Information and Computation 285 (2022) 104886

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Value iteration for simple stochastic games: Stopping criterion and learning algorithm ☆

Julia Eisentraut [a], Edon Kelmendi [b], Jan Křetínský [a], Maximilian Weininger [a,*]

[a] Technical University of Munich, Germany
[b] University of Oxford, United Kingdom

ARTICLE INFO

Article history:
Received 31 August 2020
Received in revised form 23 February 2022
Accepted 27 February 2022
Available online 4 March 2022

Keywords:
Probabilistic verification
Stochastic games
Markov decision processes
Value iteration
Reachability

ABSTRACT

The classical problem of reachability in simple stochastic games is typically solved by value iteration (VI), which produces a sequence of under-approximations of the value of the game, but is only guaranteed to converge in the limit. We provide an additional converging sequence of over-approximations, based on an analysis of the game graph. Together, these two sequences entail the first error bound and hence the first stopping criterion for VI on simple stochastic games, indicating when the algorithm can be stopped for a given precision. Consequently, VI becomes an anytime algorithm returning the approximation of the value and the current error bound. We further use this error bound to provide a learning-based asynchronous VI algorithm; it uses simulations and thus often avoids exploring the whole game graph, but still yields the same guarantees. Finally, we experimentally show that the overhead for computing the additional sequence of over-approximations often is negligible.

1. Introduction

A *simple stochastic game* (SG) [24] is a zero-sum 2.5-player game played on a graph by the two players Maximizer and Minimizer, who choose actions in their respective states. Each action is associated with a probability distribution determining the next state to move to. The objective of Maximizer is to maximize the probability of reaching a given target state; the objective of Minimizer is the opposite.

Stochastic games constitute a fundamental problem for several reasons. From the theoretical point of view, the complexity of this problem[1] is known to be in **UP** ∩ **coUP** [16], but no polynomial-time algorithm is known. Further, several other important problems can be reduced to SGs, for instance parity games, mean-payoff games, discounted-payoff games and their stochastic extensions [16]. The task of solving SGs is also polynomial-time equivalent to solving perfect information Shapley, Everett and Gillette games [1]. Besides, the problem is practically relevant in verification and synthesis. SGs can model reactive systems, with players corresponding to the controller of the system and to its environment, where quantified uncertainty is explicitly modelled. This is useful in many application domains, ranging from smart energy management [20]

☆ This research was funded in part by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement No. 291763 for TUM – IAS, the Studienstiftung des Deutschen Volkes project "Formal methods for analysis of attack-defence diagrams", TUM IGSSE Grant 10.06 (PARSEC), and the German Research Foundation (DFG) project KR 4890/2-1 "Statistical Unbounded Verification".

* Corresponding author at: Technical University of Munich, Department of Informatics, Chair I 7, Boltzmannstr. 3, 85748 Garching, Germany.
E-mail address: maxi.weininger@tum.de (M. Weininger).

[1] Formally, the problem is to decide, for a given rational $p \in [0, 1]$ whether Maximizer has a strategy ensuring probability at least p to reach the target.

https://doi.org/10.1016/j.ic.2022.104886

over autonomous urban driving [22] and robot motion planning [49] to self-adaptive systems [14]; for various recent case studies, see e.g. [56]. Finally, since Markov decision processes (MDP) [53] are a special case with only one player, SGs can serve as abstractions of large MDPs [38].

Solution techniques. There are several classes of algorithms for solving SGs, namely strategy iteration algorithms, quadratic programming and value iteration algorithms [25]. *Strategy iteration* (also known as policy iteration) computes a sequence of strategies converging to the optimal one. In each iteration, we fix the strategy of one player and compute the best response of the opponent in the resulting one-player game (Markov decision process); this is typically done using linear programming. Then the strategy we originally fixed is improved based on the response of the opponent. Since there are finitely many strategies and they are monotonically improving, an optimal strategy is found after finitely many iterations. Currently, the best known upper bound for the number of iterations is exponential in the size of the SG.

Quadratic programming encodes the SG into a single polynomially sized quadratic program. The constraints of the program mimic the structure of the SG and the objective function ensures that the unique solution is the one where both players behave optimally. Both strategy iteration and quadratic programming produce an exact solution.

In contrast, *value iteration* (VI) is an approximative method that only converges in the limit. It computes a sequence of value-vectors that approximates the true values from below, but may never reach them in finite time. Still, there is a way to obtain the exact values: after $\mathcal{O}(\gamma^2)$ iterations of the algorithm, where γ is a number exponential in the size of the SG, we can round the approximation to the precise result [18]. However, as the required number of iterations is always exponential and the algorithm gives no guarantee before it finishes, this approach is infeasible even for small SGs. It would be useful to be able to stop the VI algorithm early, after a required precision is reached.

Theoretically, the best known upper bound of the runtime of all three classes of algorithms is exponential in the size of the SG. Practically, VI currently is the preferred solution. For instance, the most used probabilistic model checkers Storm [28] and PRISM [44], as well as PRISM's branch PRISM-Games [43] use VI for MDPs and SGs as the default option; in the case of PRISM-games, VI is the only option. As the exponential number of iterations that is necessary to obtain a guarantee is impractical, PRISM-games stops when the difference between the two most recent approximations is low, and thus may return arbitrarily imprecise results [33].

VI was preferred in these model checkers, because evaluating many large linear programs respectively a very large quadratic program was expected to be slow, similar to the special case of MDPs [40]. However, a recent extensive comparison of these algorithms [42] showed that in fact both strategy iteration and value iteration can perform similarly well. Also, there exists cases where either of the two algorithms outperforms the other, depending on the structure of the given SG.

An advantage of strategy iteration and quadratic programming is that they can possibly produce a guaranteed result in less than exponential time, while VI only converges in the limit and always has to run for an exponential number of steps in order to obtain a precise result. We address this by complementing the VI algorithm with error bounds, allowing us to stop the computation early when a required precision is reached. Thus we can leverage VI's ability to typically produce good results fast while also giving guarantees on the precision.

Value iteration with guarantees. In the special case of MDPs, in order to obtain bounds on the imprecision of the result, one can employ a *bounded* variant of VI [51,11] (also called *interval iteration* [33]). Here one computes not only an under-approximation, but also an over-approximation of the actual value. On the one hand, iterative computation of the least fixpoint of Bellman equations yields an under-approximating sequence converging to the value. On the other hand, iterative computation of the greatest fixpoint yields an over-approximation, which, however, does not converge to the value, because the least and greatest fixpoint do not coincide. Moreover, it often results in the trivial bound of 1. A solution suggested for MDPs [11,33] is to modify the underlying graph, namely to collapse end components. In the resulting MDP there is only one fixpoint, thus the least and greatest fixpoint coincide and both approximating sequences converge to the actual value. In contrast, for general SGs no value iteration based procedure is known which approximates the least fixpoint from above. In this paper we provide such an algorithm. Together with standard value iteration from below, this yields a stopping criterion to interrupt execution of VI at a given precision. We show that the pre-processing approach of collapsing is not applicable in general and provide a solution on the original graph. We also characterize SGs where the fixpoints coincide and naive VI converges from above. The main technical challenge is that states in an end component in SGs can have different values, in contrast to the case of MDPs.

Practical efficiency using guarantees. We further utilize the obtained guarantees to practically improve our algorithm. Similar to the MDP case [11], the quantification of the error allows for ignoring parts of the state space, and thus a speed up without jeopardizing the correctness of the result. Indeed, we provide a technique where some states are not explored and processed at all, but their potential effect is still taken into account. It relies on learning algorithms with simulations; these are guided by the quantification of the error, using the information to decide which state to explore and analyse next. While for MDPs this idea has already demonstrated speed ups in orders of magnitude [11,3], this paper provides the first technique of this kind for SGs.

Our contribution. We can summarize our contribution as follows.

- We introduce a VI algorithm for under- and over-approximation sequences, both of which converge to the value of the game. Thus we present the first practical stopping criterion for VI on SGs, yielding an anytime algorithm[2] with guaranteed precision. We also characterize when a simpler solution is sufficient.
- We provide a learning-based algorithm, which preserves the guarantees, and which additionally is more efficient in some cases since it avoids exploring the whole state space.
- We evaluate the running times of the algorithms experimentally, concluding that obtaining guarantees requires an overhead that is either negligible or mitigated by the learning-based approach.

Related work. The works closest to ours are the following. As mentioned above, [11,33] describe the solution to the special case of MDPs. While [11] also provides a learning-based algorithm, [33] discusses the convergence rate and the exact solution. The basic algorithm of [33] is implemented in PRISM [8] and the learning approach of [11] in Storm [28]. The extension for SGs where the interleaving of players is severely limited (every end component belongs to one player only) is discussed in [58].

Further, in the area of probabilistic planning, bounded real-time dynamic programming [51] is related to our learning-based approach. However, it is limited to the setting of stopping MDPs where the target sink or the non-target sink is reached almost surely under any strategy.

For stopping SGs, the error of value iteration can be bounded without computing an additional over-approximation. [25, Section 3] describes how to transform an arbitrary SG to a polynomially larger stopping SG such that the original value can be inferred. In essence, the construction adds a transition to every action that reaches a sink state with very low probability γ; this is the same as computing the discounted reachability with discount factor γ. However, this approach is impractical, because the convergence rate of value iteration depends on the lowest occurring probability in the SG. Further, in order to be able to infer the original value, γ has to be chosen smaller than standard machine precision already for an SG with 27 states [42, Section 4.1.4]. Our algorithm works for general SGs, not only for stopping ones, without increasing the size of the game or introducing very low probabilities.

The idea of strategy iteration was suggested in [35], for the class of irreducible concurrent SGs. In [60], the authors provide a strategy iteration algorithm for turn-based SGs with average reward objectives. The algorithm relies on the linear program formulation from [36] that solves the one-player game (Markov decision process) after fixing one strategy. The algorithm is also described in the textbook [32, Chapter 6.3]. Note that reachability objectives can be encoded as average reward objectives. The formulation of strategy iteration explicitly for simple stochastic games was given in [25], together with a randomized version. For a description of further developments and a practical comparison of all three algorithm classes (value iteration, strategy iteration and quadratic programming), see the recent work [42].

The tools implementing the standard SI and/or VI algorithms for SGs are PRISM-games [43], GAVS+ [23] and GIST [19]. GAVS+ is however not maintained anymore, and more for educational purposes. GIST considers ω-regular objectives, but performs only qualitative analysis. For MDPs (games with a single player), the recent friendly competition QComp [34] gives an overview of the existing tools.

Apart from fundamental algorithms to solve SGs, there are various practically efficient heuristics that, however, provide none or weak guarantees, often based on some form of learning [10,50,61,57,2,12]. For MDPs, the first probably approximately correct (PAC [59]) algorithm was given in [55]. This result has been further extended [11] to MDP settings more relevant for verification: firstly, [11] considers non-discounted unbounded horizon, such as the reachability objective. Secondly, it treats not only the case when the transition probabilities are unknown, but also when they are given.

In terms of complexity, [9] recently proved that deciding whether an action in an MDP is optimal for the n-step bounded reachability is EXPTIME-complete. This means that the worst case running time for VI on SGs is exponential, since VI computes the n-step bounded reachability for all states and since SGs are an extension of MDPs. The runtime is not larger than exponential, as [18] proved that VI can always compute the precise value in an exponential number of steps.

Finally, the concept of simple end component that is introduced in this paper is similar to *tangles* in parity games [29]. Both concepts describe a strongly connected sub-graph in which one of the players can surely win, which forces the other player to leave this sub-graph.

This article is the extended version of the paper presented at CAV 2018 [39]. We added more examples, an in-depth description of the learning-based algorithm as well as more on the implementation of both our algorithms and more experimental results. Also, we provide the full technical proofs, including many instructive details in the proofs of Theorem 1 and 2, as well as the new Theorem 3 about the learning-based algorithm. In particular, we extracted Lemma 2, a statement relating the existence of exiting actions to the existence of end components. Further, we clarified the case distinction in the proof of Theorem 1 and show in Fig. 4 the many ways in which a simple end component can affect the values of other states. Furthermore, in Lemma 6, we provide a detailed lattice-theoretic argument about the convergence of the upper bound to a fixpoint; this includes a surprising complication which is elaborated on in Remark 2. Moreover, we fixed an error in the statement and proof of Lemma 1, cf. Footnote 9. Finally, we give—to the best of our knowledge for the first time—a formal description of how to obtain optimal strategies from value vectors, see Appendix A.

[2] An algorithm which can be stopped at any time in the computation, returning an estimate and its precision.

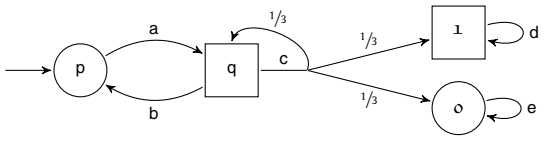


Fig. 1. An example of an SG with $S = \{\mathsf{p}, \mathsf{q}, \mathsf{1}, \mathsf{o}\}$, $S_{\square} = \{\mathsf{q}, \mathsf{1}\}$, $S_{\bigcirc} = \{\mathsf{p}, \mathsf{o}\}$, the initial state p and the set of actions $\mathsf{A} = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}\}$; $\mathsf{Av}(\mathsf{p}) = \{\mathsf{a}\}$ with $\delta(\mathsf{p}, \mathsf{a})(\mathsf{q}) = 1$; $\mathsf{Av}(\mathsf{q}) = \{\mathsf{b}, \mathsf{c}\}$ with $\delta(\mathsf{q}, \mathsf{b})(\mathsf{p}) = 1$ and $\delta(\mathsf{q}, \mathsf{c})(\mathsf{q}) = \delta(\mathsf{q}, \mathsf{c})(\mathsf{1}) = \delta(\mathsf{q}, \mathsf{c})(\mathsf{o}) = \frac{1}{3}$. For actions with only one successor, we do not depict the transition probability 1.

Several works have extended the original paper [39] already: the ideas have been lifted to a setting with limited information [6], with multiple objectives [4] and to concurrent stochastic games [30]. There also is an improved version of the algorithm based on *global* propagation of the over-approximation [52].

Organization of the paper. Section 2 introduces the basic notions and revises value iteration. Section 3 explains the idea of our approach on an example. Section 4 first characterizes the sub-graphs of the game which cause the non-convergence. Then it introduces the concept of *simple end components* and finally shows how to have a converging algorithm using a special treatment of these components. Section 5 describes the learning-based algorithm, Section 6 discusses experimental results and Section 7 concludes.

2. Preliminaries

2.1. Stochastic games

A probability distribution on a finite set X is a mapping $\delta\colon X \to [0, 1]$, such that $\sum_{x \in X} \delta(x) = 1$. The set of all probability distributions on X is denoted by $\mathcal{D}(X)$. We now define stochastic games, which are often referred to simple stochastic games or turn-based stochastic two-player games with a reachability objective. As opposed to the notation of e.g. [24], we do not have special stochastic nodes, but rather a probabilistic transition function.

Definition 1 *(SG).* A *stochastic game (SG)* is a tuple $(S, S_{\square}, S_{\bigcirc}, \mathsf{s}_0, \mathsf{A}, \mathsf{Av}, \delta)$ where S is a finite set of *states* partitioned[3] into the sets $S_{\square}$ and $S_{\bigcirc}$ of states of the player *Maximizer* and *Minimizer*[4] respectively, $\mathsf{s}_0 \in S$ is the *initial* state, A is a finite set of *actions*, $\mathsf{Av}\colon S \to 2^{\mathsf{A}}$ assigns to every state a set of *available* actions, and $\delta\colon S \times \mathsf{A} \to \mathcal{D}(S)$ is a *transition function* that given a state s and an action $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$ yields a probability distribution over *successor* states.

A *Markov decision process (MDP)* is a special case of an SG where $S_{\bigcirc} = \emptyset$, and a Markov chain (MC) is a special case of an MDP, where for all $\mathsf{s} \in S : \left|\mathsf{Av}(\mathsf{s})\right| = 1$.

We assume that SGs are non-blocking, so for all states s we have $\mathsf{Av}(\mathsf{s}) \neq \emptyset$. For a state s and an available action $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$, we denote the set of successors by $\mathsf{Post}(\mathsf{s}, \mathsf{a}) := \{\mathsf{s}' \mid \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') > 0\}$. Finally, for any set of states $T \subseteq S$, we use $T_{\square}$ and $T_{\bigcirc}$ to denote the states in T that belong to Maximizer and Minimizer, whose states are drawn in the figures as $\square$ and $\bigcirc$, respectively.

Since we consider the reachability objective, in addition to an SG we require a set of target states $F \subseteq S$. As it is irrelevant what happens after reaching a target state, we assume that all of them are absorbing, i.e. they only have one action that is a self-loop with probability 1.

An example of an SG is given in Fig. 1.

2.2. Semantics: paths and strategies

The semantics of SGs is given in the usual way, see e.g. [25], by means of strategies, the induced Markov chain and the respective probability space, as follows. An *infinite path* ρ is an infinite sequence $\rho = \mathsf{s}_0\mathsf{a}_0\mathsf{s}_1\mathsf{a}_1 \cdots \in (S \times \mathsf{A})^{\omega}$, such that for every $i \in \mathbb{N}$, $\mathsf{a}_i \in \mathsf{Av}(\mathsf{s}_i)$ and $\mathsf{s}_{i+1} \in \mathsf{Post}(\mathsf{s}_i, \mathsf{a}_i)$. *Finite paths* are defined analogously as elements of $(S \times \mathsf{A})^* \times S$.

As this paper deals with the reachability objective, we can restrict our attention to memoryless strategies, which are optimal for this objective [24]. A *strategy* of Maximizer, respectively Minimizer, is a function $\sigma\colon S_{\square} \to \mathcal{D}(\mathsf{A})$, respectively $S_{\bigcirc} \to \mathcal{D}(\mathsf{A})$, such that $\sigma(\mathsf{s}) \in \mathcal{D}(\mathsf{Av}(\mathsf{s}))$ for all s. We call a strategy *deterministic* if it maps to Dirac distributions only, i.e. if it plays a single action surely. Although deterministic strategies suffice, we still allow randomizing strategies, because they are needed for the learning-based algorithm later on. A pair (σ, τ) of strategies of Maximizer and Minimizer induces a Markov chain $\mathsf{G}^{\sigma,\tau}$ where the transition probabilities are defined as $\delta(\mathsf{s}, \mathsf{s}') = \sum_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \sigma(\mathsf{s}, \mathsf{a}) \cdot \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}')$ for states of Maximizer and analogously for states of Minimizer, with σ replaced by τ. The Markov chain induces a unique probability distribution $\mathbb{P}^{\sigma,\tau}_{\mathsf{s},\mathsf{G}}$ over measurable sets of infinite paths [7, Ch. 10].

[3] I.e., $S_{\square} \subseteq S$, $S_{\bigcirc} \subseteq S$, $S_{\square} \cup S_{\bigcirc} = S$, and $S_{\square} \cap S_{\bigcirc} = \emptyset$.

[4] The names are chosen, because Maximizer maximizes the probability of reaching a given target state, and Minimizer minimizes it.

We write $\Diamond F = \{\rho \mid \rho = s_0 a_0 s_1 a_1 \cdots \in (S \times A)^\omega \wedge \exists i \in \mathbb{N}.\ s_i \in F\}$ to denote the (measurable) set of all paths which eventually reach the target states F.

For each $s \in S$, we define the *value* in s as

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

where the equality follows from [24]. We omit the G in the subscript when it is clear from context.

To compute the value, we need take special care of *end components* (ECs). Intuitively, an EC is a subset of states of the SG, where the game can remain forever; i.e. given certain strategies of both players, there is no positive probability to exit the EC to some other state. An EC corresponds to a bottom strongly connected component in a Markov chain induced by some pair of strategies.

To define ECs formally, we first introduce the following notation. Given a set of states $T \subseteq S$ and a state $s \in T$, we say that (s, a) exits T if we have $s \in T$, $a \in Av(s)$ and $Post(s, a) \not\subseteq T$.

Definition 2 *(EC).* A non-empty set $T \subseteq S$ of states is an *end component (EC)* if there is a non-empty set $B \subseteq \bigcup_{s \in T} Av(s)$ of actions such that

1. for each $s \in T$, $a \in B \cap Av(s)$ we do *not* have (s, a) exits T,
2. for each $s, s' \in T$ there is a finite path $w = s a_0 \dots a_n s' \in (T \times B)^* \times T$, i.e. the path stays inside T and only uses actions in B.

An end component T is a *maximal end component (MEC)* if there is no other end component T' such that $T \subseteq T'$. Note that the MECs of an SG are equivalent to the MECs in an MDP with both players unified,[5] as the definition only depends on the existence of the actions. Thus the set of MECs of an SG G, denoted by $MEC(G)$, can be computed in polynomial time using the standard algorithm for MDPs, e.g. [17].

2.3. (Bounded) value iteration

Let $\mathcal{Z}$ be the set of states that cannot reach any state in F (called $\mathcal{Z}$, because they have value **z**ero). It can be computed, for example, by a reverse breadth-first-search from all states in F.

The value function V satisfies the following system of equations, referred to as the *Bellman equations*:

$$V(s) := \begin{cases} 1 & \text{if } s \in F \\ 0 & \text{if } s \in \mathcal{Z} \\ \max_{a \in Av(s)} V(s,a) & \text{if } s \in S_\square \\ \min_{a \in Av(s)} V(s,a) & \text{if } s \in S_\bigcirc \end{cases} \tag{1}$$

where[6]

$$V(s,a) := \sum_{s' \in S} \delta(s,a,s') \cdot V(s') \tag{2}$$

Moreover, V is the *least* solution to the Bellman equations, see e.g. [18]. To compute the value of V for all states in an SG, one can thus utilize the iterative approximation method *value iteration (VI)* as follows: we start with a lower bound function $L_0 \colon S \to [0,1]$ such that $L_0(s) = 1$ for target states $s \in F$ and for all other states $s \in S \setminus F$, we have $L_0(s) = 0$. Then we repetitively apply Bellman updates according to Equations (3) and (4)

$$L_n(s,a) := \sum_{s' \in S} \delta(s,a,s') \cdot L_{n-1}(s') \tag{3}$$

$$L_n(s) := \begin{cases} \max_{a \in Av(s)} L_n(s,a) & \text{if } s \in S_\square \\ \min_{a \in Av(s)} L_n(s,a) & \text{if } s \in S_\bigcirc \end{cases} \tag{4}$$

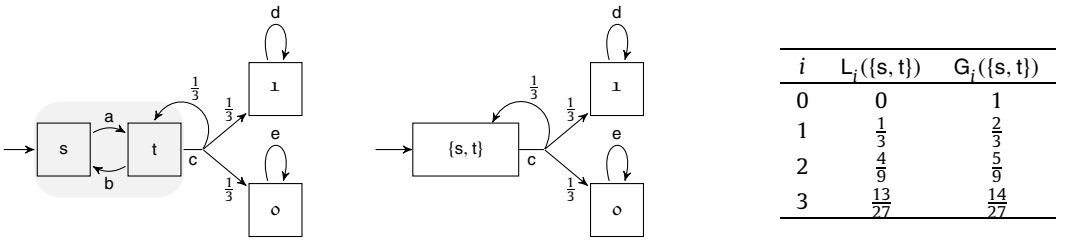
until convergence. Note that convergence may happen only in the limit even for such a simple game (even an MDP) as in Fig. 2 on the left. The sequence is continuous and monotonic, because all involved operations – summation and multiplication – are continuous and monotonic. Moreover, at all times it is a *lower* bound on V, i.e. $L_i(s) \le V(s)$ for all $s \in S$, and the least fixpoint satisfies $L^* := \lim_{n \to \infty} L_n = V$, see e.g. [18].

[5] Intuitively, this is also why it does not suffice to consider MECs in an SG, and we need to introduce the stronger notion of simple end component.

[6] Throughout the paper, for any function $f \colon S \to [0,1]$ we overload the notation and also write $f(s,a)$ meaning $\sum_{s' \in S} \delta(s,a,s') \cdot f(s')$.

Algorithm 1 Bounded value iteration algorithm.

```
1: procedure BVI(precision ε > 0)
2:     for s ∈ S do # Initialization
3:         L() (s) = 0   # Lower bound
4:         U(s) = 1   # Upper bound
5:     for s ∈ F do L(s) = 1 # Value of targets is determined a priori
6:     for s ∈ Z do U(s) = 0 # Value of sinks is determined a priori
7:     repeat
8:         UPDATE(L, U)       # Bellman updates or their modification
9:     until U(s_0) − L(s_0) < ϵ # Guaranteed error bound
```



i	$\mathsf{L}_i(\{s,t\})$	$\mathsf{G}_i(\{s,t\})$
0	0	1
1	$\frac{1}{3}$	$\frac{2}{3}$
2	$\frac{4}{9}$	$\frac{5}{9}$
3	$\frac{13}{27}$	$\frac{14}{27}$

Fig. 2. Left: an MDP (as special case of SGs) where BVI does not converge due to the grayed EC. Middle: the same MDP where the EC is collapsed, making BVI converge. Right: the approximations illustrating the convergence of the MDP in the middle.

We achieve our goal when $\mathsf{L}(s_0)$ is ε-close to the value $\mathsf{V}(s_0)$. Then we have an ε-approximation of the value of the SG $\mathsf{V}(s_0)$ and can extract a corresponding pair of ε-optimal strategies for both players, i.e. strategies σ, τ such that $\mathbb{P}_{s_0}^{\sigma,\tau}(\Diamond F)$ is ε-close to $\mathsf{V}(s_0)$. To the best of our knowledge, no explicit description of how to obtain strategies from a vector of (an ε-approximation of the) values exists, thus we provide it in Appendix A.

Unfortunately, there is no known stopping criterion which estimates how close the current under-approximation is to the value. The current tools stop when the difference between two successive approximations is smaller than a certain threshold, which can lead to arbitrarily wrong results [33].

For the special case of MDPs, it has been suggested to also compute the greatest fixpoint [51] and thus an *upper* bound as follows. The function $\mathsf{G}\colon S \to [0,1]$ is initialized for all states $s \in S$ as $\mathsf{G}_0(s) := 1$ except for states $s \in \mathcal{Z}$ with no path to a target where $\mathsf{G}_0(s) := 0$. Then we repetitively apply updates (3) and (4), where L is replaced by G. The resulting sequence G_n again is monotonic and continuous, and moreover provides an upper bound on V. The greatest fixpoint $\mathsf{G}^* := \lim_n \mathsf{G}_n$ is the greatest solution to the Bellman equations on $[0,1]^S$. This approach is called *bounded value iteration (BVI)* (or *bounded real-time dynamic programming (BRTDP)* [51,11] or *interval iteration* [33]).

If $\mathsf{L}^* = \mathsf{G}^*$ then they are both equal to V and we say that *BVI converges*. BVI is guaranteed to converge in MDPs if the only ECs are trivial, i.e. the self-loops of target states in F or ECs with no path to a target in $\mathcal{Z}$. Otherwise, if there are non-trivial ECs, BVI does not converge. For MDPs, the authors of [11,33] independently introduced the solution to "collapse" all ECs, i.e. merge all states of an EC into one and preserve only the actions exiting the EC. Computing the greatest fixpoint on the modified MDP results in another sequence U_i of upper bounds on V, converging to $\mathsf{U}^* := \lim_n \mathsf{U}_n$, and it holds that $\mathsf{U}^* = \mathsf{V}$ and BVI converges. The next section illustrates the difficulty and the solution through collapsing on an example.

Collapsing can be extended to SGs, but it does not solve the problem for arbitrary SGs. This is also illustrated in an example in the next section. In certain cases, it can still be used as a heuristic to speed up the algorithm, see Section 6.2.

In summary, all versions of BVI discussed so far and later on in the paper follow the pattern of Algorithm 1. In the naive version, UPDATE just performs the Bellman update on L and U according to Equations (3) and (4). For a general MDP, U does not converge to V, but to G^*, and thus the termination criterion may never be met if $\mathsf{G}^*(s_0) - \mathsf{V}(s_0) > 0$. If the ECs are collapsed in pre-processing then U converges to V. For the general case of SGs, the collapsing approach fails and this paper provides another version of BVI where U converges to V, based on a more detailed structural analysis of the game.

3. Example: why over-approximations do not converge

In this section, we illustrate why naive BVI does not converge on SGs and sketch our solution on a few examples. We face two problems: **(i)** states in ECs promise overly high values, even in MDPs. **(ii)** ECs in SGs do not belong to a single player and thus states in them can have different values.

We always denote by G the sequence converging to the greatest solution of the Bellman equations (monotonically from above), while we use U for any sequence over-approximating V that one or another BVI algorithm suggests.

(i) We illustrate the issue that arises already for the special case of MDPs. Consider the MDP of Fig. 2 on the left. Although $\mathsf{V}(s) = \mathsf{V}(t) = 0.5$, we have $\mathsf{G}_i(s) = \mathsf{G}_i(t) = 1$ for all i. Indeed, the upper bound for t is always updated as the maximum of $\mathsf{G}_i(t,c)$ and $\mathsf{G}_i(t,b)$. Although $\mathsf{G}_i(t,c)$ decreases over time, $\mathsf{G}_i(t,b)$ remains the same, namely equal to $\mathsf{G}_i(s)$, which in turn

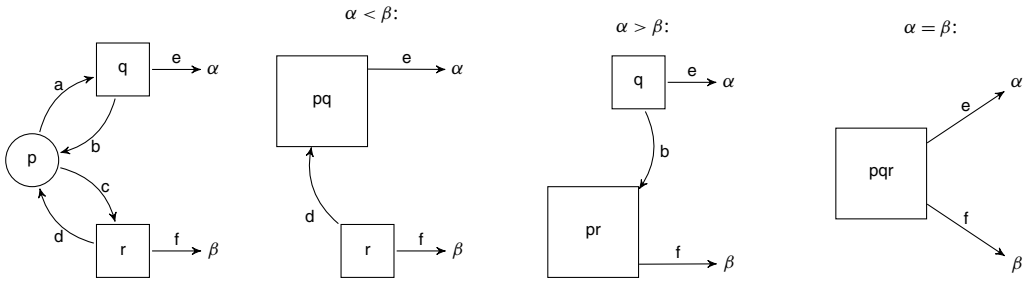


Fig. 3. Collapsing ECs in SGs may lead to incorrect results. The Greek letters on the leaving arrows denote the values of the exiting actions. Left figure: an example of an EC that can be collapsed in different ways depending on the relationship of α and β. Right three figures: correct collapsing in the different cases. In contrast to MDPs, some actions of the EC exiting the collapsed part have to be removed.

remains equal to $\mathsf{G}_i(\mathsf{s},\mathsf{a}) = \mathsf{G}_i(\mathsf{t})$. This cyclic dependency lets both s and t remain in an illusion that the value of the other one is 1.

The solution for MDPs is to remove this cyclic dependency by collapsing all MECs into singletons and removing the resulting purely self-looping actions. The middle of Fig. 2 shows the MDP after collapsing the EC $\{\mathsf{s},\mathsf{t}\}$. This turns the MDP into a stopping one, where the states in F or $\mathcal{Z}$ are reached with probability 1 under any strategy. In such MDPs, there is a unique solution to the Bellman equations. Therefore, the greatest fixpoint is equal to the least one and thus to V.

(ii) We illustrate the issues that additionally arise for general SGs. It turns out that the collapsing approach can be extended only to games where all states of each EC belong to one player only [58]. In this case, both Maximizer's and Minimizer's ECs are collapsed the same way as in MDPs.

However, when both players are present in an EC, then collapsing may not solve the issue. Consider the SG of Fig. 3. Here α and β represent the values of the respective actions.[7] There are three cases.

- First, let $\alpha < \beta$. If the bounds converge to these values we eventually observe $\mathsf{G}_i(q,e) < \mathsf{L}_i(r,f)$ and learn the induced inequality. Since p is a Minimizer's state it will never pick the action leading to the greater value of β. Therefore, we can safely merge p and q, and remove the action leading to r, as shown in the second sub-figure
- Second, if $\alpha > \beta$, p and r can be merged in an analogous way, as shown in the third sub-figure
- Third, if $\alpha = \beta$, both previous solutions as well as collapsing all three states as in the fourth sub-figure is possible. However, since the approximations may only converge to α and β in the limit, we may not know in finite time which of these cases applies and thus cannot decide for any of the collapses.

Sketching the solution. Consequently, the approach of collapsing is not applicable in general. In order to ensure BVI convergence, we suggest a different method, which we call *deflating*. It does not involve changing the state space, but rather decreasing the upper bound U_i to the least value that is currently provable (and thus still correct). To this end, we analyse the *exiting actions*, i.e. actions with successors outside of the EC, for the following reason: if the play stays in the EC forever, the target is never reached and Minimizer wins. Therefore, Maximizer has to pick some exiting action to avoid staying in the EC.

For the EC with the states s and t in Fig. 2, the only exiting action is c. In this example, since c is the only exiting action, $\mathsf{U}_i(\mathsf{t},\mathsf{c})$ is the highest possible upper bound that the EC can achieve. Thus, by decreasing the upper bound of all states in the EC to that number, we still have a safe upper bound. Moreover, with this modification BVI converges in this example, intuitively because now the upper bound of t depends on action c as it should. We choose the name "deflating" to evoke decreasing the overly high "pressure" in the EC until it equalizes with the actual "pressure" outside.

However, this does not lead to convergence of BVI yet. Consider for instance the SG in Fig. 3. It is correct to decrease the upper bound to the maximal exiting one, i.e. $\max\{\hat{\alpha},\hat{\beta}\}$, where $\hat{\alpha} := \mathsf{U}_i(\mathsf{a})$, $\hat{\beta} := \mathsf{U}_i(\mathsf{b})$ are the current approximations of α and of β, but this itself does not ensure BVI convergence. Indeed, if for instance $\hat{\alpha} < \hat{\beta}$ then deflating all states to $\hat{\beta}$ is not tight enough, as values of p and q can even be bounded by $\hat{\alpha}$. In fact, we have to find a certain sub-EC that corresponds to $\hat{\alpha}$, in this case $\{\mathsf{p},\mathsf{q}\}$ and set all its upper bounds to $\hat{\alpha}$. We define and compute these sub-ECs in the next section.

In summary, the general structure of our convergent BVI algorithm is to produce a sequence U by applying Bellman updates and occasionally finding the relevant sub-ECs and deflating them. The main technical challenge is that states in an EC in SGs can have different values, in contrast to the case of MDPs.

4. Convergent over-approximation

In Section 4.1, we characterize SGs where Bellman equations have more than one fixpoint. Based on this analysis, subsequent sections show how to alter the procedure computing the sequence G_i over-approximating V so that the resulting

[7] Precisely, they represent a probabilistic branching with probability α (or β) to $\imath$ and with the remaining probability to $\jmath$. For clarity, we omit this branching and depict only the value.

tighter sequence U_i still over-approximates V, but also converges to V, which ensures convergence of BVI. Section 5 presents the learning-based variant of our BVI.

4.1. Bloated end components cause non-convergence

As we have seen in the example of Fig. 3, BVI generally does not converge due to ECs with a particular structure of the exiting actions. The analysis of ECs relies on the extremal values that can be achieved by exiting actions (in the example, α and β). Given the value function V or just its current over-approximation U_i, we define the most profitable exiting action for Maximizer (denoted by $\square$) and Minimizer (denoted by $\bigcirc$) as follows.

Definition 3 *(*bestExit*).* Given a set of states $T \subseteq S$ and a function $f : S \to [0,1]$ (and its equivalent on state-action pairs, see Footnote 6), the best exit according to f from T of Maximizer and Minimizer, respectively, is defined as

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

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

with the convention that $\max_{\emptyset} = 0$ and $\min_{\emptyset} = 1$.

Example 1 *(Non-convergence).* In the example of Fig. 3 on the left with $T = \{\mathsf{p},\mathsf{q},\mathsf{r}\}$ and $\alpha < \beta$, we have $\mathsf{bestExit}^{\square}_{\mathsf{V}}(T) = \beta$ and $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T) = 1$. It is due to $\beta < 1$ that BVI does not converge here. We generalize this in the following lemma. △

Lemma 1 *(Multiple solutions). Let $T \subseteq S \setminus F$ be an EC. For every m satisfying $\mathsf{bestExit}^{\square}_{\mathsf{V}}(T) \le m \le \mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T)$, there is a solution $\mathsf{U} : S \to [0,1]$ to the Bellman equations, which for all states in T is greater than m, and for at least one state is equal to m.*

Proof. Intuitively, this lemma states the following: given an over-approximation that is greater than what T can actually achieve, but smaller than what Minimizer can certainly restrict to, Bellman updates (value iteration) will be stuck at the over-approximation. However, constructing the exact over-approximation where the iteration is stuck is technically involved.

Let m satisfy $\mathsf{bestExit}^{\square}_{\mathsf{V}_\mathsf{G}}(T) \le m \le \mathsf{bestExit}^{\bigcirc}_{\mathsf{V}_\mathsf{G}}(T)$. Note that we explicitly added G in the index, as we will later construct a modified SG G' and thus have to make clear to which SG we are referring to. We first construct a function from states to real numbers, then prove that it is greater than m for all states in T and afterwards that it is a solution to the Bellman equations, i.e. a fixpoint of applying Equations (3) and (4). Lastly, we show that it assigns exactly m to at least one state in T.

We obtain a modified SG G' from the given SG G as follows: for every Maximizer state in T, add an action that leads to a target state with probability m and to a sink state with probability $(1-m)$. Intuitively, this allows all Maximizer states to actually achieve the over-approximation m in every state. Further, we have to treat the corner case of ECs that only consist of Minimizer states. If there is an EC $T' \subseteq T_{\bigcirc}$, then all states in T' also get an action leading to a target with m and to a sink with the $1-m$; moreover, all other actions of these states are removed. Let $\mathsf{U} := \mathsf{V}_{\mathsf{G}'}$ be the value function of G'.

We now prove that for all states $\mathsf{s} \in T$, it holds that $\mathsf{U}(\mathsf{s}) = \mathsf{V}_{\mathsf{G}'}(\mathsf{s}) \ge m$. If s is a Maximizer state, then it has the newly added action with value m. Thus, the value in the modified SG[8] of every $\mathsf{s} \in T_{\square}$ is at least m (but possibly higher[9]).

If s is a Minimizer state, we have to treat the corner case of an EC consisting only of Minimizer states. If s is in an EC $T' \subseteq T_{\bigcirc}$, then by construction it only has an action with value m, and thus its value is (at least) m. Finally, we consider the case of a Minimizer state which is not part of an EC $T' \subseteq T_{\bigcirc}$. We know that for all leaving actions (actions $\mathsf{a}_\ell \in \mathsf{Av}(\mathsf{s})$ with $(\mathsf{s},\mathsf{a}_\ell)$ exits T) it holds that

$$\mathsf{V}_{\mathsf{G}'}(\mathsf{s},\mathsf{a}_\ell) \ge \mathsf{V}_{\mathsf{G}}(\mathsf{s},\mathsf{a}_\ell) \ge \mathsf{bestExit}^{\bigcirc}_{\mathsf{V}_\mathsf{G}} \ge m$$

The first inequality holds, because in G' we only added more possibility to reach the target, potentially increasing the value; note in particular that ECs with only Minimizer states actually have a value of 0. The second inequality follows from the definition of bestExit and the third from the choice of m.

[8] For the remainder of this sub-proof of $\mathsf{V}_{\mathsf{G}'}(\mathsf{s}) \ge m$, we omit the phrase "in the modified SG" for ease of notation.

[9] In the original paper [39], this lemma was wrong, since it claimed that the solution has to be constant on T. We provide a counter example. Consider the leftmost EC from Fig. 3. Let action e lead to a target with probability $\frac{1}{2}$ and to r with probability $\frac{1}{2}$; let $\beta = \frac{1}{2}$. Then we have $\mathsf{V}(r) = \mathsf{V}(r,f) = \frac{1}{2}$ and $\mathsf{V}(q) = \mathsf{V}(q,e) = \frac{1}{2} + \frac{1}{2}\cdot\frac{1}{2} = \frac{3}{4}$. Note that the Minimizer state p prefers to go down using action c, as $\mathsf{V}(r) < \mathsf{V}(q)$. The best exit is $\mathsf{bestExit}^{\square}_{\mathsf{V}}(T) = \frac{3}{4}$. However, picking $m = \frac{3}{4}$ and setting $\mathsf{U}(q) = \mathsf{U}(p) = \mathsf{U}(r) = m$ is not a fixpoint of Bellman updates, because applying an update on q we have $\mathsf{U}(q,e) = \frac{1}{2} + \frac{1}{2}\cdot m = \frac{7}{8} > m$. This is why we adjusted the statement of the lemma to only require equality with m in at least one state in T, not all of them. This weaker claim still suffices to prove Corollary 1.

Now we consider staying actions of s. Let X be the set of states that are reached by following Minimizer's optimal staying actions starting from s. Formally, X is initialized to $\{s\}$ and then for all $s' \in X_{\bigcirc}$ we recursively add $\arg\min_{\substack{a \in Av(s) \\ \neg(s,a)\,\text{exits}\,T}} \mathsf{Post}(s,a)$. As T is finite and only states from T can be added (since we consider staying actions), X will converge. We analyze all cycles in X. If there is an EC $X' \subseteq X_{\bigcirc}$ consisting only of Minimizer states, we already proved that its value is m. All other cycles have at least some probability to go to a Maximizer state, and we already proved that all Maximizer states in T have value at least m. So s has no other choice but to almost surely reach a Maximizer state or an EC consisting only of Minimizer states, both of which have value of at least m. Thus, for all actions of s we have proven that their value in the modified SG is at least m.

Now we show that U is a fixpoint applying Bellman updates in the original SG G. If in G' a state depended on an action that is also available in G, applying one Bellman update will not change its value; this is because U was a fixpoint of applying Bellman updates in G'. Thus we only have to prove that all states that depended on an action added in the modified SG do not change their value upon applying a Bellman update. Firstly consider a Maximizer state $s \in T_{\square}$. If $\mathsf{U}(s) > m$, it would not depend on the newly added action, and hence we know that $\mathsf{U}(s) = m$. As T is an EC, s has a staying action a_s with $\mathsf{U}(s, a_s) \geq m$, because all states in T have U of at least m. We also know that $\mathsf{U}(s, a_s) = m$, because $\mathsf{U}(s) = m$, and Maximizer would have used an action yielding more than m if it was available. So for every Maximizer state that previously used one of the newly added actions, it is optimal to choose a staying action and the estimate after one Bellman update is still m. Secondly consider a Minimizer state $s \in T_{\bigcirc}$. We only added an action for those if they were part of some EC $T' \subseteq T_{\bigcirc}$. For all states $s' \in T'$ we have $\mathsf{U}(s') = m$. Hence, by playing actions that stay in T', Minimizer can achieve a U of m. It is the optimal choice for Minimizer to do this, because by assumption all exiting actions are worse (higher number) than m. In conclusion, applying a Bellman update on U in G will not change U, and thus it is a fixpoint.

Finally we prove that at least one state $s \in T$ has $\mathsf{U}(s) = m$. We already know $\mathsf{U}(s) \geq m$ for all $s \in T$. Assume for contradiction that $\mathsf{U}(s) > m$ for all $s \in T$. Then no state would depend on one of the newly added actions. Thus, the values in G and G' would be equal, and we would also have $\mathsf{V}_{\mathsf{G}}(s) > m$ for all $s \in T$. This is a contradiction, because the value of states in an EC can be at most $\mathsf{bestExit}^{\square}_{\mathsf{V}}(T) \leq m$. So we conclude that there exists a $s \in T$ with $\mathsf{U}(s) = m$. □

Corollary 1. *If* $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T) > \mathsf{bestExit}^{\square}_{\mathsf{V}}(T)$ *for some EC* T*, then* $\mathsf{G}^* \neq \mathsf{V}$.

Proof. There are m_1, m_2 such that $\mathsf{bestExit}^{\square}_{\mathsf{V}}(T) < m_1 < m_2 < \mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T)$. By Lemma 1, for both there exists a solution to the Bellman equations U_1 and U_2. The solutions are distinct, as there always exists at least one state where $\mathsf{U}_1(s) = m_1 < m_2 \leq \mathsf{U}_2(s)$.

Thus, there exist multiple fixpoints of Bellman updating. In particular, $\mathsf{G}^* > \mathsf{L}^* = \mathsf{V}$, and BVI does not converge. □

In accordance with our intuition that ECs satisfying the above inequality should be deflated, we call them bloated.

Definition 4 *(BEC).* An EC T is called a *bloated end component (BEC)*, if $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T) > \mathsf{bestExit}^{\square}_{\mathsf{V}}(T)$.

Example 2 *(BEC).* In the example of Fig. 3 on the left with $\alpha < \beta$, the ECs $\{p, q\}$ and $\{p, q, r\}$ are BECs. △

Example 3 *(Corner cases of* bestExit*).* If an EC T has no exiting actions of Minimizer (or no Minimizer's states at all, as in an MDP), then $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T) = 1$ (the case with $\min_{\emptyset}$). Hence all numbers between $\mathsf{bestExit}^{\square}_{\mathsf{V}}(T)$ and 1 are a solution to the Bellman equations and $\mathsf{G}^*(s) = 1$ for all states $s \in T$.

Analogously, if Maximizer does not have any exiting action in T, then it holds that $\mathsf{bestExit}^{\square}_{\mathsf{V}}(T) = 0$ (the case with $\max_{\emptyset}$), T is a BEC and all numbers between 0 and $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T)$ are a solution to the Bellman equations. △

Note that in MDPs all ECs belong to one player, namely Maximizer. Consequently, all ECs are BECs except for ECs where Maximizer has an exiting action with value 1; all other ECs thus have to be collapsed (or deflated) to ensure BVI convergence in MDPs. Interestingly, all non-trivial ECs in MDPs are a problem, while in SGs through the presence of the other player some ECs can converge, namely if both players want to exit, which we illustrate in the subsequent example.

Example 4 *(Unproblematic ECs).* If in the example of Fig. 3 on the left p had an exiting action with value $\gamma < \min(\alpha, \beta)$, then Minimizer's best strategy is to pick the leaving action and ensure the smaller value. Consequently, Maximizer realizes that going to p is suboptimal, as both α and β are larger than γ. Thus all states depend on exiting actions, we have no cyclic dependencies and BVI converges. △

Our first theorem states that BECs are indeed the only obstacle for BVI convergence. To prove it, we need the following lemma.

Lemma 2. *If a set $T \subseteq S$ does not contain an EC, then there exists a state $s \in T$, such that for all $a \in \mathsf{Av}(s)$ it holds that (s, a) exits T.*

Proof. Let $T \subseteq S$ be a set of states that does not contain an EC. Assume for contradiction that for all states $s \in T$ there exists an action a, such that $\neg(s, a)$ exits T. Then we can construct an EC as follows: let $T' = \{s\}$ for some state $s \in T$. Then, for all states in T' add all successors of the actions that do not exit T to T'. Repeat this until a fixpoint is reached. This must happen, as T is finite and the actions are not exiting, i.e. only have successors in T. Now compute the strongly connected components of T' with only the staying actions. There has to be a non-trivial strongly connected component, because T' is connected and finite, and every state has at least one outgoing edge to another state in T', so there exists a cycle. The strongly connected component forms an EC, which is a contradiction. Thus we know that for some state, all actions are leaving T. □

We will also sometimes use the contrapositive of this lemma, i.e. that if every state in a set T has at least one staying action, then T contains an EC.

Theorem 1 *(Convergence without BECs). If an SG contains no non-trivial BECs, i.e. no BECs in $S \setminus (F \cup \mathcal{Z})$, then $\mathsf{G}^* = \mathsf{V}$, i.e. value iteration from above converges to the value in the limit.*

Proof (*Structure*). We first give the general idea of the proof.

We assume towards a contradiction, that there is some state s with a positive difference $\mathsf{G}^*(s) - \mathsf{V}(s) > 0$. Then we will look at the set of states X that have the maximal difference and prove the following.

No state in X "depends on the outside". (5)

This means that for all states in X, the best action must not have successors outside of X. Best action means that it minimizes V for Minimizer states and maximizes G^* for Maximizer states.

Then, using a case distinction over the possible graph structures of X, we show that the only way to satisfy Statement (5) is that we have an EC $Z \subseteq X$ where the best action for every state is not exiting Z. However, this implies that Z is a BEC (and in fact even a *simple end component*, which will be introduced in the next section). This is a contradiction to the assumption that the game is BEC-free, and thus proves our goal.

Proof (*Complete*). We denote the difference in a state by $\Delta(s) := \mathsf{G}^*(s) - \mathsf{V}(s)$. We also use $\Delta(s, a)$, see Footnote 6. Assume for contradiction there exists a state $s \in S$ with positive difference $\Delta(s) > 0$.

Let $X := \{s \in S \mid \Delta(s) = \max_{s \in S} \Delta(s)\}$ denote the set of all states with maximal difference. Note that by assumption of there being a state with positive difference, X is non-empty and for all $s \in X$ we have $\Delta(s) > 0$. By definition, we require $\mathsf{V}(1) = \mathsf{G}^*(1) = 1$ for every $1 \in F$. Hence, $\Delta(1) = 0$ and thus, $F \cap X = \emptyset$. Analogously, it holds that $\mathcal{Z} \cap X = \emptyset$. Hence we know that $X \subseteq S \setminus (F \cup \mathcal{Z})$.

We will now prove that no state in X "depends on the outside", i.e.

$$\forall (s, a) \text{ exits } X : \begin{cases} \mathsf{G}^*(s, a) < \mathsf{G}^*(s) & \text{if } s \in X_\square \\ \mathsf{V}(s, a) > \mathsf{V}(s) & \text{if } s \in X_\bigcirc \end{cases} \tag{5}$$

To prove Statement (5), first observe the following:

$$\forall (s, a) \text{ exits } X : \Delta(s, a) < \Delta(s). \tag{6}$$

This holds, because if (s, a) exits X, then $\exists t \in \mathsf{Post}(s, a) \setminus X$. Since $t \notin X$, $\Delta(t) < \Delta(s)$. Then the following chain of equations proves Statement (6):

$$\begin{aligned} \Delta(s, a) &:= \mathsf{G}^*(s, a) - \mathsf{V}(s, a) && \text{(Definition of } \Delta) \\ &= \sum_{s' \in S} \delta(s, a, s') \cdot (\mathsf{G}^*(s') - \mathsf{V}(s')) && \text{(Definition of } \mathsf{V} \text{ and } \mathsf{G}\text{, pulling together the sums)} \\ &= \sum_{s' \in S} \delta(s, a, s') \cdot \Delta(s') && \text{(Definition of } \Delta) \\ &< \Delta(s) \end{aligned}$$

The last inequality holds, since $\delta(s, a, t) > 0$ and $\Delta(t) < \Delta(s)$, so there is one summand that is smaller than $\Delta(s)$; also, $\Delta(s)$ is the maximum difference, so there can be no larger summand to make up for the loss.

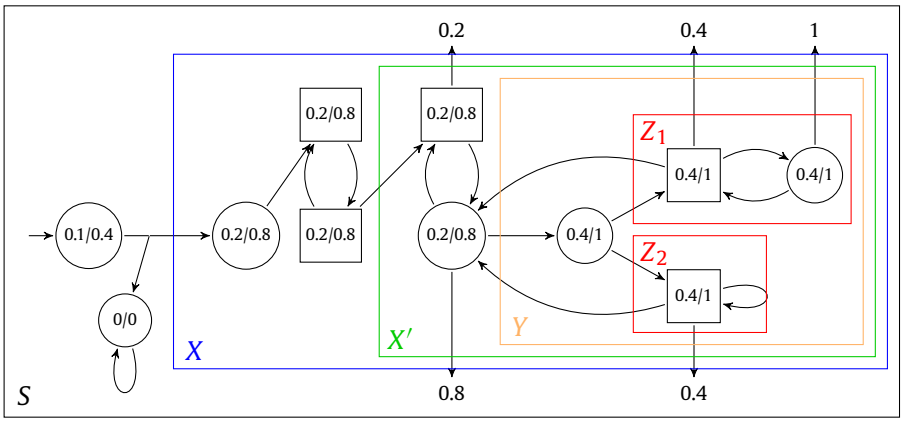


Fig. 4. An example of an SG where all the sets in the proof of Theorem 1 are different. Numbers on exiting arrows denote values, see Footnote 7. Every state is inscribed with V/G.

Now we can prove Statement (5) with the following chain of equations:

$$
\begin{aligned}
\mathsf{G}^*(s,a) - \mathsf{V}(s,a) &= \Delta(s,a) && \text{(Definition of } \Delta\text{)}\\
&< \Delta(s) && \text{(Using Statement (6))}\\
&= \mathsf{G}^*(s) - \mathsf{V}(s) && \text{(Definition of } \Delta\text{)}\\
&\leq \begin{cases} \mathsf{G}^*(s) - \mathsf{V}(s,a) & \text{if } s \in S_{\square} \\ \mathsf{G}^*(s,a) - \mathsf{V}(s) & \text{if } s \in S_{\bigcirc} \end{cases}
\end{aligned}
$$

The last inequality holds since for every available action a and for $s \in S_{\square} : \mathsf{V}(s) \geq \mathsf{V}(s,a)$, as Maximizer uses the maximum action, and dually for $s \in S_{\bigcirc} : \mathsf{G}^*(s) \leq \mathsf{G}^*(s,a)$. Simplifying the inequation yields Statement (5).

Now we make a case distinction over the possible graph structures of X and use Statement (5) and Lemma 2 to derive a contradiction.

- If X does not contain an EC,
  then by Lemma 2 there exists a state $s \in X$, such that for all $a \in \mathsf{Av}(s)$ we have (s,a) exits X. Thus, this state "depends on the outside", which by Statement (5) yields a contradiction. Formally, if $s \in X_{\square}$, we have that $\forall a \in \mathsf{Av}(s) : \mathsf{G}^*(s,a) < \mathsf{G}^*(s)$, because all actions exit X and Statement (5). However, by definition $\mathsf{G}^*(s) = \max_{a \in \mathsf{Av}(s)} \mathsf{G}^*(s,a)$, so there has to be an equality for some action. The argument works analogously for a Minimizer state $s \in X_{\bigcirc}$.
- If X contains an EC,
  we need to make further steps. States can have a large difference just by depending on a BEC. Our further steps will allow us to find a BEC inside X. See Fig. 4 for an example of a game where all the sets we introduce in the following are different.
  We now need the notion of bottom MEC. When talking about the whole game, bottom MECs are defined as MECs without leaving actions. However, we need the notion of bottom MEC *in* X. Formally, this requires that every successor of an action leaving the bottom MEC reaches a state outside of X with positive probability under all pairs of strategies. Intuitively, a bottom MEC in X is a MEC, such that after leaving it, there is no other MEC in X. They are computed by computing the MEC decomposition of X, ordering the MECs topologically and picking one at the end of a chain.
  Let $X' \subseteq X$ be a bottom MEC in X. X' exists, because by assumption X contains an EC, so there also is an EC that is bottom in X. If X' was a MEC at the end of the topological ordering of MECs in X, but not bottom, that would mean that there is a successor state s' of some action exiting X', such that there is a pair of strategies under which the play from s' surely remains in X. This would imply that s' can reach an EC inside X, which is a contradiction to the assumption that X' is at the end of the topological ordering of MECs in X.
  Let $m = \max_{s \in X'} \mathsf{G}^*(s)$ be the maximum upper bound in X', and let $Y := \{s \mid s \in X' \wedge \mathsf{G}^*(s) = m\}$ be the states with that maximum upper bound.
  $\forall s \in Y, \exists a \in \mathsf{Av}(s) : \neg(s,a)$ exits Y, i.e. all states in Y have actions that stay in Y. We prove this by a case distinction over where the actions of Y can exit to. Let $s \in Y$ be an arbitrary state. There has to be some action a_m with $\mathsf{G}^*(s,a_m) = \mathsf{G}^*(s) = m$. We say that action a_m exits X towards a set of states T, if (s,a_m) exits X and some successor of the action is in T.
  - Action a_m cannot exit towards $S \setminus X$. Otherwise, s would "depend on the outside" and we get a contradiction as in the case of X not containing an EC.

- No action can exit towards $X \setminus X'$, because X' is a bottom MEC in X. If an action left towards some state $t \in X \setminus X'$, then from t there would be no reachable EC in X. Hence, by Lemma 2, the states that t can reach must contain some state that only has actions exiting X, which yields a contradiction as in the case of X not containing an EC.
- a_m cannot exit towards $X' \setminus Y$, as by definition of Y all states $\mathsf{s}' \in X' \setminus Y$ have $\mathsf{G}^*(\mathsf{s}') < m$.
- The only remaining possibility is that a_m stays in Y.

Let $Z \subseteq Y$ be a bottom MEC in Y. Z exists, since by the previous step all states in Y have actions staying in Y, and by the contrapositive of Lemma 2 this implies the existence of a MEC in Y.

Note that for all states $\mathsf{s} \in Z$ we have $\mathsf{V}(\mathsf{s}) = m - c$, where $c = \max_{\mathsf{s} \in S} \Delta(\mathsf{s})$, as $Z \subseteq Y$ and $Z \subseteq X$. So in fact, Z is a *simple EC*, which will be introduced in the next subsection. This fact is relevant for the proof of Theorem 2.

We now prove that Z is a BEC by a case distinction on the bestExit of Maximizer.

- If $\mathsf{bestExit}^{\square}_{\mathsf{G}^*}(Z) > m$,
  then the Maximizer state s that has this exit would use it to get $\mathsf{G}^*(\mathsf{s}) > m$. However, we know that $\mathsf{G}^*(\mathsf{s}) = m$, so this is a contradiction.
- If $\mathsf{bestExit}^{\square}_{\mathsf{G}^*}(Z) = m$,
  then let $(\mathsf{s}, \mathsf{a}_\ell)$ be a state-action pair, such that $\mathsf{s} \in Z_{\square}$, $(\mathsf{s}, \mathsf{a}_\ell)$ exits Z and $\mathsf{G}^*(\mathsf{s}, \mathsf{a}_\ell) = \mathsf{G}^*(\mathsf{s}) = m$. This pair exists by the assumption that there is an exit of Maximizer from Z that achieves m. However, we showed earlier, that such an action a_ℓ can only stay in Y (note that $Z \subseteq Y$, thus the earlier case distinction fully applies). Since $(\mathsf{s}, \mathsf{a}_\ell)$ exits Z, we know that there is a successor of that state-action pair in $Y \setminus Z$. This is a contradiction, because Z is a bottom MEC in Y. The argumentation is analogous to that of an action exiting Y towards $X \setminus X'$.
- If $\mathsf{bestExit}^{\square}_{\mathsf{G}^*}(Z) < m$,
  first note that we are not done yet, because the case distinction talks about $\mathsf{bestExit}_{\mathsf{G}^*}$, while the definition of BEC uses $\mathsf{bestExit}_{\mathsf{V}}$. We still need to show that the true value of Minimizer's best exit is higher than the true value of Maximizer's one. There are two possibilities, depicted in Fig. 4 as Z_1 and Z_2.
  - If there are $\bigcirc$-exits,
    then let $(\mathsf{s}, \mathsf{a}_\ell)$ be the minimal $\bigcirc$-exit, i.e. the state-action pair with $\mathsf{s} \in Z_{\bigcirc}$ and $(\mathsf{s}, \mathsf{a}_\ell)$ exits Z with $\mathsf{V}(\mathsf{s}, \mathsf{a}_\ell) = \mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(Z)$. We know that $(\mathsf{s}, \mathsf{a}_\ell)$ cannot exit towards $X \setminus X'$ or $Y \setminus Z$, by the same bottom MEC argumentation as before. It cannot exit towards $X' \setminus Y$, because then $\mathsf{G}^*(\mathsf{s}, \mathsf{a}_\ell) < m = \mathsf{G}^*(\mathsf{s})$, but Minimizer uses the minimal available action. Thus, it exits towards $S \setminus X$, i.e. $(\mathsf{s}, \mathsf{a}_\ell)$ exits X. Then, by Statement (5) we know that $\mathsf{V}(\mathsf{s}, \mathsf{a}_\ell) > \mathsf{V}(\mathsf{s})$. Recall that all states in Z have the same value. Additionally, for all $\mathsf{s}' \in Z_{\square}, \mathsf{a}' \in \mathsf{Av}(\mathsf{s}')$ we have $\mathsf{V}(\mathsf{s}') \geq \mathsf{V}(\mathsf{s}', \mathsf{a}')$ since Maximizer picks the best action. Let $(\mathsf{s}', \mathsf{a}')$ be the state-action pair used for $\mathsf{bestExit}^{\square}_{\mathsf{V}}(Z)$. Then we know that Z is a BEC by the following inequation chain:
    $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(Z) = \mathsf{V}(\mathsf{s}, \mathsf{a}_\ell) > \mathsf{V}(\mathsf{s}) = \mathsf{V}(\mathsf{s}') \geq \mathsf{V}(\mathsf{s}', \mathsf{a}') = \mathsf{bestExit}^{\square}_{\mathsf{V}}(Z)$.
  - If there are no $\bigcirc$-exits,
    then $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}} = 1$, since $\min_{\emptyset} = 1$. Hence, by Lemma 1, $m = 1$. Recall that in the case distinction we assumed $\mathsf{bestExit}^{\square}_{\mathsf{G}^*}(Z) < m$. Thus Z is a BEC, since

    $$\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(Z) = 1 = m > \mathsf{bestExit}^{\square}_{\mathsf{G}^*}(Z) \geq \mathsf{bestExit}^{\square}_{\mathsf{V}}(Z).$$

  The fact that Z is a BEC is a contradiction to the assumption of the theorem that the SG contains no non-trivial BECs, as $Z \subseteq S \setminus (F \cup \mathcal{Z})$. Thus we arrive at a contradiction in all cases, and Theorem 1 is proven. □

In Section 4.2, we show how to eliminate BECs by collapsing their "core" parts, called below MSECs (maximal simple end components). Since MSECs can only be identified with enough information about V, Section 4.3 shows how to avoid direct *a priori* collapsing and instead dynamically deflate candidates for MSECs in a conservative way.

4.2. The core of the problem: simple end components

Now we turn our attention to SG with BECs. In a BEC all Minimizer's exiting actions have a higher value than what Maximizer can achieve; hence, Minimizer will not use exiting actions, but prefers staying in the EC and steering Maximizer towards his worse exiting actions. Consequently, only Maximizer wants to take an exiting action.

Intuitively, solving a BEC amounts to computing the attractors of Maximizer's exits, i.e. those states that can reach an exit if Minimizer plays optimally. All states in such an attractor have the same value, namely that of the best exit they can reach. Thus, we define the following sub-component.

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

A SEC T is *maximal (MSEC)* if there is no SEC T' such that $T \subsetneq T'$.

To give another intuition: an EC is simple, if Minimizer cannot keep Maximizer away from his bestExit. Independently of Minimizer's decisions, Maximizer can reach the bestExit almost surely, unless Minimizer decides to leave, in which case Maximizer would achieve an even higher value.

Algorithm 2 FIND_MSEC.

```
1: function FIND_MSEC(f : S → [0, 1])
2:     Av' ← Av
3:     for s ∈ S_○ do                    # Keep only optimal actions for Minimizer
4:         Av'(s) ← {a ∈ Av(s) | f(s, a) = min_{b∈Av(s)} f(s, b)}
5:     return MEC(G_[Av/Av'])   # MEC(G_[Av/Av']) are MSECs of the original G
```

In the MDP case, every EC is simple, as Maximizer can decide to use any exit. As a result, in MDPs all states of an EC have the *same value* and can all be collapsed into one state. In the SG case, Maximizer may be restricted by Minimizer's behaviour or even not given any chance to exit the EC at all. As a result, a BEC may contain several parts, each with different value, intuitively corresponding to different exits. Thus instead of MECs, we have to decompose into finer MSECs and only collapse these.

Example 5 *(SECs).* Assume $\alpha < \beta$ in the example of Fig. 3. Then $\{\mathsf{p}, \mathsf{q}\}$ is a SEC and an MSEC. Further observe that action c is sub-optimal for Minimizer and removing it does not affect the value of any state, but simplifies the graph structure. Namely, it destructs the whole EC into several (here only one) SECs and some non-EC states (here r). △

Lemma 3 *(SECs are the core of the problem). For a set $T \subseteq S$, the following two statements hold.*

1. *If T is a BEC, then there exists a $T' \subseteq T$ such that T' is a SEC.*
2. *If T is a SEC, then either T also is a BEC or $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T) = \mathsf{bestExit}^{\Box}_{\mathsf{V}}(T)$.*

Proof. 1. Let τ be an optimal strategy of Minimizer. If $T \subseteq S$ is a BEC, then $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T) > \mathsf{bestExit}^{\Box}_{\mathsf{V}}(T)$, so the optimal strategy of Minimizer does not contain any exiting actions, or formally for all (s, a) exits T with $\mathsf{s} \in T_{\bigcirc}$, we have $\tau(\mathsf{s})(\mathsf{a}) = 0$. Hence, every state in the MDP G^{τ} induced by fixing an optimal Minimizer strategy has an available staying action, and thus it still contains an EC $T' \subseteq T$, by the contrapositive of Lemma 2.
Note that T' can be a subset of T, if Minimizer restricts the game to a part of the EC, as in Example 5.

2. Let $T \subseteq S$ be a SEC. Note that our goal is equivalent to showing

$$\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T) \geq \mathsf{bestExit}^{\Box}_{\mathsf{V}}(T),$$

as that means either the two numbers are equal, or the best Minimizer exit is greater and thus T is a BEC.
If there is no Minimizer exit, we have $\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T) = 1 \geq \mathsf{bestExit}^{\Box}_{\mathsf{V}}(T)$, using the convention that $\min_{\emptyset} = 1$ and the fact that no probability can be greater than 1.
If on the other hand there is a Minimizer exit, then let (s, a) be the best Minimizer exit. Then the following chain of equations proves our goal:

$$\begin{aligned}
\mathsf{bestExit}^{\bigcirc}_{\mathsf{V}}(T) &= \mathsf{V}(\mathsf{s}, \mathsf{a}) && \text{(By } (\mathsf{s}, \mathsf{a}) \text{ being the best Minimizer exit)} \\
&\geq \mathsf{V}(\mathsf{s}) && \text{(By definition of } \mathsf{V}(\mathsf{s}) \text{ (Equation (1)))} \\
&= \mathsf{bestExit}^{\Box}_{\mathsf{V}}(T) && \text{(As } T \text{ is a SEC)}
\end{aligned}$$

□

So SECs are the core of the problem, as by Theorem 1 BECs are the reason for non-convergence, and by Lemma 3 every BEC contains a SEC. As all states in a SEC have the same value, they are easy to treat; we can even collapse them as in MDPs without changing the value.

Thus, we need to find SECs. Algorithm 2, called FIND_MSEC, shows how to compute the inclusion maximal MSECs. It returns the set of all MSECs if called with parameter V. However, as we want to compute the value and hence cannot assume to know it, later we also call this function with other parameters $f : S \to [0, 1]$.

The idea of the algorithm is the following: Minimizer actions with a non-minimum value cannot be part of any SEC and thus should be ignored when identifying SECs. Hence, we construct the set of available actions Av' which does not contain non-minimum Minimizer actions. (The previous example illustrates that ignoring these actions is indeed safe as it does not change the value of the game.) We denote the game G where the available actions Av are changed to the new available actions Av' (ignoring the Minimizer's sub-optimal ones) as $\mathsf{G}_{[\mathsf{Av}/\mathsf{Av}']}$. Once removed, Minimizer has no choices to affect the outcome of the game and thus every remaining EC is simple. Another intuition for this is that we fix Minimizer's strategy to be optimal according to f and then compute the ECs in the resulting MDP.

Lemma 4 *(Correctness of Algorithm 2). $T \in \mathsf{FIND\_MSEC}(\mathsf{V})$ if and only if T is an MSEC.*

Proof. Direction $\Rightarrow$ We want to show: $T \in \mathsf{FIND\_MSEC}(\mathsf{V}) \implies T$ is an MSEC. Let $T \in \mathsf{FIND\_MSEC}(\mathsf{V})$. So T is a MEC of the game where the mapping of available actions is Av'. Hence for all $\mathsf{s} \in T_{\bigcirc}, \mathsf{a} \in \mathsf{Av}'(\mathsf{s})$ we have $\mathsf{V}(\mathsf{s},\mathsf{a}) = \min_{b \in \mathsf{Av}(\mathsf{s})} \mathsf{V}(\mathsf{s},b) = \mathsf{V}(\mathsf{s})$. We now show that for all $\mathsf{s} \in T$ we have $\mathsf{V}(\mathsf{s}) = \mathsf{bestExit}^{\square}_{\mathsf{V}}(T)$.

- If there is no exit for Maximizer, then $\mathsf{bestExit}^{\square}_{\mathsf{V}}(T) = 0$, and all states in T have the value 0, since Minimizer can force the game to stay inside the EC forever.
- If there is an exiting state e for Maximizer, then also from each state in T there is a path using only states in T to e. For each state $\mathsf{s} \in T$ we can compute $\mathsf{V}(\mathsf{s})$ by recursively applying the Bellman equations. In each application, we choose an action that leads us closer to e, i.e. for t being the next state on the path to e, choose a such that $\mathsf{V}(\mathsf{s},\mathsf{a}) = \delta(\mathsf{s},\mathsf{a},t) \cdot \mathsf{V}(t) + \sum_{\mathsf{s}' \in \mathsf{Post}(\mathsf{s},\mathsf{a}) \setminus \{t\}} \delta(\mathsf{s},\mathsf{a},\mathsf{s}') \cdot \mathsf{V}(\mathsf{s}')$. Since for each state in T we have a path to e, we can replace every $\mathsf{V}(\mathsf{s}')$ in this way. By repeating this and thereby multiplying all the probabilities $\delta(\mathsf{s},\mathsf{a},t)$, the factor in front of the terms that do not contain $\mathsf{V}(e)$ approaches 0, and thus we get $\mathsf{V}(\mathsf{s},\mathsf{a}) = \mathsf{V}(e)$ for an arbitrary $\mathsf{s} \in T$ and a staying action a. Since we showed before that for Minimizer all actions have the same value, all Minimizer states have as value $\mathsf{V}(e)$. Maximizer can certainly achieve $\mathsf{V}(e)$ by picking the action with the maximal value. Maximizer cannot achieve more than $\mathsf{V}(e)$, because this is defined to be the best exit from the EC, and thus the best value that can be achieved from any state in the EC.

Since T is a MEC of the game with some available actions removed, it certainly is an EC in the original game. Thus, from this and the previous argument, we know that T is a SEC.

T is inclusion maximal, because if there was some $T' \supsetneq T$, such that T' is a SEC, then there exists some state $\mathsf{s} \in T' \setminus T$ with all staying actions of this state having the value $\mathsf{bestExit}^{\square}_{\mathsf{V}}(T)$. If it is a Minimizer's state, it cannot have a lower exit available, because otherwise T' would not be a SEC. Thus no staying action of s is removed by Line 4, and it should also be part of the MEC in the modified game. This contradicts the assumption that $\mathsf{s} \notin T$, and thus T is an MSEC.

Direction $\Leftarrow$ Let T be an MSEC. We need to show that T is a MEC of the game where the mapping of available actions is Av'.

We first show that T is an EC in the modified game. Since T is an EC in the original game, there exists a B such that for each $\mathsf{s} \in T, \mathsf{a} \in B \cap \mathsf{Av}(\mathsf{s})$ we do not have (s,a) exits T by the first condition of Definition 2. Moreover, $B \cap \mathsf{Av}(\mathsf{s})$ is non-empty, as otherwise there could not be a path starting in s and using only actions from B, see the second condition of Definition 2. Hence for all $\mathsf{s} \in T, \mathsf{a} \in B \cap \mathsf{Av}(\mathsf{s})$ the following holds.

$$
\begin{aligned}
\mathsf{V}(\mathsf{s},\mathsf{a}) &:= \sum_{s' \in S} \delta(\mathsf{s},\mathsf{a},\mathsf{s}') \cdot \mathsf{V}(\mathsf{s}') && \text{(by definition of } V(\mathsf{s},\mathsf{a}) \text{ (Equation (2)))} \\
&= \sum_{s' \in S} \delta(\mathsf{s},\mathsf{a},\mathsf{s}') \cdot \mathsf{bestExit}^{\square}_{\mathsf{V}}(T) && \text{(since } \mathsf{s}' \in T \text{ and } T \text{ is simple)} \\
&= \mathsf{bestExit}^{\square}_{\mathsf{V}}(T)
\end{aligned}
$$

So every action that stays in T (i.e. does not exit T), in particular every action in B, is not removed from the game, because for all $\mathsf{s} \in T, \mathsf{a} \in \mathsf{Av}(\mathsf{s}), \neg(\mathsf{s},\mathsf{a})\,\mathsf{exits}\,T$ we have $\mathsf{V}(\mathsf{s}) = \mathsf{bestExit}^{\square}_{\mathsf{V}}(T) = \mathsf{V}(\mathsf{s},\mathsf{a})$. The first equality comes from T being an MSEC, the second is what we have just shown.

Since no action in B is removed from the game and T is still an EC after the removal, T is inclusion maximal by the same argumentation as at the end of the $\Rightarrow$ case, and thus $T \in \mathsf{FIND\_MSEC}(\mathsf{V})$. □

Remark 1 *(Hypothetical algorithm with an oracle).* In Section 3, we have seen that collapsing MECs does not ensure BVI convergence on SGs. Collapsing does not preserve the values, since in BECs states with different values would be collapsed. Hence we only want to collapse MSECs, where the values are the same. If we collapse all MSECs of a game G, the resulting game G' does not contain any BECs (as by Lemma 3 every BEC contains a SEC), and hence by Theorem 1 BVI converges. Using a similar argument as the correctness of collapsing in MDPs [11], we could show that the values in G' are equivalent to those in G.

However, the difficulty with this algorithm is that it requires an oracle to compare values, for instance a sufficiently precise approximation of V. Consequently, we cannot pre-compute the MSECs, but have to find them while running BVI.

But since the approximations converge only in the limit we may never be able to conclude on simplicity of some ECs. For instance, if $\alpha = \beta$ in Fig. 3, and if the approximations converge at different speeds, then Algorithm 2 always outputs only a part of the EC, although the whole EC on $\{\mathsf{p},\mathsf{q},\mathsf{r}\}$ is simple.

4.3. Deflating simple end components

Since MSECs cannot be identified from approximations of V for sure, we refrain from collapsing[10] and instead only decrease the over-approximation in a conservative way. We call the method *deflating*, by which we mean decreasing the

[10] Our subsequent method can be combined with local collapsing whenever the lower and upper bounds on V are conclusive.

Algorithm 3 DEFLATE.

```
1: function DEFLATE(State set T, f : S → [0, 1])
2:     for s ∈ T do
3:         f(s) ← min(f(s), bestExit^□_f(T)) # Decrease the upper bound
4:     return f
```

Algorithm 4 UPDATE procedure for bounded value iteration on SG.

```
1: procedure UPDATE(L: S → [0, 1], U: S → [0, 1])
2:     L, U get updated according to Eq. (3) and (4) # Bellman updates
3:     for T ∈ FIND_MSEC(L) do        # Use lower bound to find ECs
4:         U ← DEFLATE(T, U)          # and deflate the upper bound there
```

upper bound of all states in an EC to its $\mathsf{bestExit}^{\square}_{\mathsf{U}}$, see Algorithm 3. The procedure DEFLATE (called on the current upper bound U_i) decreases this upper bound to the minimum possible value according to the current approximation and thus prevents states from only depending on each other, as in SECs.

Overall, we want to gradually approximate SECs and perform the corresponding adjustments, but we do not commit to any of the approximations by collapsing and thus definitively changing the game graph.

Lemma 5 *(*DEFLATE *is sound). For any $f : S \to [0,1]$ such that $f \geq \mathsf{V}$ (where $\geq$ is point-wise comparison) and any state-set $T \subseteq S$, it holds that* $\mathsf{DEFLATE}(T, f) \geq \mathsf{V}$.

Proof. Let $T \subseteq S$ and $f : S \to [0,1]$ be such that $f \geq \mathsf{V}$. We reformulate the goal to saying that for all states $\mathsf{s} \in T$ it holds that $\min(f(\mathsf{s}), \mathsf{bestExit}^{\square}_f(T)) \geq \mathsf{V}(\mathsf{s})$.
This is equivalent to our goal, because the change in Line 3 is the only change Algorithm 3 applies and because the comparison of the functions $\mathsf{DEFLATE}(T, f)$ and V is point-wise.

If in Line 3 of DEFLATE the expression $\min(f(\mathsf{s}), \mathsf{bestExit}^{\square}_f(T))$ gets evaluated to $f(\mathsf{s})$ or if $f(\mathsf{s}) = \mathsf{bestExit}^{\square}_f(T)$, by assumption of $f \geq \mathsf{V}$ the goal trivially holds.

If $f(\mathsf{s}) > \mathsf{bestExit}^{\square}_f(T)$, the following chain of equations proves our goal:

$$\begin{aligned}
\mathsf{DEFLATE}(T, f)(\mathsf{s}) &= \mathsf{bestExit}^{\square}_f(T) \\
&\geq \mathsf{bestExit}^{\square}_{\mathsf{V}}(T) && \text{(Since } f \geq \mathsf{V}\text{)} \\
&\geq \mathsf{V}(\mathsf{s}) && \text{(Since no state can achieve a greater value than the best exit)}
\end{aligned}$$

□

Using DEFLATE, we can define an improved UPDATE procedure which makes the BVI algorithm (Algorithm 1) converge. The difference to the standard update are the additional lines 3 and 4. After the usual Bellman updates in Line 2, we compute the MSEC candidates according to the current under-approximation in Line 3. Then for every MSEC candidate, the upper bound is deflated in Line 4. Since DEFLATE is sound, the upper bound always remains a correct over-approximation. Intuitively, the over-approximation converges, because L converges to the value in the limit, and thus we eventually find the correct MSECs and deflate them.

The main loop of Algorithm 1 using Algorithm 4 as UPDATE describes an operator $\mathcal{B} : ([0,1]^S, [0,1]^S) \to ([0,1]^S, [0,1]^S)$, i.e. given two functions from states to $[0,1]$, it computes two updated versions of these functions. In order to prove the correctness and termination of Algorithm 1 using Algorithm 4 as UPDATE, we first prove that the sequence of upper bounds converges to a fixpoint.

Lemma 6 *(Upper bound converges to a fixpoint). The limit of repeatedly applying the operator $\mathcal{B}$ to the initial lower and upper bound exists and is a fixpoint of the operator $\mathcal{B}$, i.e.* $\lim_{i\to\infty} \mathcal{B}^i(\mathsf{L}_0, \mathsf{U}_0) = \mathcal{B}\left(\lim_{i\to\infty} \mathcal{B}^i(\mathsf{L}_0, \mathsf{U}_0)\right)$.

Proof. On a high level, the proof amounts to phrasing the problem so that the Kleene fixpoint theorem is applicable. Concretely, we follow the proof of [27, 8.15 CPO Fixpoint Theorem I.], adjusting some details to fit our setting.

We consider the domain $[0,1]^{|S|} \times [0,1]^{|S|}$, i.e. every element consists of two functions from states to real numbers, the under- and over-approximation. We say $(\mathsf{L}_1, \mathsf{U}_1) \preceq (\mathsf{L}_2, \mathsf{U}_2)$ if and only if both $\mathsf{L}_1 \leq \mathsf{L}_2$ and $\mathsf{U}_1 \geq \mathsf{U}_2$ with component-wise comparison. Intuitively, an element on the right side of $\preceq$ is a more precise approximation. The bottom element of the domain is $(\vec{0}, \vec{1})$, where $\vec{a}$ denotes the function that assigns a to all states. We further restrict the domain to exclude elements of the domain that are trivially irrelevant for the computation. Concretely, we exclude all tuples (L, U) where $\mathsf{L}(\mathsf{s}) < 1$ for a target state $\mathsf{s} \in F$ or $\mathsf{U}(\mathsf{s}) > 0$ for a state with no path to the target $\mathsf{s} \in \mathcal{Z}$. Then the bottom element is $\bot = (\mathsf{L}_0, \mathsf{U}_0)$, where $\mathsf{L}_0(\mathsf{s})$ is 1 for target states and 0 everywhere else, and $\mathsf{U}_0(\mathsf{s})$ is 0 for states in $\mathcal{Z}$ and 1 everywhere else. Note that these are the vectors that we have before the first iteration of the main loop of Algorithm 4. The comparator $\preceq$ induces a complete

partial order over the domain, as we have a bottom element, and as every directed subset has a supremum; the latter claim holds, because $\preceq$ reduces to component-wise comparisons between real numbers from $[0, 1]$, where suprema exist. For more details on the definition of directed set and complete partial orders, we refer to [27, Definition 7.7] respectively [27, Definition 8.1].

We prove several facts about $\mathcal{B}$ that we need for the final argument. $\mathcal{B}$ first applies Bellman updates and then additionally deflates the over-approximation. Bellman updates are monotonic (see Section 2.3) and deflating is monotonic, as in Line 3 of Algorithm 3 we take the minimum of the current value and the best exit. Thus for all i and every pair of approximations (L, U) we have $\mathcal{B}^i(\mathsf{L}, \mathsf{U}) \preceq \mathcal{B}^{i+1}(\mathsf{L}, \mathsf{U})$. Intuitively, applying $\mathcal{B}$ can only make the approximations more precise. It follows that

$$\bot \preceq \mathcal{B}(\bot) \preceq \cdots \preceq \mathcal{B}^i(\bot) \preceq \mathcal{B}^{i+1}(\bot) \preceq \dots$$

form an ascending chain. Since the domain with $\preceq$ is a complete partial order, we know that

$$\lim_{i\to\infty} \mathcal{B}^i(\bot) = \sup_{i\geq 0} \mathcal{B}^i(\bot), \tag{7}$$

and that the supremum exists.

We now argue that $\mathcal{B}$ eventually is continuous. Bellman updates are continuous (see Section 2.3). Applying Bellman updates to the lower bound converges to the true value V, see e.g. [18]. Thus, after a finite number of steps n the optimal actions of Minimizer according to L do not change any more, because the lower approximation has converged enough such that L_n of every suboptimal action is larger than L_n of every optimal action. In case of multiple optimal actions, one of them might converge at slower speed, and only some optimal action is selected, not all of them. Still, the selected actions stay constant for every $n' \geq n$. This implies that we always find and deflate the same ECs. Thus deflating an EC T simplifies to applying a Bellman update on the whole T, updating every over-approximation to the best exit. Thus, for every $n' \geq n$, deflating is also continuous. Since we argue about behaviour in the limit, it suffices that $\mathcal{B}$ is eventually continuous.

Then we can conclude:

$$\begin{aligned}
\mathcal{B}(\lim_{i\to\infty} \mathcal{B}^i(\mathsf{L}_0, \mathsf{U}_0)) &= \mathcal{B}(\sup_{i\geq 0} \mathcal{B}^i(\bot)) && \text{(Definition of } \bot \text{ and Equation (10))}\\
&= \sup_{i\geq 0} \mathcal{B}(\mathcal{B}^i(\bot)) && \text{(The supremum is certainly larger than } \mathcal{B}^n\text{, thus } \mathcal{B} \text{ is continuous.)}\\
&= \sup_{i\geq 1} \mathcal{B}^i(\bot)\\
&= \sup_{i\geq 0} \mathcal{B}^i(\bot) && \text{(since } \bot \leq \mathcal{B}^n(\bot) \text{ for all } n\text{)}\\
&= \lim_{i\to\infty} \mathcal{B}^i(\mathsf{L}_0, \mathsf{U}_0) && \text{(Definition of } \bot \text{ and Equation (10))}
\end{aligned}$$

□

Remark 2. Lemma 6 was assumed without justification in the conference version of the paper, as it relies on standard methods from lattice theory and seemed obvious. However, there is one surprising complication: the operator $\mathcal{B}$ is not continuous in general. This is because the algorithm deflates SEC-candidates, and these candidates vary depending on the current under-approximation. A state might be part of a SEC-candidate for a less precise under-approximation, but not be part of a SEC-candidate for a more precise under-approximation. In the first case it is deflated and its upper bound decreased. In the second case, it is not deflated and its upper bound stays. This violates continuity, since the upper bound in the first case is more precise, even though the element of the domain was more precise in the second case. This is why we argue about eventual continuity in the proof of Lemma 6.

Theorem 2 *(Soundness and completeness). Algorithm 1, using Algorithm 4 as* UPDATE*, produces monotonic sequences* L *under- and* U *over-approximating* V*, and terminates for every* $\varepsilon > 0$.

Proof. We denote by L_i and U_i the lower/upper bound function after the i-th call of UPDATE. L_i and U_i are monotonic under- respectively over-approximations of V because they are updated via Bellman updates, which preserve monotonicity and the under-/over-approximating property (this can be shown by a simple induction), and from Lemma 5.

Note that UPDATE, FIND_MSEC and DEFLATE take finite time, as all the for-loops in the algorithms iterate over finite sets. So it remains to prove that the main loop of Algorithm 1 terminates, i.e. that for all $\varepsilon > 0$ there exists an n such that $\mathsf{U}_n(\mathsf{s}_0) - \mathsf{L}_n(\mathsf{s}_0) < \varepsilon$. It suffices to show that $\lim_{n\to\infty} \mathsf{U}_n - \mathsf{V} = 0$, because $\lim_{n\to\infty} \mathsf{L}_n = \mathsf{V}$ (from e.g. [18]).

In the following, let $\mathsf{U}^* := \lim_{n\to\infty} \mathsf{U}_n$ and $\Delta(\mathsf{s}) := \mathsf{U}^*(\mathsf{s}) - \mathsf{V}(\mathsf{s})$; we also use $\Delta(\mathsf{s}, \mathsf{a})$, see Footnote 6. Assume for contradiction that the algorithm does not converge, i.e. there exists a state with $\Delta(\mathsf{s}) > 0$.

The rest of the proof is structured as follows.

Step 1 From $\Delta(s) > 0$ we derive that there has to be a SEC Z by using an analogue of the proof of Theorem 1.
Step 2 Then we show that by applying one more iteration of the loop of Algorithm 1, a SEC $Z' \subseteq Z$ is found and deflated, thereby decreasing the upper bound.
Step 3 This is a contradiction, because by Lemma 6, U^* is a fixpoint. Thus we get that for all $s \in S$ we have $\Delta(s) = 0$ and hence convergence from above.

The full technical proof follows.

Step 1 We reuse parts of the proof of Theorem 1 to prove that there exists a SEC Z. Note that as before when proving Theorem 1, we have the assumption that there exists a state with $\Delta(s) > 0$.
The difference is, that instead of the greatest fixpoint G^* we use limit of the upper bound computed by our algorithm U^*. We prove the analogue of Statement (5).

$$\forall (s,a) \text{ exits } X : \begin{cases} U^*(s,a) < U^*(s) & \text{if } s \in X_\square \\ V(s,a) > V(s) & \text{if } s \in X_\bigcirc \end{cases}$$

The only argument that we need to adjust is that for the fact: $\forall s \in S_\square, a \in Av(s) : U^*(s) \geq U^*(s,a)$. For a Bellman update this property holds, as the upper bound of a state is always the maximum upper bound of all its actions.
The additional deflating does not violate the property. Deflating is only executed if $s \in Z$ for some EC Z that was found by the latest call of FIND_MSEC. It sets $U(s) = \text{bestExit}^\square_U(Z)$. For actions staying inside Z, all successors have the same upper bound, namely $\text{bestExit}^\square_U(Z)$. Thus all staying actions a satisfy $\text{bestExit}^\square_U(Z) = U(s) = U(s,a)$. No exiting action a can have $U(s,a) > \text{bestExit}^\square_U(Z)$, because then by definition of bestExit we would have $U(s,a) = \text{bestExit}^\square_U(Z)$. So all exiting actions satisfy $U(s) = \text{bestExit}^\square_U(Z) \geq U(s,a)$. As every application of deflating preserves the property, it also transfers to U^*.
Then, using the analogue of Statement (5) and Lemma 2, we can construct the sets X, X', Y and Z as before. All states in Z have the same value, as noted already in the proof of Theorem 1. This implies that Z is a SEC, as the only possible value for the states is $\text{bestExit}^\square_V(Z)$. If there is no Maximizer exit, the value of all states as well as $\text{bestExit}^\square_V(Z)$ is 0. If there is a Maximizer exit, let (s,a) be the best exit. Then $V(s) \geq V(s,a) = \text{bestExit}^\square_V(Z)$, and an EC cannot have a higher value than that of its best Maximizer exit.
Step 2 Applying one more iteration of the loop of Algorithm 1, i.e. calling UPDATE once more, finds an EC $Z' \subseteq Z$. This is the case, because after some finite number of steps L_i is close enough to V such that the following holds: $\forall s \in Z_\bigcirc, (s,a)$ exits $Z : L_i(s,a) > L_i(s)$. This will happen, because all $\bigcirc$-exits from Z lead towards $S \setminus X$, and hence by Statement (5), we have $V(s,a) > V(s)$. So from some point onward, for all $i' \geq i$, we have $L_{i'}(s,a) > V(s) \geq L_{i'}(s)$. Then, when computing FIND_MSEC($L_{i'}$), all actions exiting Z are removed and the MECs of the modified game are computed. This will result in finding an EC $Z' \subseteq Z$, because every state has to have at least one action, and all actions have to stay inside Z.
Z' can be a strict subset of Z, for example if the BEC looks as depicted in Fig. 3 and $\alpha = \beta$, but $L_{i'}(q,e) \neq L_{i'}(r,f)$ for all i', because they converge at different speeds. However, this poses no problem for the convergence of our algorithm; intuitively, after Z' is deflated, the states from $Z \setminus Z'$ adjust their upper bounds according to their best exit, as going to Z' is suboptimal.
When calling UPDATE once more on U^*, we can instantiate L with some L_i that is such that we find $Z' \subseteq Z$ (by previous arguments). Since Z' is returned by FIND_MSEC(L_i), the algorithm executes DEFLATE(Z', U^*). Then for all $s \in Z'$ we set $U^*_{\text{new}}(s) = \text{bestExit}^\square_{U^*}(Z')$, thereby decreasing the upper bound and yielding the contradiction $\text{bestExit}^\square_{U^*}(Z') < U^*(s)$, because of the following argument.
There have to be $\square$-exits. If there were no $\square$-exits in Z', then for all i we have $\text{bestExit}^\square_{U_i}(Z') = 0$ (the case of $\max_\emptyset$), and hence $\Delta(s)$ would be 0; however, since $Z' \subseteq X$, it has to hold that $\Delta(s) > 0$. Let (s_e, a_e) be one of the best $\square$-exits, i.e. $U^*(s_e,a_e) = \text{bestExit}^\square_{U^*}(Z')$. Using Statement (5), we know that $U^*(s_e,a_e) < U^*(s_e)$.
Step 3 Finally, we get $U^*_{\text{new}}(s_e) = \text{bestExit}^\square_{U^*}(Z') = U^*(s_e,a_e) < U^*(s_e)$. This is a contradiction, because by Lemma 6 U^* is a fixpoint, but applying one more update yields a smaller U^*_{new}.
Hence our assumption that there was some state with a positive difference is wrong, and thus BVI also converges from above. Thus, the algorithm will terminate for every $\varepsilon > 0$. $\square$

The story so far:

1. Using only Bellman updates on an over-approximation need not converge to the value (see Corollary 1).
2. Since maximal ECs can have non-constant values, instead we have to focus on maximal *simple* ECs. Collapsing those allows BVI to converge (see Remark 1).
3. As we cannot find maximal simple ECs without knowing a close enough approximation of the value, we identify candidates for simple ECs on the fly and gradually *deflate* them (see Algorithm 4).
4. This is correct, because deflating is sound, and it converges, because we eventually find the correct maximal simple ECs and deflate them (see Theorem 2).

Algorithm 5 Update procedure for the learning/BRTDP version of BVI on SG.

```
1:  procedure UPDATE(L: S → [0,1], U: S → [0,1])
    # Simulating
2:      ρ ← s_0
3:      repeat
4:          s' ← sampled as successor of the last state in ρ
5:          ρ ← ρ s'
6:      until s' ∈ F or SIMULATION_STUCK
    # Deflating
7:      if SIMULATION_STUCK then
8:          for T ∈ FIND_MSEC_ρ(L) do
9:              DEFLATE(T, U)
    # Bellman updates
10:     L, U get updated by Eq. (3) and (4) on all states s ∈ ρ
```

5. Learning-based algorithm

State spaces of systems in model checking often are very large, so that standard algorithms take too long; the state space might be so large that it doesn't even fit onto memory. To tackle this problem, we harvest our convergence results and BVI algorithm for SGs to extend the asynchronous learning-based approach of BRTDP (bounded real time dynamic programming) to SGs.

Asynchronous value iteration selects in each round a subset $T \subseteq S$ of states and performs the Bellman update in that round only on T. Consequently, it may speed up computation if "important" states are selected, i.e. states that have a high probability to be visited and where the current error still is large. However, using standard VI one does not know the current error and thus cannot estimate the importance of the states; moreover, if certain states are not selected infinitely often the lower bound may not even converge.

In the setting of bounded value iteration, the current error bound is known for each state and thus convergence can easily be enforced. This gave rise to algorithms such as BRTDP in the setting of stopping MDPs [51], where the states are selected as those that appear on a simulation run. Very similar is the adaptation for general MDP [11, Section 4.1].

These algorithms in principle still have access to every parameter of the model, e.g. the exact transition function for every state-action pair. The advantage is that they are based on *partial exploration*, i.e. they try to avoid working on the whole state space by only considering those states that were encountered in simulations. This allows to work on the "core" of the model, which typically is much smaller than the whole state space [41]. Still, in the worst case the whole state space might be explored. Note that there are variants of the algorithms which do not know the transition function, but only the minimum transition probability for MDPs [11, Section 4.2] and for SGs [6].

Algorithm 5 shows the pseudocode for the BRTDP version of UPDATE. After describing the algorithm, we illustrate it as well as its advantages on several examples. Then we explain how some parts of the algorithm can be improved. Afterwards, we talk about the differences between Algorithm 5 and its predecessor from [11], as well as advantages and disadvantages when compared to the deterministic Algorithm 4 and finally prove its correctness.

Description of the algorithm. First a run of the SG is simulated (Lines 2-6). It starts with the initial state and then repeatedly samples a successor of the last state in the current run. When sampling, the non-deterministic choice is resolved by picking the "most promising action". For a Minimizer state, this is the action with the lowest lower bound, for a Maximizer state dually it is the highest upper bound; in case of ties, the choice is resolved uniformly at random. The probabilistic choice can be resolved by sampling a successor according to the transition distribution; a different idea is described below. The simulation is stopped either when a target state is reached or when it is stuck (SIMULATION_STUCK) inside an EC that it cannot leave, see Example 6. Several ways to detect if a simulation is stuck are discussed after the examples. If the simulation was stuck in an EC, we need to deflate this EC to get new information (Line 7-9), see Example 7. Note that in the pseudocode, we wrote $\mathsf{FIND\_MSEC}_\rho(\mathsf{L})$ to indicate that we do not search for MSECs in the whole game, but only on the path we just constructed. The procedure uses the knowledge of the transition function to ensure that a cycle that occurred along the path is indeed an EC. Afterwards Bellman updates are executed on all states that were just simulated (Line 10). These updates happen asynchronously and backwards, which allows them to propagate information faster, see Example 8.

Example 6 shows why we need to check whether a simulation is stuck.

Example 6 *(Simulation).* Consider the SG in Fig. 2. The starting state s only has a single available action with a single successor. The next state t can choose between action b and c. Both currently have an upper bound of 1, so the action is picked randomly. Assume action c was selected; then the next state is sampled according to the distribution $\delta(\mathsf{t}, \mathsf{c})$. Let o be the next state in the simulation. If we just had the breaking condition $\mathsf{s}' \in F$, we would continue sampling infinitely. The additional condition to detect that the simulation is stuck in the EC $\{\mathsf{o}\}$ allows us to stop. Several ways to detect whether a simulation is stuck are discussed after the examples. △

Example 7 demonstrates when and how DEFLATE is executed.

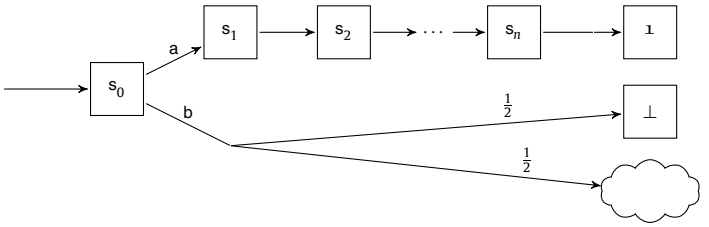


Fig. 5. An example of an SG to illustrate the advantages of BRTDP. The cloud symbolizes an arbitrarily large state space that need not be explored.

Example 7 *(Deflating).* Consider the SG in Fig. 2. When a simulation is stuck in the EC $\{o\}$, DEFLATE is executed on this EC. Since there is no exit, the upper bound is correctly set to 0.

If afterwards the simulation is stuck in the EC $\{s, t\}$, because $U(t, b) = 1 > \frac{2}{3} = U(t, c)$, then DEFLATE is executed on $\{s, t\}$, setting the upper bound of both states to the best current approximation $\mathsf{bestExit}^{\square}_{U}(\{s, t\}) = U(t, c) = \frac{2}{3}$.

In the next iteration, the simulation is stuck again as $U(t, b) = \frac{2}{3} > \frac{5}{9} = U(t, c)$, and thus DEFLATE is executed again. This is repeated until the required precision is reached. △

Example 8 shows that information can be propagated faster using asynchronous VI.

Example 8 *(Propagation of information).* Consider the top path in the SG in Fig. 5. To propagate the information that the target can be reached surely from s_n to s_0, normal VI needs $n + 1$ steps. After simulating the top path, BRTDP executes the Bellman updates in a backward manner, directly using the updated information: first $L(s_n)$ is set to 1; then $L(s_{n-1})$ is updated and can immediately use the information that $L(s_n) = 1$, and hence also is set to 1. Thus after one simulation of the top path and one update, BRTDP knows the result for this game, while normal VI needs n steps. This idea of immediately using the information available is part of asynchronous value iteration, and not directly linked to simulating paths; however, simulating the paths gives an easy and reasonable way to decide which states to update first. △

Example 9 shows how BRTDP can avoid exploring the whole state space.

Example 9 *(Avoid state space exploration).* Consider the bottom path in the SG in Fig. 5. The cloud symbolizes an arbitrarily large state space with arbitrary value. After sampling $\bot$ as successor once, $U(s_0, b) \leq 0.5$. Then b is never used in simulations from then on, as the upper bound of the other action is higher, and hence the other action is "more promising". So the cloud will most likely not be explored often, potentially (if $\bot$ is picked as successor the first time) never. This way, BRTDP can ignore billions of states, if they are not relevant for convergence. The value of s_0 certainly depends on action a, so the exact value of action b does not need to be computed. In contrast, normal VI and BVI have to remember the bounds for all the states symbolized by the cloud. △

Detecting SIMULATION_STUCK**.** We now discuss ways to detect that a simulation is stuck. A safe approach is to keep a partial model of all states of the current simulation and their successors, and check this partial model for ECs. This is similar to the idea of candidates in [26], but in contrast to that work it can utilize the full information about the model. However, as it is acceptable to have false positives (report stuck if we are not stuck), we can use heuristics such as stopping after the simulation has a certain length, which is a lot faster to check and easier than keeping track of a partial model. Several heuristics about when to stop are discussed and compared in [11,58]; in our own experiments, however, none of their suggestions performed as well as the simple idea of stopping the simulation when the length of the run becomes longer than twice the size of the currently explored state space. Even if deflation is not necessary in an EC and the simulation has a positive probability to exit, it might still loop several times. Hence the bound to stop the simulation should be greater than the explored state space, to allow for some looping. However, the approaches of [11,58] allowed to cycle for a very long time, thus sometimes wasting time before generating new information. For a more in-depth discussion of stopping heuristics, we refer the interested reader to [5].

Improvements for probabilistic choice. Another part of the algorithm that can be heuristically improved is how to resolve the probabilistic choice. Instead of picking the successor according to the transition distribution, we can weight the probabilities with the difference $U - L$. It was shown in [11,58] that this converges faster than just using the probabilities. This is the case, because states where we still know little (high difference between U and L) get more weight than states where we know a lot. In particular, states that have already converged ($U - L = 0$) are not assigned any weight. Note that it is important to not only consider the difference between the bounds, but also the transition distribution, as states that are reached with a high probability are more important than states that are only reached with a very small probability.

Comparison. Algorithm 5 is built upon the algorithm in [11]. The key differences are that (i) in SG there also is a minimizing player, who minimizes the lower bound instead of maximizing the upper bound and that (ii) in SG the approach of collapsing does not work in general, and hence we use deflating.

In contrast to using Algorithm 4 as UPDATE, BRTDP uses simulation and asynchronous partial updates and deflations. The (dis-)advantages of BRTDP on SG compared to BVI are the same as for the special case of MDP [11]: the backward updates propagate information faster and the exploration of parts of the state space can be avoided, as illustrated in Examples 8 and 9. This can save both time and memory. However, if large parts of the state space are relevant or if there are many ECs, simulations tend to perform worse than straightforward updates on the whole state space. Still, for models that are too large to load into memory, BRTDP is an option to get at least some information on the value.

Theorem 3 *(BRTDP soundness and completeness). Algorithm 1, using Algorithm 5 as* UPDATE*, produces monotonic sequences* L *under- and* U *over-approximating* V*, and terminates almost surely for every* $\varepsilon > 0$*.*

Proof. Soundness (L and U being monotonic sequences under- respectively over-approximating the value) follows from the same argument as in the proof for the deterministic algorithm. The fact that we only update parts of the state space does not affect the correctness of the Bellman update or deflation.

For completeness, the proof is similar to the one for MDPs [11, Theorem 3]. The difference is (i) that we also have to argue that Minimizer picks correct actions in the simulations by using the one with minimal lower bound and (ii) that instead of collapsing ECs we deflate them.

As before, we denote by L_i and U_i the under-/over-approximations after the i-th iteration of the main loop and show that $\mathsf{U}_\infty(\mathsf{s}_0) - \mathsf{L}_\infty(\mathsf{s}_0) = 0$, which implies that for every $\varepsilon > 0$ the algorithm terminates. Note that in contrast to the previous theorems, we only have to show that the approximations converge in the initial state, not in all states. We instead prove the claim that the approximations converge in all states that are sampled infinitely often by the simulations.

Let S_∞ be exactly those states that, when running the algorithm forever, are sampled infinitely often almost surely. Since every simulation starts in s_0, we have $\mathsf{s}_0 \in S_\infty$.

We call actions that are optimal according to U respectively L approx-optimal. Concretely, in iteration i an action in a Maximizer state is approx-optimal if it has the highest upper bound U_i; and dually in a Minimizer state if it has the lowest lower bound L_i. Every approx-optimal action has a positive probability to be chosen, and every successor of such an action has a positive probability to be sampled. Thus, when considering a state $\mathsf{s} \in S_\infty$ and an action a that is approx-optimal infinitely often, this action is picked infinitely often almost surely; further, all successors are sampled infinitely often almost surely, thus we have $\mathsf{Post}(\mathsf{s}, \mathsf{a}) \subseteq S_\infty$. Note that actions that are not picked infinitely often are detected as not being approx-optimal in some iteration, cf. Example 9.

We know the approximations for all states in S_∞ are updated infinitely often almost surely. Thus we can use the same argument as in the proofs of Theorems 1 and 2: assume there was a state $\mathsf{s} \in S_\infty$ with a positive difference $\mathsf{U}_\infty(\mathsf{s}) - \mathsf{L}_\infty(\mathsf{s}) > 0$. Then we can consider the set of states $X \subseteq S_\infty$ with maximal difference, where again we know that no state may "depend on the outside": since the sampling picks the optimal actions (according to U_∞ respectively L_∞ for Maximizer/Minimizer states) and every state in S_∞ is updated infinitely often, we know that no optimal action can lead outside of X. Thus, by the same case distinction as in Theorem 1, we have that X has to contain a bottom MEC Z. However, since $X \subseteq S_\infty$, Z is almost surely sampled, detected and deflated. If Z has no exit, its upper bound becomes 0, decreasing the difference to 0; if Z has an exit, it has to exit towards a state outside of X, thereby again "depending on the outside" and decreasing the upper bound. Thus we get the contradiction to the assumption that there was a state $\mathsf{s} \in S_\infty$ with a positive difference and can conclude that the approximations converge for all states that are sampled infinitely often, in particular the initial state. □

6. Experimental results

In the following, we present our experimental results on how these theoretical improvements perform practically on MDPs and on SGs. After describing the used models, tools and the technical setup, we report on some optimizations which we considered. Next, we present our results on MDPs and finally on SGs. Note that we include tests on MDPs since there are not so many SG models available for benchmarking, and since it is interesting to see how the more general approach of deflating compares to the more specialized collapsing.

6.1. Models and tools

We implemented both our algorithms as an extension of PRISM-games [43], a branch of PRISM [44] that allows for modelling SGs, utilizing previous work of [11,58] for MDP and SG with single-player ECs, respectively.

We tested the implementation on the SGs from the PRISM-games case studies[11] that have reachability properties and one additional model from [13] (thus using all models with reachability properties that were used in [58]). We compared the results with both the explicit and the hybrid engine of PRISM-games, but since the models are small both of them performed similar and we only display the results of the faster hybrid engine in Table 4.

Furthermore we ran experiments on MDPs from the PRISM benchmark suite [45] and on the adversarial example from [33]. We compared our results there to the hybrid and explicit engine of PRISM 4.5, the interval iteration of [33]

[11] prismmodelchecker.org/games/casestudies.php.

Table 1
Verification times (in seconds) for both our algorithms, BVI and BRTDP, applied to the model cloud with N=6, using various values of i for Optimization (III). An X in the table indicates a timeout.

	1	10	100	1000
BVI	X	123	123	117
BRTDP	11	12	25	255

as implemented in PRISM, the BRTDP implementation of [11] and the hybrid engine of Storm 1.3.0 [28], which uses topological value iteration. All the models, MDPs as well as SGs, are described in Appendix C.

Technical setup. All of the experiments were conducted on a server with 256 GB RAM and 2 Intel(R) Xeon(R) E5-2630 v4 2.20 GHz processors. We limited the computation to one core to avoid results being incomparable due to different times spent parallelizing. All model checkers worked at a precision of $\varepsilon = 10^{-6}$. Each experiment had a timeout of 15 minutes, which is represented by an X in the tables. We set the available Java memory to 16 GB. Still, the largest versions of csma and mer could not be loaded with the explicit engine of PRISM. Since the learning-based approaches are randomized, we took the average of 20 repetitions of the experiments.

6.2. Optimizations

We considered the following optimizations:

(I) collapsing of SECs when we are certain that it is safe;
(II) preferring exiting actions during the simulation in BRTDP;
(III) executing FIND_MSEC and DEFLATE only every i steps for various i;
(IV) caching the MECs of the SG and restricting the SEC computation to them.

When used with the learning-based algorithm, optimizations (I) and (II) reduce the length of simulations. As an example, consider the SG in Fig. 1: after DEFLATE was executed on the EC $\{p, q\}$, both action b and c have the same upper bound. So in state q, a simulation randomizes uniformly between picking b and c, which means that there is a positive probability to cycle through the EC again. Since we are only interested in runs that leave the EC, we want to minimize the time that is spent cycling through an EC again.

Optimization (I) does this by collapsing ECs where we are certain that collapsing is safe, i.e. that the EC is a SEC. Collapsing a SEC intuitively means replacing it with a single state that has only the exiting actions available, thereby removing the possibility to loop inside the EC. Thus we always use the best leaving action immediately. Extending the collapsing from MDPs to SGs requires some technical changes, which are explained in Appendix B.

If it is not clear how to collapse an EC, e.g. if it looks like in Fig. 3, we can still avoid cycling by using Optimization (II), i.e. preferring exiting actions if possible. For example, if in Fig. 3 the actions d and f for state r have the same value, we prefer f.

Our implementation of optimization (I) and (II) did not result in a significant speed up, because both checking whether an EC can really be collapsed as well as checking in each step whether some action is exiting was about as costly as the gain it brought in our implementation. However, with a better implementation these optimizations are promising, as they ensure that there no longer is a probability for simulations to cycle in an EC.

Optimization (I) can also be used for the deterministic algorithm, but our experiments did not show a significant speed up there as well.

Since finding all MSEC candidates is a costly operation, Optimization (III) decreases the number of times that FIND_MSEC and DEFLATE are executed, by only calling the procedures every i steps for some i. Optimization (III) had the largest impact on the model cloud (with scaling parameter $N = 6$), as depicted in Table 1. For $i = 1$, BVI was not able to finish within 15 minutes, while for the other i it was done within approximately 2 minutes. Looking at BRTDP on this model, we see that increasing the i increased the verification time from 11 seconds for $i = 1$ to 255 for $i = 1000$. Note that depending on the combination of algorithm and model, both a small and a large i can be advantageous. Probably the number of MSECs as well as their position in the model play a role. On the one hand, if i is too small, time is wasted to repeatedly compute the current MSEC decomposition. On the other hand, if i is too large, then we might need to wait for i iterations of the main loop before we can deflate some MSEC. As in our experiments both very small and very large i (1 or 1000) brought the possibility of a huge increase in verification time, for all following experiments we chose $i = 10$. This number always yielded a verification time that was very close to the optimum we could achieve.

Finally, Optimization (IV) is a technical improvement that speeds up the computation of SECs. In Algorithm 2, we compute the MECs of the modified SG with only optimal Minimizer actions. However, note that any MEC in the modified SG is an EC in the original SG, and hence a subset of a MEC in the original SG. Thus, we do not have to look for MECs in the whole modified SG, but it suffices to perform the computation on the MECs of the original SG. By caching these original

Table 2
CPU time for each MDP experiment in seconds. For each model, there first is a row giving the name of the model and the parameter(s), and then several rows for the different parameter values we tried. We compared our deterministic and learning based approaches (BVI and BRTDP) to the deterministic and learning based approaches based on collapsing ([33] and [11]). We also compared to the explicit engine of PRISM (PRISM_e), and the hybrid engines of PRISM (PRISM_h) and the topological value iteration of Storm, all three of which are without guarantees. X indicates a timeout, W that the returned result was wrong.

Model (scaling-parameters)	With guarantees				Without guarantees		
	Deterministic		Learning-based				
	BVI	[33]	BRTDP	[11]	PRISM_e	PRISM_h	Storm
firewire (dl)							
220	418	389	5	4	389	202	125
240	634	512	5	4	491	346	163
260	894	621	5	4	617	695	186
wlan (k, COL)							
4, 2	20	17	5	4	18	12	3
4, 6	36	34	5	3	34	16	7
6, 2	715	691	5	4	760	128	53
6, 6	709	701	5	5	773	134	53
zeroconf (K, N)							
2, 20	14	9	3	7	9	7	1
2, 1000	14	10	8	11	9	7	1
10, 20	254	233	7	8	163	133	26
10, 1000	282	170	8	10	155	146	27
csma (N, K)							
2, 2	1	1	7	2	1	<1	<1
2, 6	5	3	93	39	3	1	<1
3, 2	4	3	22	12	3	1	<1
3, 6	X	X	X	X	X	47	X
leader (N)							
3	3	2	4	4	1	1	<1
4	5	3	10	9	1	1	<1
5	8	6	17	28	2	2	<1
6	20	14	50	X	8	8	3
mer (x, n)							
10^{-4}, 1500	X	X	73	15	629	83	145
10^{-4}, 3000	X	X	70	13	X	172	516
0.1, 1500	X	X	X	X	571	115	146
0.1, 3000	X	X	X	X	X	235	513
hm (N, p)							
20, 0.5	70	33	X	604	W	W	W
20, 0.9	67	33	X	585	W	W	W

MECs, we can speed up the computation drastically, especially since typically MECs are small and many states are not part of any MEC.

6.3. MDP results

We compare seven different approaches to compute the reachability probability as well as optimal strategies on MDPs. We group these algorithms as follows.

Value iteration with guarantees. There are four guaranteed value iteration approaches – two of them based on collapsing ([11] and [33]) and two based on the new idea of deflating (BRTDP and BVI). Comparing columns 2 and 3 as well as 4 and 5 of Table 2, we see that both the deterministic (BVI and [33]) and both the learning-based (BRTDP and [11]) approaches perform similarly. We conclude that collapsing and deflating are both useful for practical purposes. The collapsing based approaches are usually slightly faster. This might be the case, because after an EC is collapsed, it will not have to be considered again, while for our new approach it is deflated every time. However, the difference might also depend on other implementation details, for example the different implementations of SIMULATION_STUCK between the two learning-based approaches.

Value iteration without guarantees. We compare BVI to the usual (unguaranteed) value iteration of PRISM's explicit engine (column 6 of Table 2) and see that the guaranteed approach did not take significantly more time in most cases. In all models but zeroconf for K=10 and mer for n=1500, PRISM_e and BVI produce times in the same order of magnitude. This implies that the overhead for the computation of the guarantees often is negligible.

Table 3
The number of states for each model and the number of states that the two simulation based approaches of [11] and of this paper (BRTDP) explored. Models and their scaling parameters are denoted on the left as in Table 2. An X indicates a timeout.

Model (scaling-parameters)	#States	[11]	BRTDP
firewire (dl)			
220	10,490,495	792	751
240	13,366,666	779	702
260	15,255,584	791	596
wlan (k, COL)			
4, 2	345,118	767	199
4, 6	728,990	764	333
6, 2	5,007,666	858	116
6, 6	5,007,670	691	133
zeroconf (K, N)			
2, 20	89,586	393	125
2, 1000	89,586	1,625	898
10, 20	3,001,911	1,161	599
10, 1000	3,001,911	5,358	782
csma (N, K)			
2, 2	1,038	964	965
2, 6	66,718	64,341	33,413
3, 2	36,850	22,883	26,650
3, 6	84,856,004	X	X
leader (N)			
3	364	335	313
4	3,172	2,789	2,599
5	27,299	21,550	8,281
6	237,656	128,593	22,368
mer (x, n)			
10^{-4}, 1500	8,862,064	2,603	2,032
10^{-4}, 3000	17,722,564	2,632	2,028
0.1, 1500	8,862,064	X	X
0.1, 3000	17,722,564	X	X
hm (N, p)			
20, 0.5	41	41	X
20, 0.9	41	41	X

Note that for hm the approaches without guarantees return the wrong result (denoted by "W" in the table), e.g. PRISM_e reports 0.35 when the true result is 0.9. The model checker does not print a warning that this might have happened.

Hybrid approaches. Compared to the hybrid engine of Storm and PRISM (columns 7 and 8 in Table 2), BVI is vastly outperformed on larger models; however, this difference is because BVI explicitly constructs the whole model, while the hybrid engines can avoid this using symbolic representations. An implementation of BVI using the hybrid engine would most probably also get the speed up of this approach, and hence again be comparable. Looking at the differences between PRISM's explicit and hybrid engine we see that the gain of the different engine is much larger than the overhead for computing the guarantees.

Simulation-based approaches. The simulation based approaches BRTDP and [11] perform well on firewire, wlan and zeroconf, even outperforming Storm in some cases. For firewire, they are two orders of magnitude faster. So in certain cases, simulation based approaches can produce a huge speed-up while still giving guarantees, as already noted in [11]. However for csma, leader and mer they are not well suited, as they need to explore thousands of states to achieve convergence. For the first two rows of mer they are still faster, since the explored part of the state space is very small in comparison to the whole model and not too large in general, but as the model is scaled, the number of relevant states grows too large for the simulation based approaches to work well. In hm, the adversarial handcrafted model, every value iteration algorithm is slow due to the slow convergence of the values. Moreover, the simulation based approaches have an additional problem that increases their runtime: every time a simulation starts in the initial state, it has a high probability, around 99.99% for our choice of N, to lead back to the initial state and not reach a target or a sink. Thus, simulations are often stopped by SIMULATION_STUCK without making any progress. Intuitively, the algorithm assumes that it does not reach a target because of an EC, while in fact it is only due to the low probability to reach the goal.

The results in Table 3 show that [11] explores a larger portion of the state space for almost all experiments. This is due to the different choice of the heuristic SIMULATION_STUCK, i.e. the number of steps before a simulation is stopped. The implementation in [11] allows for simulating longer, and hence more of the state space is explored. Depending on the model

Table 4
Experimental results for the experiments on SGs. Models and their scaling parameters are denoted on the left as in Table 2. Columns 2, 3 and 4 display the verification time in seconds for each of the solvers, namely PRISM-games (referred as PRISM), our deterministic algorithm (BVI) and our learning-based algorithm (BRTDP). The next two columns compare the number of states that BRTDP explored (#States_B) to the total number of states in the model. The rightmost column shows the number of MSECs in the model. An X indicates a timeout.

Model	Model checking times			Model properties		
(scaling-parameters)	PRISM	BVI	BRTDP	#States_B	#States	#MSECs
mdsm (prop)						
1	10	14	20	895	62,245	1
2	6	9	28	471	62,245	1
cdmsn						
	3	4	9	1,213	1,240	1
teamform (N)						
3	5	7	102	7,337	12,476	0
4	10	14	X	X	96,666	0
cloud (N)						
5	6	15	19	1,311	8,842	4,421
6	9	123	12	768	34,954	17,477

structure, this can be advantageous or detrimental for the verification time. In hm, the longer simulations are necessary to have a good chance of reaching a target or sink and making progress, so here [11] outperforms BRTDP. In contrast, in leader with N=6 [11] explores 128,593 states without producing a result, while BRTDP can terminate after seeing only 22,368 states. Here, stopping a simulation early prevents the algorithm from wasting time exploring irrelevant parts of the state space or cycling in ECs for a long time.

In general one can see, that the approach of deflating works well also on MDPs and that giving guarantees often is possible without significant overhead.

6.4. SG results

Since we gave the first guaranteed value iteration approach, we performed experiments to empirically estimate the overhead for the upper bound computation. We compared both our algorithms, the deterministic BVI and the learning-based BRTDP, to the hybrid engine of PRISM-games; note that the latter does not give any guarantees on the value. The results in Table 4 show that on all our benchmarks at least one of our approaches was close to the time that the hybrid engine of PRISM-games needed. The amount of times deflate has to be executed grows when there are many MSECs, as for example in cloud. This explains the longer runtime that BVI has, especially for cloud with N=6. Luckily, for this model apparently only a small part of the state space is relevant for convergence, and thus BRTDP performs well. On the other hand, for cdmsn and teamform a large part of the state space is relevant, so that BRTDP is slow in comparison and does not even finish in time for teamform with N=4.

In conclusion, we see that deflating can be very costly, but the cost often is negligible or can be mitigated by using the learning-based approach.

7. Conclusions

We have provided the first stopping criterion for value iteration on simple stochastic games and an anytime algorithm with bounds on the current error and thus guarantees on the precision of the result. The main technical challenge was that states in end components in SGs can have different values, in contrast to the case of MDPs. We have shown that collapsing is in general not possible, but we utilized the analysis of the game graph to obtain the procedure of *deflating*, a solution on the original graph. Besides, whenever a simple end component is identified for sure it can be collapsed and the two techniques of collapsing and deflating can thus be combined.

The experiments indicate that the price to pay for the overhead to compute the error bound is often negligible. For each of the experimental models, at least one of our two implementations has performed similar to or better than the standard approach that yields no guarantees. Further, the obtained guarantees facilitate (e.g. learning-based) heuristics which treat only a part of the state space and can thus potentially lead to huge improvements. Surprisingly, already our straightforward adaptation of a learning-based algorithm from MDP to SG yields good results, sometimes palliating the overhead of our non-learning method, despite the most naive implementation of deflating. Future work could reveal whether other heuristics or more efficient implementation can lead to huge savings as in the case of MDP [11].

The concept of simple end components has already been generalized to settings with limited information [6], with multiple objectives [4] and to concurrent stochastic games [30]. It is probable that it can also be applied in settings with

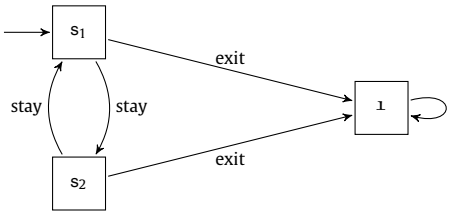


Fig. A.6. An example of an SG (also MDP) where following (ε-)optimal actions is not necessarily an (ε-)optimal strategy.

other objectives, e.g. expected reward or parity. To this end, the relationship between simple end components and tangles in parity games [29] should also be investigated.

Declaration of competing interest

This research was funded in part by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement No. 291763 for TUM – IAS, the Studienstiftung des Deutschen Volkes project "Formal methods for analysis of attack-defence diagrams", TUM IGSSE Grant 10.06 (PARSEC), and the German Research Foundation (DFG) project KR 4890/2-1 "Statistical Unbounded Verification".

Appendix A. Obtaining ε-optimal strategies

In this appendix, we show how to obtain ε-optimal strategies, given an SG $\mathsf{G} = (S, S_\Box, S_\bigcirc, \mathsf{s}_0, \mathsf{A}, \mathsf{Av}, \delta)$ and vectors of under- and over approximants of the value, L respectively U. Intuitively, the strategy just plays an action which is optimal according to the approximants. However, this might not be sufficient, as the following counter-example shows.

Consider the SG (that also is an MDP) from Fig. A.6. After one update we have $\mathsf{L}(\mathsf{s}_1) = \mathsf{L}(\mathsf{s}_2) = 1$. Thus, $\arg\max_{\mathsf{a}\in\mathsf{Av}(\mathsf{s}_i)} \mathsf{L}(\mathsf{s}_i, \mathsf{a}) = \{\mathit{stay}, \mathit{exit}\}$ and L is ε-close to the true value (in fact it is equal). Both actions are indeed optimal, as after playing *stay*, the other state still can reach the target with probability 1. However, using the strategy that in both states picks the *stay*-action results in probability 0 to reach the target. Intuitively, the problem is that picking ε-optimal actions only guarantees that they are one-step ε-optimal, but does not ensure that the target is actually reached in the end. Thus, in order to obtain ε-optimal strategies, we additionally have to ensure that the strategies "make progress" towards the target.

A way to achieve this would be to randomize over all ε-optimal actions; however, this is suboptimal, as theoretically it needs the additional power of randomization and practically the strategy has a chance of cycling unnecessarily by using the actions that stay instead of making progress.

In [7, Remark 10.104], the authors observe that L_i corresponds to the i-step-bounded reachability probability (for MDPs, but the argument extends to SGs). They suggest that a strategy obtaining this value can be generated by choosing an optimal action according to L_i in the first state, then according to L_{i-1} in the next step and so on.[12] However, note that this strategy requires memory as well as knowledge of all intermediate results L_i.

We now describe a way to obtain a memoryless deterministic ε-optimal strategy for Maximizer and Minimizer, using only the final approximants L respectively U. We also prove the correctness of our strategies. Concretely, we want to find a σ such that for all τ: $\mathbb{P}^{\sigma,\tau}_{\mathsf{s}_0}(\Diamond F) > \mathsf{V}(\mathsf{s}_0) - \varepsilon$. Dually we also want a τ such that for all σ: $\mathbb{P}^{\sigma,\tau}_{\mathsf{s}_0}(\Diamond F) < \mathsf{V}(\mathsf{s}_0) + \varepsilon$. Intuitively, for the Minimizer it suffices to just play actions which are optimal according U. For the Maximizer, we play optimally according to L, but additionally have to ensure that the actions make progress; we achieve this by performing a backwards search from the target states and picking only those actions that decrease the distance to the targets.

A.1. Assumptions on the approximants

We first state several assumptions which the approximants have to satisfy and for each assumption prove that it is satisfied for approximants coming from our algorithm. For ease of notation, we use the operator $\arg\operatorname{opt}$ to find the set of optimal actions for a state s:

$$\arg\operatorname*{opt}_{\mathsf{a}\in\mathsf{Av}(\mathsf{s})} := \begin{cases} \arg\max_{\mathsf{a}\in\mathsf{Av}(\mathsf{s})} & \text{if } \mathsf{s} \in S_\Box \\ \arg\min_{\mathsf{a}\in\mathsf{Av}(\mathsf{s})} & \text{if } \mathsf{s} \in S_\bigcirc \end{cases}$$

- The approximants are correct, i.e. for all states $\mathsf{L}(\mathsf{s}) \le \mathsf{V}(\mathsf{s}) \le \mathsf{U}(\mathsf{s})$. This assumption is satisfied for our algorithm by Theorem 2.

[12] In the previous example, for all $i > 1$, both *stay* and *exit* are optimal, while for $i = 1$ only *exit* is optimal. Thus, this strategy ensures that the target is reached.

- The approximants are ε-close in the initial state, i.e. $\mathsf{V}(\mathsf{s}_0) - \mathsf{L}(\mathsf{s}_0) < \varepsilon$ and $\mathsf{U}(\mathsf{s}_0) - \mathsf{V}(\mathsf{s}_0) < \varepsilon$. This is satisfied for our algorithm, as the bounds are correct and the stopping criterion ensures $\mathsf{U}(\mathsf{s}_0) - \mathsf{L}(\mathsf{s}_0) < \varepsilon$.
- For all target states the lower approximant is 1. This assumption is satisfied by the initialization of the algorithm.
- Unfolding the Bellman equation for the approximants can only make them more precise, i.e. for all states s:

$$\mathsf{L}(\mathsf{s}) \leq \operatorname*{arg\,opt}_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{L}(\mathsf{s}')$$

$$\mathsf{U}(\mathsf{s}) \geq \operatorname*{arg\,opt}_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{U}(\mathsf{s}')$$

We first prove this assumption for the lower approximants. Let i be the iteration in which the algorithm computing the approximants finished. We denote by L_j the approximants in the j-th iteration for all $j \leq i$. Then we have $\mathsf{L} = \mathsf{L}_i$ by definition. The lower approximant is computed according to the Bellman equations (3) and (4). Thus for all states s we have

$$\begin{aligned}\mathsf{L}_i(\mathsf{s}) &:= \operatorname*{arg\,opt}_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{L}_{i-1}(\mathsf{s}') \\ &\leq \operatorname*{arg\,opt}_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{L}_i(\mathsf{s}'),\end{aligned}$$

where the second inequality follows the fact that the bounds are monotonically increasing, i.e. $\mathsf{L}_{i-1}(\mathsf{s}) \leq \mathsf{L}_i(\mathsf{s})$.
For the upper approximant, the proof is mostly analogous, replacing $\leq$ with $\geq$. However, we also have to address the additional deflating that can modify the upper approximants. If a state s is deflated down to x, then it is part of an EC T and all states in the T were deflated to x as well. As T is an EC, every $\mathsf{s} \in T$ has a staying action a_s. Since all successors of this action are in the EC and have an upper estimate of x, we have $\mathsf{U}(\mathsf{s}, \mathsf{a}_s) = x$. Hence, if s is a Minimizer state, we have $\mathsf{U}(\mathsf{s}) = x$ and $\arg\min_{\mathsf{a}' \in \mathsf{Av}(\mathsf{s})} \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{U}(\mathsf{s}') \leq \mathsf{U}(\mathsf{s}, \mathsf{a}_s) = x$ and the assumption is satisfied. If s is a Maximizer state, we also use the fact that $x = \mathsf{bestExit}^{\square}_{\mathsf{U}}(T)$, and thus for all leaving actions a_ℓ we have $\mathsf{U}(\mathsf{s}, \mathsf{a}_\ell) \leq x$. Then, as all staying actions have an estimate of x and all leaving actions have at most x, we can conclude that $\arg\max_{\mathsf{a}' \in \mathsf{Av}(\mathsf{s})} \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{U}(\mathsf{s}') = x$.
Note that given the true values V, unfolding the Bellman equations for one step results in exactly the same values, as the value vector is a fixpoint. Thus, in that case we have an equality.

A.2. Strategies following only the approximants

Let σ^{L} and τ^{U} be the strategies that pick an action that is optimal according to the approximants L respectively U. Formally, for all states s, let $\sigma^{\mathsf{L}}(\mathsf{s})$ be an element of $\arg\max_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{L}(\mathsf{s}')$. Dually, let $\tau^{\mathsf{U}}(\mathsf{s})$ be an element of $\arg\min_{\mathsf{a} \in \mathsf{Av}(\mathsf{s})} \sum_{\mathsf{s}' \in S} \delta(\mathsf{s}, \mathsf{a}, \mathsf{s}') \cdot \mathsf{U}(\mathsf{s}')$.

We now prove that following these strategies for a finite number of steps does not decrease the probability to reach the target. For this, let $\Diamond^i X := \{\rho \in (S \times \mathsf{A})^\omega \mid \rho = \mathsf{s}_0\mathsf{a}_0\mathsf{s}_1\mathsf{a}_1 \cdots \wedge\ \mathsf{s}_i \in X\}$ denote the measurable set of all paths which reach a state from the set $X \subseteq S$ after exactly i steps. We mention two technical details: firstly, we only require that X is visited after exactly i steps. We do *not* require that this is the first visit to X. Secondly, this definition only restricts a finite prefix of the path; however, as we have defined the probability distribution only over infinite paths, this set contains infinite paths.

Lemma 7 *(Following the approximants for a finite time). For all states $\mathsf{s} \in S$ and all $i \in \mathbb{N}_0$, the following two statements hold. For all Minimizer strategies τ:*

$$\sum_{\mathsf{s}' \in S} \mathbb{P}_{\mathsf{s}}^{\sigma^{\mathsf{L}}, \tau}[\Diamond^i\{\mathsf{s}'\}] \cdot \mathsf{L}(\mathsf{s}') \geq \mathsf{L}(\mathsf{s}).$$

Dually, for all Maximizer strategies σ:

$$\sum_{\mathsf{s}' \in S} \mathbb{P}_{\mathsf{s}}^{\sigma, \tau^{\mathsf{U}}}[\Diamond^i\{\mathsf{s}'\}] \cdot \mathsf{U}(\mathsf{s}') \leq \mathsf{U}(\mathsf{s}).$$

Proof. We show only the first statement, as the second proof is completely analogous. We proceed by induction on i, the number of steps for which we follow σ^{L}. For $i = 0$,

$$\mathbb{P}_{\mathsf{s}}^{\sigma^{\mathsf{L}}, \tau}[\Diamond^0\{\mathsf{s}'\}] = \begin{cases} 1 & \text{if } \mathsf{s} = \mathsf{s}' \\ 0 & \text{otherwise} \end{cases}.$$

Thus, the left side of the inequality evaluates to $\mathsf{L}(\mathsf{s})$ and the base case is completed.

For the induction step, the following chain of equations proves our goal. We justify every transformation below the equations.

$$\begin{aligned}
&\sum_{\mathsf{s}' \in S} \mathbb{P}_{\mathsf{s}}^{\sigma^{\mathsf{L}},\tau}[\Diamond^{i+1}\{\mathsf{s}'\}] \cdot \mathsf{L}(\mathsf{s}') \\
&= \sum_{t \in S} \mathbb{P}_{\mathsf{s}}^{\sigma^{\mathsf{L}},\tau}[\Diamond^{i}\{t\}] \cdot \left(\sum_{\mathsf{s}' \in S} \delta(t, \pi(t), \mathsf{s}') \cdot \mathsf{L}(\mathsf{s}') \right) && \text{(Step 1)} \\
&\geq \sum_{t \in S} \mathbb{P}_{\mathsf{s}}^{\sigma^{\mathsf{L}},\tau}[\Diamond^{i}\{t\}] \cdot \mathsf{L}(t) && \text{(Step 2)} \\
&\geq \mathsf{L}(\mathsf{s}) && \text{(Induction hypothesis)}
\end{aligned}$$

For Step 1, instead of considering paths that follow the strategies for $i+1$ steps, we consider paths that follow it for i steps and explicitly unfold the final step. Here, $\pi(\mathsf{s}') = \begin{cases} \sigma^{\mathsf{L}}(\mathsf{s}') & \text{if } \mathsf{s} \in S_{\square} \\ \tau(\mathsf{s}') & \text{if } \mathsf{s} \in S_{\bigcirc} \end{cases}$.

For Step 2, we use the fourth assumption on the approximants, namely that unfolding the Bellman equation can only make them more precise. If t is a Maximizer state, by definition we have

$$\sigma^{\mathsf{L}}(t) = \operatorname*{arg\,max}_{\mathsf{a} \in \mathsf{Av}(t)} \sum_{\mathsf{s}' \in S} \delta(t, \mathsf{a}, \mathsf{s}') \cdot \mathsf{L}(\mathsf{s}')$$

and thus can immediately apply the assumption. If t is a Minimizer state, we know that the assumption holds for $\arg\min$ of all available actions, so it also holds for whichever action τ uses. In the final step, we apply the induction hypothesis. □

Lemma 8. *τ^{U} is an ε-optimal strategy of Minimizer.*

Proof. Using Lemma 7, we have that for all states s, all Maximizer strategies σ and all $i \in \mathbb{N}_0$:

$$\begin{aligned}
&\sum_{\mathsf{s}' \in S} \mathbb{P}_{\mathsf{s}}^{\sigma,\tau^{\mathsf{U}}}[\Diamond^{i}\{\mathsf{s}'\}] \cdot \mathsf{U}(\mathsf{s}') \\
&= \left(\sum_{\mathsf{s}' \in F} \mathbb{P}_{\mathsf{s}}^{\sigma,\tau^{\mathsf{U}}}[\Diamond^{i}\{\mathsf{s}'\}] \cdot \mathsf{U}(\mathsf{s}') \right) + \left(\sum_{\mathsf{s}' \in S \setminus F} \mathbb{P}_{\mathsf{s}}^{\sigma,\tau^{\mathsf{U}}}[\Diamond^{i}\{\mathsf{s}'\}] \cdot \mathsf{U}(\mathsf{s}') \right) && \text{(Splitting the sum)} \\
&\leq \mathsf{U}(\mathsf{s}) && \text{(Lemma 7)}
\end{aligned}$$

The second sum over all non-target states certainly evaluates to 0 or more. Also, $\mathsf{U}(\mathsf{s}') = 1$ for all $\mathsf{s}' \in F$. Hence, it holds that

$$\sum_{\mathsf{s}' \in F} \mathbb{P}_{\mathsf{s}}^{\sigma,\tau^{\mathsf{U}}}[\Diamond^{i}\{\mathsf{s}'\}] \leq \mathsf{U}(\mathsf{s}).$$

Thus, pulling the sum into the definition of the paths, we get

$$\mathbb{P}_{\mathsf{s}}^{\sigma,\tau^{\mathsf{U}}}[\Diamond^{i} F] \leq \mathsf{U}(\mathsf{s}).$$

Note that as this statement holds for all i and all target states are absorbing, we can also take the limit and consider all paths that reach a target state $\Diamond F$. Considering the initial state s_0 and using that U is correct and ε-optimal, we conclude that τ^{U} is ε-optimal:

$$\mathbb{P}_{\mathsf{s}_0}^{\sigma,\tau^{\mathsf{U}}}[\Diamond F] \leq \mathsf{U}(\mathsf{s}_0) \leq \mathsf{V}(\mathsf{s}_0) + \varepsilon. \quad \square$$

A.3. Ensuring maximizer makes progress

We already saw in the counter-example in the beginning that the dual of Lemma 8 is not true, as only playing actions according to the approximants is not ε-optimal. We will now show how to augment σ^{L} to become ε-optimal.

We employ a folklore construction which is sometimes called *attractor*, similar[13] to the one used in [15, Section 5.3]. In other words: we perform a backwards search from the target state, using only actions that are ε-optimal according to L.

[13] In that paper, they ensure that the target or sink states are reached almost surely. We require that target states are reached with positive probability.

Formally, we use the following recursive procedure. Let $S_0^{\text{attr}} = F$. For $i \geq 1$, we define

$$S_i^{\text{attr}} = \{s \in S_\square \mid \exists a \in \arg\max_{a' \in Av(s)} L(s,a').\text{Post}(s,a) \cap S_{i-1}^{\text{attr}} \neq \emptyset\}$$
$$\cup \{s \in S_\bigcirc \mid \forall a \in Av(s).\text{Post}(s,a) \cap S_{i-1}^{\text{attr}} \neq \emptyset\}.$$

We terminate when $S_j^{\text{attr}} = S_{j-1}^{\text{attr}}$.

For all i, S_i^{attr} contains those states that have a path of length at most i to the target which for Maximizer states uses only ε-optimal actions, and for Minimizer states has no other choice but to continue on a path towards the target. This can be proven by simple induction on the length of the path. Note that the states with no path to the target are not in S_i^{attr} for any i.

Let $\mathcal{P} := \{s \in S \mid V(s) > 0\}$ be all states with a positive value. As every state with a positive value has a path to a target state, it holds that $\mathcal{P} = S_j^{\text{attr}}$, where j is the iteration where the recursive procedure terminates.

We now define the ε-optimal Maximizer strategy $\sigma^{L,\text{attr}}$. Note that for states in F or states with value 0, the strategy is irrelevant as either we anyway have reached target or no strategy can reach the target (against a rational opponent). Thus, here $\sigma^{L,\text{attr}}$ picks an arbitrary available action. For every state in $s \in \mathcal{P} \setminus F$, we pick an action as follows: let i be the minimum number with $s \in S_i^{\text{attr}}$. As $s \in \mathcal{P}$, such an i exists. Then the strategy $\sigma^{L,\text{attr}}(s)$ picks an action from

$$\{a \in Av(s) \mid a \in \arg\max_{a' \in Av(s)} L(s,a') \wedge \text{Post}(s,a) \cap S_{i-1}^{\text{attr}} \neq \emptyset\}.$$

Such an action exists, because s was added in the i-th step, hence it must have an action leading to S_{i-1}^{attr} by definition of S_i^{attr}. Note that not all successors of the action have to be in S_{i-1}^{attr}, but we require only at least one in order to have a positive probability of making progress. Further, note that $i - 1 \geq 0$, since $s \notin F$, so the strategy is well defined.

Under this strategy $\sigma^{L,\text{attr}}$, for every Minimizer strategy τ, every state in $s \in \mathcal{P} \setminus F$ has a positive probability to reach the target states. To prove this, observe that every s is in some S_i^{attr}. For a Maximizer state, by construction it uses an action with a positive chance to reach a state in S_{i-1}^{attr}; for Minimizer, it has no choice but to put positive probability on reaching S_{i-1}^{attr}. Repeating the argument until $i = 0$ shows the claim.

Lemma 9. *$\sigma^{L,attr}$ is an ε-optimal strategy of Maximizer.*

Proof. We again use Lemma 7 and split the sum as in the proof of Lemma 8. However, this time we split S into F, $\mathcal{P}$ and the remaining states with value 0. Note that for states with value 0 also the lower approximant is 0, since by assumption the lower approximant is correct. Hence, the summand considering the value 0 states evaluates to 0. Thus we have that for all states s, all Minimizer strategies τ and all $i \in \mathbb{N}_0$:

$$\left(\sum_{s' \in F} \mathbb{P}_s^{\sigma^{L,\text{attr}},\tau}[\Diamond^i\{s'\}] \cdot L(s')\right) + \left(\sum_{s' \in \mathcal{P} \setminus F} \mathbb{P}_s^{\sigma^{L,\text{attr}},\tau}[\Diamond^i\{s'\}] \cdot L(s')\right) \geq L(s)$$

Here is the crucial difference between constructing a Maximizer and Minimizer strategy: for Maximizer, it is possible that the whole probability mass still remains in $\mathcal{P}$. Following a strategy that maximizes L for a finite number of steps does not decrease the chance of reaching the targets, but it might be necessary to eventually switch to the optimal strategy in order to actually realize the value. This is why we used the attractor construction to ensure that under $\sigma^{L,\text{attr}}$ every state in $\mathcal{P} \setminus F$ has a positive probability to reach the target. Then, for $i \to \infty$, the second summand converges to 0. Intuitively, more and more probability leaves towards the target or sink states. Formally, as from every state there exists a path with positive probability to reach a target, the probability to reach F in $|S|$ steps is at least $q := p^{|S|}$, where p is the minimum probability occurring in the SG. Since $\sum_i^\infty q \cdot (1-q)^i = 1$ (geometric series), the probability to remain in $\mathcal{P} \setminus F$ is 0.

Having ensured that the probability mass actually arrives at the target, we can consider the infinite behaviour and conclude:

$$\mathbb{P}_s^{\sigma^{L,\text{attr}},\tau}[\Diamond F] \geq L(s)$$

Note that we again pulled the sum into the definition of the paths and used that for target states the lower approximant is initialized to 1. Finally, considering the initial state and using that L is correct and ε-close, we arrive at our goal:

$$\mathbb{P}_{s_0}^{\sigma^{L,\text{attr}},\tau}[\Diamond F] \geq L(s_0) \geq V(s_0) - \varepsilon. \quad \square$$

Note that, given the true values V, all inequalities in the proofs become equalities. Thus, we also showed how to construct optimal strategies from the true value vector.

Appendix B. Definition of COLLAPSE

The way in which we define COLLAPSE is not only able to collapse ECs in MDPs, but also simple ECs (SEC, see Definition 5) in SGs. Note that every EC in an MDP is a SEC. If there are no actions of Maximizer leaving the SEC, we have to keep staying actions, so the SEC becomes an absorbing state (we do not want a state without actions, as we assumed the game is non-blocking). Otherwise all staying actions are removed and the SEC becomes a single state, whose available actions are all the exiting actions of Maximizer states in the SEC. Note that we have to remove Minimizer actions, because otherwise we would allow Maximizer to use an exit that Minimizer wants to prevent.

Definition 6 *(COLLAPSE).* Let $\mathsf{G} = (S, S_\Box, S_\bigcirc, \mathsf{s}_0, \mathsf{A}, \mathsf{Av}, \delta)$ be an SG and T a SEC in G. Then $\mathsf{COLLAPSE}(\mathsf{G}, T) = \mathsf{G}' = (S', S'_\Box, S'_\bigcirc, \mathsf{s}'_0, \mathsf{A}', \mathsf{Av}', \delta')$, where G' is defined as follows:

- $S' = (S \setminus T) \cup \{\mathsf{s}_T\}$
- $S'_\Box = (S_\Box \setminus T) \cup \{\mathsf{s}_T\}$
- $S'_\bigcirc = S_\bigcirc \setminus T$
- $\mathsf{s}'_0 = \begin{cases} \mathsf{s}_T & \text{if } \mathsf{s}_0 \in T \\ \mathsf{s}_0 & \text{otherwise} \end{cases}$
- $\mathsf{A}' = \mathsf{A} \cup \{\bot\}$, where $\bot \notin \mathsf{A}$ is a new action.
- $\mathsf{Av}'(\mathsf{s})$ is defined for all $\mathsf{s} \in S'$ by:
  - $\mathsf{Av}(\mathsf{s})$,
    if $\mathsf{s} \in (S \setminus T)$, i.e. $\mathsf{s} \neq \mathsf{s}_T$
    (Rest stays the same)
  - $\bigcup_{t \in T_\Box} \{\mathsf{a} \in \mathsf{Av}(t) \mid (t, \mathsf{a}) \text{ exits } T\}$,
    if $\mathsf{s} = \mathsf{s}_T$ and $\exists t \in T_\Box : (t, \mathsf{a}) \text{ exits } T$
    (Keep leaving Maximizer actions, if there is an exit for Maximizer)
  - $\{\bot\}$,
    if $\mathsf{s} = \mathsf{s}_T$ and $\neg\exists t \in T_\Box : (t, \mathsf{a}) \text{ exits } T$
    (Keep a staying action, if there is no exit for Maximizer)
- δ' is defined for all $\mathsf{s} \in S'$ and $\mathsf{a} \in \mathsf{Av}'(\mathsf{s})$ by:
  - $\delta'(\mathsf{s}, \mathsf{a})(\mathsf{s}') = \delta(\mathsf{s}, \mathsf{a})(\mathsf{s}')$
    for all $\mathsf{s}' \in S'$ with $\mathsf{s}, \mathsf{s}' \neq \mathsf{s}_T$
    (Rest stays the same)
  - $\delta'(\mathsf{s}, \mathsf{a})(\mathsf{s}_T) = \sum_{\mathsf{s}' \in T} \delta(\mathsf{s}, \mathsf{a})(\mathsf{s}')$,
    if $\mathsf{s} \neq \mathsf{s}_T$
    (going to T)
  - $\delta'(\mathsf{s}_T, \mathsf{a})(\mathsf{s}') = \delta(\mathsf{s}, \mathsf{a})(\mathsf{s}')$
    for all $\mathsf{a} \in \mathsf{Av}'(\mathsf{s}_T)$, the unique $\mathsf{s} \in T$ with $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$ and all $\mathsf{s}' \in S' \setminus \{\mathsf{s}_T\}$
    (leaving from T)
  - $\delta'(\mathsf{s}_T, \mathsf{a})(\mathsf{s}_T) = \sum_{t \in T} \delta(\mathsf{s}, \mathsf{a})(t)$
    for all $\mathsf{a} \in \mathsf{Av}'(\mathsf{s}_T)$ and the unique $\mathsf{s} \in T$ with $\mathsf{a} \in \mathsf{Av}(\mathsf{s})$
    (staying in T when using an exit)
  - $\delta'(\mathsf{s}_T, \bot)(\mathsf{s}_T) = 1$
    if $\bot \in \mathsf{Av}'(\mathsf{s}_T)$
    (staying in T in the case of no Maximizer exits)

Note that when defining δ in the case of leaving from T, we assume that no action is available for multiple states in T. If there are duplicates, we rename the actions before collapsing, e.g. by indexing them with their respective state.

When computing the reachability probability of reaching a target set F, we also have to adjust the target set as follows: $F' = (F \setminus T) \cup \begin{cases} \emptyset & \text{if } T \cap F = \emptyset \\ \{\mathsf{s}_T\} & \text{otherwise} \end{cases}$.

We now argue about the correctness of this definition. Firstly, note that G' is well-defined. All states in $S \setminus T$ remain a part of the state space, and the controlling player as well as the available actions remain unchanged. All states from T have been replaced by the newly added s_T. To argue about the transition function, we make a case distinction:

- Going to T: if an action previously lead to a state $\mathsf{s}' \in T$, now this probability mass leads to s_T. Note that an action might have multiple different successors in T, thus we sum over all $\mathsf{s}' \in T$ when defining $\delta'(\mathsf{s}, \mathsf{a})(\mathsf{s}_T)$. Thus, δ' still is a valid probability distribution, since all probability mass previously going to a state in T now goes to s_T.

- Using an exit from T: the third and fourth items in the definition of δ' deal with the case that we use an action $a \in Av'(s_T)$ that was an exit from T. Note that for this case we use the additional assumption that — possibly by renaming actions — there were no states $s, s' \in T$ such that $a \in Av(s) \cap Av(s')$, i.e. actions are unique for the states in T. So for an action a there is a unique state-action pair (s, a) with $s \in T$ that identifies this exit. Thus, for transitions leading to states s' outside of T, we can reuse the old transition function $\delta(s, a)(s')$; this is done by the third item. For transitions leading back into T (which is possible, since for a to be an exit we do not require all successors to be outside of T, but only at least one of them), we use the same idea as in the previous item: we sum the probability of all transitions leading to any state $t \in T$ and assign it to s_T; this is done by the fourth item. Thus, δ' still is a valid probability distribution, since (i) it is defined for every available action of s_T, and (ii) all probability mass leaving T remains unchanged, and all probability mass staying in T now leads to s_T.
- If there is no exit for Maximizer in T, formally $\neg\exists t \in T_\square : (t, a)$ exits T, then we use the newly added action $\bot$ that loops surely.

Appendix C. Description of the experimental models

Our experiments are based on the ones that were conducted in [58]. Most models we use are also analyzed in that thesis, and we obtained them from the website,[14] where the author of the thesis made them available for download. We used the exact models from that website, but partly modified the properties to be of a form that our implementation can handle. These modifications did not change the semantics of the property, e.g. instead of formulating a property that a probability is greater than a certain number (P>0.999) we compute the maximal probability (Pmax=?), and then manually check whether it is greater than the number. The only models not from [58] are the MDPs csma and leader and the MC hm.

We consider six MDP models, namely firewire, wlan, zeroconf, csma, leader and mer. The first five are part of the PRISM benchmark suite [45], mer is from [31]. The four SG models are mdsm, cdmsn, team-form and cloud, and the first three are contained in the PRISM-games case studies.[15] Cloud is from [13]. Note that some of the SG models actually contain more than two players. However, since there are at most two coalitions, they can be viewed as an SG with only two players. We will now shortly describe all models, the properties we check and the parameters we use for scaling.

firewire [48]:
This case study models the protocol known as FireWire, which is a leader election protocol of the IEEE 1394 High Performance Serial Bus. Several devices connected to a bus can use the protocol to dynamically elect a leader. We compute the probability Pmax=? [Fleader_elected], so the maximal probability with which a leader gets elected before the deadline. By this one can check the property that a leader gets elected with a certain, optimally high, probability. To scale the model up, we raise the deadline.

wlan [47]:
This model describes the two-way handshake mechanism of the IEEE 802.11 medium access control (WLAN protocol). Two stations try to communicate with each other; however, if both of them send at once, a collision occurs. We are interested in computing the maximum probability that both stations transmit their messages correctly, i.e. Pmax=? [F s1=12 & s2=12], where s1 and s2 describe the state of the stations, and 12 is the final state where the transmission was successful. To scale the model up, we increase the maximal backoff k and the maximal number of collisions COL.

zeroconf [46]:
Zeroconf is a protocol for dynamically assigning an IP address to a device, provided that several other hosts have already blocked some IP addresses. The device picks some IP randomly and then sends probes to check whether this address is already in use. The parameters that we use to scale the model are N, the number of other hosts already possessing an IP address and K, the number of probes sent. The probability we are interested in is Pmin=? [F configured], so the minimum probability with which the device obtains an IP address.

csma: [45]
This case study concerns the IEEE 802.3 CSMA/CD (Carrier Sense, Multiple Access with Collision Detection) protocol. N is the number of stations and K is the maximum backoff. Pmin=? [F min_backoff_after_success<=K] is the probability we are interested in, namely that a message of some station is eventually delivered before K backoffs.

leader [45]:
This case study is based on the asynchronous leader election protocol of [37]. This protocol solves the following problem. Given an asynchronous ring of N processors design a protocol such that they will be able to elect a leader (a uniquely designated processor) by sending messages around the ring. The probability we are interested in is Pmax=? [F “elected”], so that at some point a leader is elected.

mer [31]:
In the Mars Exploration Rover there is a resource arbiter that handles distributing resources to different users. There is a probability x that the communication between the arbiter and the users fails. We change this probability to influence the

[14] http://www.prismmodelchecker.org/files/thesismujma/.
[15] prismmodelchecker.org/games/casestudies.php.

structure of the MDP. The probability we compute is Pmax=? [F err_G], which is the maximum probability that an error occurs.

hm [33]:

An example that was handcrafted to show when value iteration without guarantees fails. It consists of two chains of states of length n that lead to the target and a sink state; however, at every position in the chain there is some probability to go back to the initial state. This also is an adversarial example for simulation based algorithms, as the probability to reach a sink state and make any progress the simulation has to be lucky n times in a row, which is unlikely.

mdsm [20]:

This case study models multiple households which all consume different amounts of energy over time. To minimize the peak energy consumption, they utilize the distributed energy management "Microgrid Demand-Side Management" (mdsm). The property we check is the maximal probability with which the first household can deviate from the management algorithm, i.e. Pmax=? [F deviated], which should be smaller than 0.01. We check the property once for player 1 and once for player 2.

cdmsn [20,54]:

This model describes a set of agents which have different sites available and different preferences over these sites. The collective decision making algorithm of this case study is utilized so that the agents agree on one decision. We analyse the model to find the strategy for player 1 to make the agents agree on the first site with a high probability, so <<p1>> Pmax=? [F all_prefer_1].

team-form [21]:

As in the previous case study, there is a set of agents in a distributed environment. They need to form teams so they are able to perform a set of tasks together. We want to compute a strategy so that the first task is completed with the maximal possible probability, so we check the property <<p1,p2,p3>> Pmax=? [F task1_completed]. The model can be scaled using the number of agents N. However, for this model PRISM's pre-computations already solve the problem and hence it cannot be used to compare the approaches; thus, it is not included in any of our tables.

cloud [13]:

This model describes several servers and virtual machines forming a cloud system. The controller of the system wants to deploy a web application on one of the virtual machines, but it is possible that the servers fail due to hardware failures. We compute the strategy and the maximal probability for the controller to successfully deploy his software, so <<controller>> Pmax=? [F deployed]. The model can be scaled by increasing the number of virtual machines N.

References

[1] D. Andersson, P.B. Miltersen, The complexity of solving stochastic games on graphs, in: ISAAC, Springer, 2009, pp. 112–121.
[2] G. Arslan, S. Yüksel, Decentralized Q-learning for stochastic teams and games, IEEE Trans. Autom. Control 62 (2017) 1545–1558, https://doi.org/10.1109/TAC.2016.2598476.
[3] P. Ashok, K. Chatterjee, P. Daca, J. Kretínský, T. Meggendorfer, Value iteration for long-run average reward in Markov decision processes, in: CAV (1), Springer, 2017, pp. 201–221.
[4] P. Ashok, K. Chatterjee, J. Kretínský, M. Weininger, T. Winkler, Approximating values of generalized-reachability stochastic games, in: LICS, ACM, 2020, pp. 102–115.
[5] P. Ashok, P. Daca, J. Kretínský, M. Weininger, Statistical model checking: black or white?, in: ISoLA (1), Springer, 2020, pp. 331–349.
[6] P. Ashok, J. Kretínský, M. Weininger, PAC statistical model checking for Markov decision processes and stochastic games, in: CAV (1), Springer, 2019, pp. 497–519.
[7] C. Baier, J. Katoen, Principles of Model Checking, MIT Press, 2008.
[8] C. Baier, J. Klein, L. Leuschner, D. Parker, S. Wunderlich, Ensuring the reliability of your model checker: interval iteration for Markov decision processes, in: CAV (1), Springer, 2017, pp. 160–180.
[9] N. Balaji, S. Kiefer, P. Novotný, G.A. Pérez, M. Shirmohammadi, On the complexity of value iteration, in: ICALP, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 102:1–102:15.
[10] R.I. Brafman, M. Tennenholtz, A near-optimal polynomial time algorithm for learning in certain classes of stochastic games, Artif. Intell. 121 (2000) 31–47, https://doi.org/10.1016/S0004-3702(00)00039-4.
[11] T. Brázdil, K. Chatterjee, M. Chmelik, V. Forejt, J. Kretínský, M.Z. Kwiatkowska, D. Parker, M. Ujma, Verification of Markov decision processes using learning algorithms, in: ATVA, Springer, 2014, pp. 98–114.
[12] L. Busoniu, R. Babuska, B.D. Schutter, A comprehensive survey of multiagent reinforcement learning, IEEE Trans. Syst. Man Cybern. Part C 38 (2008) 156–172, https://doi.org/10.1109/TSMCC.2007.913919.
[13] R. Calinescu, S. Kikuchi, K. Johnson, Compositional reverification of probabilistic safety properties for large-scale complex IT systems, in: Monterey Workshop, Springer, 2012, pp. 303–329.
[14] J. Cámara, G.A. Moreno, D. Garlan, Stochastic game analysis and latency awareness for proactive self-adaptation, in: SEAMS, ACM, 2014, pp. 155–164.
[15] K. Chatterjee, L. de Alfaro, T.A. Henzinger, Strategy improvement for concurrent reachability and turn-based stochastic safety games, J. Comput. Syst. Sci. 79 (2013) 640–657, https://doi.org/10.1016/j.jcss.2012.12.001.
[16] K. Chatterjee, N. Fijalkow, A reduction from parity games to simple stochastic games, in: GandALF, 2011, pp. 74–86.
[17] K. Chatterjee, M. Henzinger, Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification, in: SODA, SIAM, 2011, pp. 1318–1336.
[18] K. Chatterjee, T.A. Henzinger, Value iteration, in: O. Grumberg, H. Veith (Eds.), 25 Years of Model Checking - History, Achievements, Perspectives, Springer, 2008, pp. 107–138.
[19] K. Chatterjee, T.A. Henzinger, B. Jobstmann, A. Radhakrishna, Gist: a solver for probabilistic games, in: CAV, 2010, pp. 665–669.
[20] T. Chen, V. Forejt, M.Z. Kwiatkowska, D. Parker, A. Simaitis, Automatic verification of competitive stochastic systems, Form. Methods Syst. Des. 43 (2013) 61–92, https://doi.org/10.1007/s10703-013-0183-7.
[21] T. Chen, M.Z. Kwiatkowska, D. Parker, A. Simaitis, Verifying team formation protocols with probabilistic model checking, in: CLIMA, Springer, 2011, pp. 190–207.

[22] T. Chen, M.Z. Kwiatkowska, A. Simaitis, C. Wiltsche, Synthesis for multi-objective stochastic games: an application to autonomous urban driving, in: QEST, 2013, pp. 322–337.
[23] C. Cheng, A.C. Knoll, M. Luttenberger, C. Buckl, GAVS+: an open platform for the research of algorithmic game solving, in: TACAS, Springer, 2011, pp. 258–261.
[24] A. Condon, The complexity of stochastic games, Inf. Comput. 96 (1992) 203–224, https://doi.org/10.1016/0890-5401(92)90048-K.
[25] A. Condon, On algorithms for simple stochastic games, in: Advances in Computational Complexity Theory, DIMACS/AMS, 1993, pp. 51–71.
[26] P. Daca, T.A. Henzinger, J. Kretínský, T. Petrov, Faster statistical model checking for unbounded temporal properties, ACM Trans. Comput. Log. 18 (2017) 12:1–12:25, https://doi.org/10.1145/3060139.
[27] B.A. Davey, H.A. Priestley, Introduction to Lattices and Order, Cambridge University Press, 2002.
[28] C. Dehnert, S. Junges, J. Katoen, M. Volk, A storm is coming: a modern probabilistic model checker, in: CAV (2), Springer, 2017, pp. 592–600.
[29] T. van Dijk, Attracting tangles to solve parity games, in: CAV (2), Springer, 2018, pp. 198–215.
[30] J. Eisentraut, J. Kretínský, A. Rotar, Stopping criteria for value and strategy iteration on concurrent stochastic reachability games, CoRR, arXiv:1909.08348 [abs], http://arxiv.org/abs/1909.08348, 2019.
[31] L. Feng, M.Z. Kwiatkowska, D. Parker, Automated learning of probabilistic assumptions for compositional reasoning, in: FASE, Springer, 2011, pp. 2–17.
[32] J. Filar, K. Vrieze, Competitive Markov Decision Processes, Springer Science & Business Media, 2012.
[33] S. Haddad, B. Monmege, Interval iteration algorithm for mdps and imdps, Theor. Comput. Sci. 735 (2018) 111–131, https://doi.org/10.1016/j.tcs.2016.12.003.
[34] E.M. Hahn, A. Hartmanns, C. Hensel, M. Klauck, J. Klein, J. Kretínský, D. Parker, T. Quatmann, E. Ruijters, M. Steinmetz, The 2019 comparison of tools for the analysis of quantitative formal models - (QCOMP 2019 competition report), in: TACAS (3), Springer, 2019, pp. 69–92.
[35] A.J. Hoffman, R.M. Karp, On nonterminating stochastic games, Manag. Sci. 12 (1966) 359–370, https://doi.org/10.1287/mnsc.12.5.359.
[36] A. Hordijk, L. Kallenberg, Linear programming and Markov decision chains, Manag. Sci. 25 (1979) 352–362, https://doi.org/10.1287/mnsc.25.4.352.
[37] A. Itai, M. Rodeh, Symmetry breaking in distributed networks, Inf. Comput. 88 (1990) 60–87, https://doi.org/10.1016/0890-5401(90)90004-2.
[38] M. Kattenbelt, M.Z. Kwiatkowska, G. Norman, D. Parker, A game-based abstraction-refinement framework for Markov decision processes, Form. Methods Syst. Des. 36 (2010) 246–280, https://doi.org/10.1007/BF01415989.
[39] E. Kelmendi, J. Krämer, J. Kretínský, M. Weininger, Value iteration for simple stochastic games: stopping criterion and learning algorithm, in: CAV (1), Springer, 2018, pp. 623–642.
[40] J. Kretínský, T. Meggendorfer, Efficient strategy iteration for mean payoff in Markov decision processes, in: ATVA, Springer, 2017, pp. 380–399.
[41] J. Kretínský, T. Meggendorfer, Of cores: a partial-exploration framework for Markov decision processes, Log. Methods Comput. Sci. 16 (2020), https://lmcs.episciences.org/6833.
[42] J. Kretínský, E. Ramneantu, A. Slivinskiy, M. Weininger, Comparison of algorithms for simple stochastic games, in: GandALF, 2020, pp. 131–148.
[43] M. Kwiatkowska, G. Norman, D. Parker, G. Santos, Prism-games 3.0: stochastic game verification with concurrency, equilibria and time, in: CAV (2), Springer, 2020, pp. 475–487.
[44] M.Z. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: verification of probabilistic real-time systems, in: CAV, Springer, 2011, pp. 585–591.
[45] M.Z. Kwiatkowska, G. Norman, D. Parker, The PRISM benchmark suite, in: QEST, IEEE Computer Society, 2012, pp. 203–204.
[46] M.Z. Kwiatkowska, G. Norman, D. Parker, J. Sproston, Performance analysis of probabilistic timed automata using digital clocks, Form. Methods Syst. Des. 29 (2006) 33–78, https://doi.org/10.1007/s10703-006-0005-2.
[47] M.Z. Kwiatkowska, G. Norman, J. Sproston, Probabilistic model checking of the IEEE 802.11 wireless local area network protocol, in: PAPM-PROBMIV, Springer, 2002, pp. 169–187.
[48] M.Z. Kwiatkowska, G. Norman, J. Sproston, Probabilistic model checking of deadline properties in the IEEE 1394 firewire root contention protocol, Form. Asp. Comput. 14 (2003) 295–318, https://doi.org/10.1007/s001650300007.
[49] S.M. LaValle, Robot motion planning: a game-theoretic foundation, Algorithmica 26 (2000) 430–465, https://doi.org/10.1007/s004539910020.
[50] J. Li, W. Liu, A novel heuristic Q-learning algorithm for solving stochastic games, in: IJCNN, 2008, pp. 1135–1144.
[51] H.B. McMahan, M. Likhachev, G.J. Gordon, Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees, in: ICML, ACM, 2005, pp. 569–576.
[52] K. Phalakarn, T. Takisaka, T. Haas, I. Hasuo, Widest paths and global propagation in bounded value iteration for stochastic games, in: CAV (2), Springer, 2020, pp. 349–371.
[53] M.L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, Wiley Series in Probability and Statistics, Wiley, 1994.
[54] F. Saffre, A. Simaitis, Host selection through collective decision, ACM Trans. Auton. Adapt. Syst. 7 (2012) 4:1–4:16, https://doi.org/10.1145/2168260.2168264.
[55] A.L. Strehl, L. Li, E. Wiewiora, J. Langford, M.L. Littman, PAC model-free reinforcement learning, in: ICML, ACM, 2006, pp. 881–888.
[56] M. Svorenová, M. Kwiatkowska, Quantitative verification and strategy synthesis for stochastic games, Eur. J. Control 30 (2016) 15–30, https://doi.org/10.1016/j.ejcon.2016.04.009.
[57] A. Tcheukam, H. Tembine, One swarm per queen: a particle swarm learning for stochastic games, in: SASO, 2016, pp. 144–145.
[58] M. Ujma, On verification and controller synthesis for probabilistic systems at runtime, Ph.D. thesis, University of Oxford, UK, 2015, http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.711811.
[59] L.G. Valiant, A theory of the learnable, Commun. ACM 27 (1984) 1134–1142, https://doi.org/10.1145/1968.1972.
[60] O. Vrieze, S. Tijs, T.E. Raghavan, J. Filar, A finite algorithm for the switching control stochastic game, OR Spektrum 5 (1983) 15–24, https://doi.org/10.1007/BF01720283.
[61] M. Wen, U. Topcu, Probably approximately correct learning in stochastic games with temporal logic specifications, in: IJCAI, IJCAI/AAAI Press, 2016, pp. 3630–3636, http://www.ijcai.org/Abstract/16/511.