (1999), 203–228.

[2] Pranav Ashok, Krishnendu Chatterjee, Przemyslaw Daca, Jan Křetínský, and Tobias Meggendorfer. 2017. Value Iteration for Long-Run Average Reward in Markov Decision Processes. In *CAV (LNCS)*, Vol. 10426. Springer, 201–221.

[3] Christel Baier, Marcus Daum, Clemens Dubslaff, Joachim Klein, and Sascha Klüppelholz. 2014. Energy-Utility Quantiles. In *NFM (LNCS)*, Vol. 8430. Springer, 285–299.

[4] Christel Baier, Clemens Dubslaff, and Sascha Klüppelholz. 2014. Trade-off analysis meets probabilistic model checking. In *CSL-LICS*. ACM, 1:1–1:10.

[5] 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 (LNCS)*, Vol. 8411. Springer, 1–16.

[6] Christel Baier, Joachim Klein, Sascha Klüppelholz, and Sascha Wunderlich. 2017. Maximizing the Conditional Expected Reward for Reaching the Goal. In *TACAS (LNCS)*, Vol. 10206. 269–285.

[7] Nicole Bäuerle and Jonathan Ott. 2011. Markov Decision Processes with Average-Value-at-Risk criteria. *Math. Meth. of OR* 74, 3 (2011), 361–379.

[8] Tanya Styblo Beder. 1995. VAR: Seductive but dangerous. *Financial Analysts Journal* 51, 5 (1995), 12–24.

[9] 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).

[10] Tomás Brázdil, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. 2013. Trading Performance for Stability in Markov Decision Processes. In *LICS*. IEEE Computer Society, 331–340.

[11] Véronique Bruyère, Emmanuel Filiot, Mickael Randour, and Jean-François Raskin. 2017. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. *Inf. Comput.* 254 (2017), 259–295.

[12] John F. Canny. 1988. Some Algebraic and Geometric Computations in PSPACE. In *STOC*. ACM, 460–467.

[13] Stefano Carpin, Yinlam Chow, and Marco Pavone. 2016. Risk aversion in finite Markov Decision Processes using total cost criteria and average value at risk. In *ICRA*. IEEE, 335–342.

[14] Krishnendu Chatterjee, Vojtech Forejt, and Dominik Wojtczak. 2013. Multi-objective Discounted Reward Verification in Graphs and MDPs. In *LPAR (LNCS)*, Vol. 8312. Springer, 228–242.

[15] Krishnendu Chatterjee, Zuzana Komárková, and Jan Křetínský. 2015. Unifying Two Views on Multiple Mean-Payoff Objectives in Markov Decision Processes. In *LICS*. IEEE Computer Society, 244–256.

[16] Krishnendu Chatterjee, Zuzana Křetínská, and Jan Křetínský. 2017. Unifying Two Views on Multiple Mean-Payoff Objectives in Markov Decision Processes. *LMCS* 13, 2 (2017).

[17] Lorenzo Clemente and Jean-François Raskin. 2015. Multidimensional beyond Worst-Case and Almost-Sure Problems for Mean-Payoff Objectives. In *LICS*. IEEE Computer Society, 257–268.

[18] Costas Courcoubetis and Mihalis Yannakakis. 1995. The Complexity of Probabilistic Verification. *J. ACM* 42, 4 (1995), 857–907.

[19] Kousha Etessami, Marta Z. Kwiatkowska, Moshe Y. Vardi, and Mihalis Yannakakis. 2008. Multi-Objective Model Checking of Markov Decision Processes. *LMCS* 4, 4 (2008).

[20] J.A. Filar, D. Krass, and K.W Ross. 1995. Percentile performance criteria for limiting average Markov decision processes. *IEEE Trans. Automat. Control* 40, 1 (Jan 1995), 2–10.

[21] Jerzy A. Filar, Dmitry Krass, and Keith W. Ross. 1995. Percentile performance criteria for limiting average Markov decision processes. *IEEE Trans. Automat. Control* 40 (1995), 2–10.

[22] Vojtech Forejt, Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. 2011. Quantitative Multi-objective Verification for Probabilistic Systems. In *TACAS (LNCS)*, Vol. 6605. Springer, 112–127.

[23] Hugo Gilbert, Paul Weng, and Yan Xu. 2017. Optimizing Quantiles in Preference-Based Markov Decision Processes. In *AAAI*. AAAI Press, 3569–3575.

[24] Christoph Haase and Stefan Kiefer. 2015. The Odds of Staying on Budget. In *ICALP (LNCS)*, Vol. 9135. Springer, 234–246.

[25] Christoph Haase, Stefan Kiefer, and Markus Lohrey. 2017. Computing quantiles in Markov chains with multi-dimensional costs. In *LICS*. IEEE Computer Society, 1–12.

[26] Yonghui Huang and Xianping Guo. 2016. Minimum Average Value-at-Risk for Finite Horizon Semi-Markov Decision Processes in Continuous Time. *SIAM Journal on Optimization* 26, 1 (2016), 1–28.

[27] Masayuki Kageyama, Takayuki Fujii, Koji Kanefuji, and Hiroe Tsubaki. 2011. Conditional Value-at-Risk for Random Immediate Reward Variables in Markov Decision Processes. *American J. Computational Mathematics* 1, 3 (2011), 183–188.

[28] Jan Křetínský and Tobias Meggendorfer. 2018. *Conditional Value-at-Risk for Reachability and Mean Payoff in Markov Decision Processes.* Technical Report abs/1805.xxxxx. arXiv.org.

[29] Xiaocheng Li, Huaiyang Zhong, and Margaret L. Brandeau. 2017. Quantile Markov Decision Process. *CoRR* abs/1711.05788 (2017).

[30] Christopher W. Miller and Insoon Yang. 2017. Optimal Control of Conditional Value-at-Risk in Continuous Time. *SIAM J. Control and Optimization* 55, 2 (2017), 856–884.

[31] M. L. Puterman. 1994. *Markov Decision Processes.* J. Wiley and Sons.

[32] Mickael Randour, Jean-François Raskin, and Ocan Sankur. 2017. Percentile queries in multi-dimensional Markov decision processes. *FMSD* 50, 2-3 (2017), 207–248.

[33] R. Tyrrell Rockafellar and Stanislav Uryasev. 2000. Optimization of Conditional Value-at-Risk. *Journal of Risk* 2 (2000), 21–41.

[34] R Tyrrell Rockafellar and Stanislav Uryasev. 2002. Conditional value-at-risk for general loss distributions. *Journal of banking & finance* 26, 7 (2002), 1443–1471.

[35] Michael Ummels and Christel Baier. 2013. Computing Quantiles in Markov Reward Models. In *FoSSaCS (LNCS)*, Vol. 7794. Springer, 353–368.

II Note on Copyright

Springer Nature

The author has obtained the licence to include Papers A, B and D in the thesis from Springer Nature through the Copyright Clearance Center's RightsLink® service. The respective licence numbers are 4844770336906, 4844770517249, and 4844770740275.

LNCS Open Access

According to the Open Access policy of the *Lecture Notes in Computer Science* of the Springer-Verlag GmbH, the author of the thesis is permitted to include Paper C in the thesis. The relevant excerpt follows:

> This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (`http://creativecommons.org/licenses/by/4.0/`), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.
>
> The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

LIPIcs

According to the rules for publishing in LIPIcs (Leibniz International Proceedings in Informatics) with Schloss Dagstuhl Leibniz-Zentrum für Informatik GmbH, the author of the thesis is permitted to include Paper E in the thesis. The relevant excerpt follows:

> All publication series follow the concept of OpenAccess, i.e., the articles are freely available online for the reader and the rights are retained by the author.

For more information, please see `http://www.dagstuhl.de/publikationen/`.

ACM

According to the copyright policy of the Association for Computing Machinery (ACM), the author of the thesis is permitted to include Paper F in the thesis. The relevant excerpt follows:

> Reuse of any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.

For more information, please see `https://www.acm.org/publications/policies/copyright-policy`, in particular Section 2.5.