Approximating Values of Generalized-Reachability Stochastic Games

Pranav Ashok
Technical University of Munich
Germany
ashok@in.tum.de

Krishnendu Chatterjee
IST Austria
Austria
Krishnendu.Chatterjee@ist.ac.at

Jan Křetínský
Technical University of Munich
Germany
jan.kretinsky@in.tum.de

Maximilian Weininger
Technical University of Munich
Germany
maxi.weininger@tum.de

Tobias Winkler
RWTH Aachen University
Germany
tobias.winkler@cs.rwth-aachen.de

Abstract

Simple stochastic games are turn-based 2½-player games with a reachability objective. The basic question asks whether one player can ensure reaching a given target with at least a given probability. A natural extension is games with a conjunction of such conditions as objective. Despite a plethora of recent results on the analysis of systems with multiple objectives, the decidability of this basic problem remains open. In this paper, we present an algorithm approximating the Pareto frontier of the achievable values to a given precision. Moreover, it is an anytime algorithm, meaning it can be stopped at any time returning the current approximation and its error bound.

CCS Concepts: • **Theory of computation** → *Algorithmic game theory*; **Verification by model checking**; • **Mathematics of computing** → *Probability and statistics.*

Keywords: Stochastic games; Multiple Reachability Objectives; Pareto frontier; Anytime algorithm

ACM Reference Format:
Pranav Ashok, Krishnendu Chatterjee, Jan Křetínský, Maximilian Weininger, and Tobias Winkler. 2020. Approximating Values of Generalized-Reachability Stochastic Games. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '20), July 8–11, 2020, Saarbrücken, Germany.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3373718.3394761

1 Introduction

Simple stochastic games [27] are zero-sum turn-based stochastic games (SG) with two players, which we call Maximizer and Minimizer. The objective of player Maximizer is to maximize the probability of reaching a given target set of states, while player Minimizer aims at the opposite. The basic decision problem is to determine whether there is a strategy for Maximizer achieving at least a given probability threshold. These games are interesting theoretically: the problem is known to be in NP ∩ co-NP, but whether it belongs to P is a major and long-standing problem. Moreover, several other important game problems such as parity games reduce to it [17]. Besides, they are also practically relevant: they can serve as a tool for synthesis with safety/co-safety objectives in environments with stochastic uncertainty.

Multi-objective stochastic systems have attracted a lot of attention recently, both SG and the special case with only one player (Markov decision processes, MDP [43]). They model and enable optimization with respect to conflicting goals, where a desired trade-off is to be considered. A natural multi-dimensional generalization of the reachability threshold constraint $\mathbb{P}[\Diamond T] \geq t$ is a conjunction $\bigwedge_i \mathbb{P}[\Diamond T_i] \geq t_i$ giving rise to *generalized-reachability* (or *multiple-reachability*) *stochastic games*, similar to e.g. generalized mean-payoff SG [8, 16]. The problem is then to decide whether a given vector of thresholds can be achieved by Maximizer. Note that these games are not determined [23], and in this paper we consider the lower-value (worst-case) problem formulation, i.e. finding a strategy of Maximizer that can guarantee the vector no matter what Minimizer does.

The main results established in the literature are as follows. For MDP, while generalized mean-payoff can be solved in P [10, 15], generalized reachability is PSPACE-hard and can be solved in exponential time [44]. For SG, generalized mean-payoff has been solved for almost-sure conditions only [8, 16] and approximation of the values for generalized mean-payoff as well as generalized reachability are still open. The generalized-reachability SG problem is only known to be decidable for the subclass of *stopping* SGs with a *2-dimensional* objective [13] (an SG is stopping if under any strategies a designated set of sinks is reached almost surely).

LICS '20, July 8–11, 2020, Saarbrücken, Germany

ACM ISBN 978-1-4503-7104-9/20/07.
https://doi.org/10.1145/3373718.3394761

The main open question for generalized-reachability SG is decidability. There are several important subgoals towards this problem: From the decidability perspective, stopping SG with more than 2-dimensional objectives, or general SG with 2- (or more) dimensional objectives have been open. Moreover, the same holds even for ε-approximability. From the algorithmic perspective, [23] provides a converging sequence of *lower bounds* on the *Pareto frontier*, i.e. the set of achievable vectors that are pointwise-maximal (in other words, vectors which cannot be improved in one dimension without sacrificing another one). It is open whether converging upper bounds can also be computed. Since such bounds would imply ε-approximability, this open question is the most imminent.

Our contribution in this paper is twofold. Firstly, we prove the following theorem:

> **Theorem:** *The set of all achievable vectors in an* arbitrary *(not necessarily stopping) SG with generalized-reachability objective of* any dimension *can be effectively approximated for any given precision ε.*

Secondly, we provide a value-iteration algorithm that approximates the Pareto frontier by giving converging lower *and upper* bounds. Consequently, it becomes an *anytime algorithm*, providing the current approximation and its error at each moment of the computation. Thus our first contribution resolves the ε-approximability open question and our second contribution resolves the algorithmic open question; both results are for arbitrary SG with generalized-reachability objectives of any dimension.

Convergent upper bounds on the value are known to be notoriously difficult to achieve. Until recently, the default engine for analysis in the most used probabilistic model checker PRISM [40] and PRISM-Games [22] used *value iteration*, e.g. [43], which converges to the value from below, but because of the used stopping criteria the results could be arbitrarily wrong [35]. For a solution with a given precision, one could use linear programming instead, which however, does not scale well for MDP, and, more importantly, does not work at all for SG [28]. For MDP, value iteration has been extended [11, 35] so that it provides not only the under-approximating convergent sequence, but also an over-approximating one, calling the technique "bounded value iteration" (due to [41]) or "interval iteration", respectively. Its essence is to collapse *maximal end components* (MECs) of the MDP, thereby not changing the values; on MDP without MECs the over-approximating sequence converges to the actual value of the (collapsed as well as original) MDP. This technique was further extended to MDP with mean-payoff objective [2]. In contrast, for SG one cannot collapse MECs since they account for non-trivial alternating structure, as opposed to MDP, where any desired action exiting the MEC can be taken almost surely. Therefore, a more complex procedure has been proposed for SG [37]: Depending on the current under-approximation, problematic parts of MECs are dynamically identified and their over-approximation is lowered to over-approximations of certain actions exiting the MEC, as exemplified and explained later. We lift this procedure to general dimensions. Note that we do not give any convergence rate for our algorithm, because it is not possible to extend the argument of the single-dimensional case [19] in a straightforward manner. This argument requires the lowest probability occurring in a play to be bounded. However, in the multi-dimensional setting strategies may need infinite memory and hence there is no lower bound on the probability that a strategy assigns to actions [24, Appendix B1, full version]. Giving bounds on the convergence is an interesting direction of future work.

This paper combines and extends several techniques from literature to obtain the corresponding result for the multi-dimensional case:

- Firstly, we use the Bellman operator extended to downward-closed sets (instead of just real values) [23], allowing for value iteration in the multi-dimensional setting.
- Secondly, we exploit the technique of [37], which in the single-dimensional setting repetitively identifies the problematic parts of MECs hindering convergence.
- Thirdly, in order to apply this technique, we reduce the multi-dimensional problem to a continuum of single-dimensional problems, by splitting the Pareto front into directions, similar to [33].
- Fourthly, we group the single-dimensional problems into finitely many *regions*, similar in spirit to regions of timed automata [1] since they are essentially given by orderings of the approximate values of certain actions. Nevertheless, due to the projective geometry of the problem, we need to work slightly more generally with simplicial complexes, see e.g. [36].

The main technical difficulty is to identify (i) the parts of MECs with an unjustified too high upper bound and (ii) the value to which it should be decreased in each step. Both of these depend on the desired trade-off between the targets. As we compute the whole set of achievable vectors, we need to consider all possible trade-offs, which are, moreover, uncountably many.

Related work. Already for a decade, MDP have been extensively studied in the setting of multiple objectives. Multiple objectives have been considered both qualitative, such as reachability and LTL [30], as well as quantitative, such as mean payoff [10, 15], discounted sum [18], or total reward [32]. The expectation has been combined with variance in [12]. Beside expectation queries, conjunctions of percentile (threshold) queries have been considered for various objectives [10, 21, 31, 45]. Further, for general Boolean combinations for Markov chains with total reward, [34] approximates the value, while computability is still open. In contrast, [47] shows that Boolean combinations over mean payoff games become quickly undecidable. For the specifics of the two-dimensional case and the interplay of the two objectives, see [4]. The usage

of the multi-dimensional setting is discussed in [5, 6], comparing multiple rewards and quantiles and reporting how they have practically been applied and found useful by domain experts.

More recently, SG have been also analyzed with multiple objectives; [46] provides an overview and implementation of existing algorithms for Pareto frontier computation for multi-objective total reward, reachability, and probabilistic LTL properties as well as mixtures thereof. However, the computation is limited to stopping SGs, i.e. ones without end components

Multiple mean-payoff objective was first examined in [8] and both the qualitative and the quantitative problems are coNP-complete [16]. Although Boolean combinations of mean-payoff are undecidable in general [47], in certain subclasses of SG they can be approximated [9]. Boolean combinations of total-reward objectives were approximated in the case of stopping games [23] and applied to autonomous driving [25], where LTL is reduced to total reward in the case of stopping games and, for dimension two, the problem is shown decidable in [13].

PRISM-Games [38] provides tool support for several multi-player multi-objective settings [39]. Other tools supporting multi-player settings, GAVS+ [26] and GIST [20], are not maintained any more and are limited to single-objective settings.

In many settings, Pareto frontiers can be ε-approximated in polynomial time [42]. Pareto frontiers are constructed for the generalized mean-payoff objective for 2-player (non-stochastic) games in [14], MDPs in [10, 21], and SGs in [9]. For the generalized-reachability, the Pareto frontier is approximated for MDP in [30], but for SG the Pareto frontier is not even known to be given by finitely many points, except for dimension two [13]. In contrast, in the single-dimensional case, the value is known to be a multiple of a denominator that can be calculated from the syntactic description of the game [19].

Structure of the paper After recalling the basic notions in Section 2, we illustrate the problem, the difficulties and our solution on examples in Section 3. The algorithm is described and the correctness intuitively explained in Section 4 and formally proven in Section 5. The proofs of several technical statements are, for the sake of readability, relayed to [3, Appendix]. We conclude in Section 6.

2 Preliminaries

2.1 Stochastic Games

A probability distribution on a finite set X is a mapping $\delta : X \to [0,1]$, such that $\sum_{x \in X} \delta(x) = 1$. The set of all probability distributions on X is denoted by $\mathcal{D}(X)$. Given a dimension $n \in \mathbb{N}$, often implicitly clear from context, and $c \in \mathbb{R}$, we let $\vec{c}$ denote the n-dimensional vector with all components equal to c. For a vector $\vec{v}$, its i-th component is denoted $\vec{v}_i$. We compare vectors component-wise, i.e. $\vec{u} \leq \vec{v}$ if $\vec{u}_i \leq \vec{v}_i$ for all i. In this paper, we restrict ourselves to non-negative vectors, i.e. elements of $\mathbb{R}^n_{\geq 0}$

Now we define turn-based two-player stochastic games. As opposed to the notation of e.g. [27], we do not have special stochastic nodes, but rather a probabilistic transition function.

Definition 2.1 (SG). A *stochastic game (SG)* is a tuple $(S, S_\Box, S_\bigcirc, s_0, A, Av, \delta)$, where S is a finite set of *states* partitioned into the sets $S_\Box$ and $S_\bigcirc$ of states of the player *Maximizer* and *Minimizer*, respectively, $s_0 \in S$ is the *initial* state, A is a finite set of *actions*, $Av : S \to 2^A$ assigns to every state a set of *available* actions, and $\delta : S \times A \to \mathcal{D}(S)$ is a *transition function* that given a state s and an action $a \in Av(s)$ yields a probability distribution over *successor* states.

A Markov decision process (MDP) is then a special case of SG where $S_\bigcirc = \emptyset$. We assume that SG are non-blocking, so for all states s we have $Av(s) \neq \emptyset$.

For a state s and an available action $a \in Av(s)$, we denote the set of successors by $Post(s,a) := \{s' \mid \delta(s,a,s') > 0\}$. We say a state-action pair (s,a) is an *exit* of a set of states T, written (s,a) exits T, if $\exists t \in Post(s,a) : t \notin T$, i.e., if with some probability a successor outside of T could be chosen. Further, we use $Exits(T) = \{(s,a) \mid s \in T, a \in Av(s), (s,a) \text{ exits } T\}$ to denote all exits of a state set $T \subseteq S$. Finally, for any set of states $T \subseteq S$, we use $T_\Box$ and $T_\bigcirc$ to denote the states of T that belong to Maximizer and Minimizer, whose states are drawn in the figures as $\Box$ and $\bigcirc$, respectively.

The semantics of SG is given in the usual way by means of strategies and the induced Markov chain [7] and its respective probability space, as follows. An *infinite path* ρ is an infinite sequence $\rho = s_0 a_0 s_1 a_1 \cdots \in (S \times A)^\omega$, such that for every $i \in \mathbb{N}$, $a_i \in Av(s_i)$ and $s_{i+1} \in Post(s_i, a_i)$. *Finite path*s are defined analogously as elements of $(S \times A)^* \times S$. A *strategy* of Maximizer or Minimizer is a function $\sigma : (S \times A)^* \times S_\Box \to \mathcal{D}(A)$ or $(S \times A)^* \times S_\bigcirc \to \mathcal{D}(A)$, respectively, such that $\sigma(\rho s) \in \mathcal{D}(Av(s))$ for all s. We call a strategy *deterministic* if it maps to Dirac distributions only; otherwise, it is *randomizing*. A pair (σ, τ) of strategies of Maximizer and Minimizer induces an (infinite state) Markov chain $G^{\sigma,\tau}$ with finite paths as states, s_0 being initial, and the transition function $\delta(ws, wsas') = \sigma(ws)(a) \cdot \delta(s,a,s')$ for states of Maximizer and analogously for states of Minimizer, with σ replaced by τ. The Markov chain induces a unique probability distribution $\mathbb{P}^{\sigma,\tau}$ over measurable sets of infinite paths [7, Ch. 10] (the usual index with the initial state is not used since it is fixed already in the game).

2.2 End Components

Now we recall a fundamental tool for analysis of MDP called end components. An end component of a SG is then defined as the end component of the underlying MDP with both players unified.

Definition 2.2 (EC). A non-empty set $T \subseteq S$ of states is an *end component (EC)* if there is a non-empty set $B \subseteq \bigcup_{s \in T} Av(s)$ of actions such that

1. for each $s \in T, a \in B \cap Av(s)$, we have $(s,a) \notin Exits(T)$,

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.

Intuitively, ECs correspond to bottom strongly connected components of the Markov chains induced by possible strategies. Hence for some pair of strategies all possible paths starting in an EC remain there. An EC T is a *maximal end component (MEC)* if there is no other end component T' such that $T \subseteq T'$. Given an SG G, the set of its MECs is denoted $\mathsf{MEC}(G)$ and can be computed in polynomial time [29].

2.3 Generalized Reachability

For a set $T \subseteq S$, we write $\Diamond T := \{s_0 a_0 s_1 a_1 \dots \in (S \times A)^\omega \mid \exists i \in \mathbb{N} : s_i \in T\}$ to denote the (measurable) set of all paths which eventually reach T. A *generalized-reachability objective* (of dimension n) is an n-tuple $\mathcal{T} = (T_1, \dots, T_n)$ of state sets $T_i \subseteq S$. A vector $\vec{v}$ (of dimension n) is *achievable* if there is a strategy σ of Maximizer such that for all strategies τ of Minimizer

$$\forall i \in \{1, \dots, n\} \quad \mathbb{P}^{\sigma,\tau}(\Diamond T_i) \geq \vec{v}_i$$

Note that, since these games are not determined [23], this corresponds to the lower value, i.e. the worst case analysis.

For a given state s, the set of points achievable *from* s, meaning in a game where the initial state is set to s, is denoted $\mathfrak{A}_{\mathcal{T}}(s)$ or just $\mathfrak{A}(s)$ when $\mathcal{T}$ is clear from context.

2.4 Basic Geometry Notation and Pareto Frontiers

In order to consider convex combinations of sets, we define scaling of a set $X \subseteq \mathbb{R}^n$ by a constant $c \in [0, 1]$ as $c \cdot X = \{c \cdot x \mid x \in X\}$, and the Minkowski sum of sets X and Y as $X + Y = \{x + y \mid x \in X, y \in Y\}$. The convex hull of a set X is denoted by $conv(X) = \{\sum_{i=1}^{k} a_i x_i \mid k \geq 1, x_i \in X, a_i \geq 0, \sum_{i=1}^{k} a_i = 1\}$.

A *downward closure* of a set X of vectors is $dwc(X) := \{y \mid \exists x \in X : y \leq x\}$. A set X is *downward closed* if $X = dwc(X)$. The set $\mathfrak{A}$ of achievable points is clearly downward closed.

It will be convenient to use a few basic notions of projective geometry, which we now recall. Intuitively, a *direction* is a ray from the origin $\vec{0}$ into the (n-dimensional) first quadrant. As such, we may represent it with any vector $\vec{v} \neq \vec{0}$ on that ray. Then all vectors $\lambda \cdot \vec{v}$ for any $\lambda \in \mathbb{R}_{>0}$ are equivalent and represent the same direction. For instance, direction $\mathbf{d} = [(1, 0, 0)]$ denotes the x-axis and it is equal to $[(\lambda, 0, 0)]$ for any $\lambda > 0$. Formally, a direction $\mathbf{d} = [\vec{v}]$ is the set $\{\lambda \cdot \vec{v} \mid \lambda \in \mathbb{R}_{>0}\}$. We denote by $\mathsf{D} = \{[\vec{v}] \mid \vec{v} \in dwc(\{\vec{1}\})\}$ the set of all directions (in the first quadrant).

Given a set X of points and a direction $\mathbf{d}$, X *evaluated in direction* $\mathbf{d}$ is the (Euclidean) length of the vector from the origin to the farthermost intersection of X and $\mathbf{d}$, denoted

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

with the usual $\sup \emptyset = 0$. Fig. 1 illustrates an evaluation of a direction on an achievable set. Intuitively, it describes what is achievable if we prefer the dimensions in the "ratio" given by $\mathbf{d}$. Another example is the whole set (blue and red) of Fig. 4a: evaluated in $[(1, 1)]$ it yields $\sqrt{2}/2$.

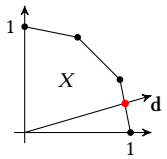


Figure 1. Example showing a Pareto frontier of a set X, a direction $\mathbf{d}$, and the point of intersection of $\mathbf{d}$ with the frontier, depicted as ● in distance $X[\mathbf{d}]$ from the origin.

Given a downward closed set X, its *Pareto frontier* is the set of farthermost points in each direction:

$$\mathfrak{P}(X) = \{\vec{x} \mid \mathbf{d} \in \mathsf{D}, \mathbf{d} = [\vec{x}], X[\mathbf{d}] = ||\vec{x}||\}$$

The *Pareto frontier of a state* s is the Pareto frontier of the set achievable in s, i.e. $\mathfrak{P}(s) := \mathfrak{P}(\mathfrak{A}(s))$. The *Pareto set of the game* is $\mathfrak{P} := \mathfrak{P}(s_0)$. Thus by definition, $\mathfrak{P} = \mathfrak{P}(\mathfrak{A}(s_0))$ and, further, $dwc(\mathfrak{P})$ is (the closure of) $\mathfrak{A}(s_0)$.[1] Note that it is not known whether $\mathfrak{A}$ is closed, since it is not known whether the suprema of achievable points are also achievable. Our notion of $\mathfrak{P}$ includes these suprema, which is why it is only equal to the closure of $\mathfrak{A}$.

2.5 Problem Formulation

In this paper, we are interested in ε-approximating $\mathfrak{P}$. In terms of under- and over-approximation:

> Given an SG, generalized-reachability objective $\mathcal{T}$, and precision $\varepsilon > 0$, the task is to construct sets $\mathcal{L}, \mathcal{U} \subseteq \mathbb{R}^{|\mathcal{T}|}$ such that for each direction $\mathbf{d} \in \mathsf{D}$, $\mathcal{L}[\mathbf{d}]$ and $\mathcal{U}[\mathbf{d}]$ are effectively computable and we have
>
> $$\mathcal{L}[\mathbf{d}] \leq \mathfrak{P}[\mathbf{d}] \leq \mathcal{U}[\mathbf{d}] \quad \text{and} \quad \mathcal{U}[\mathbf{d}] - \mathcal{L}[\mathbf{d}] < \varepsilon \,.$$

2.6 Multi-dimensional and Bounded Value Iteration

In this section we recall two extensions of the standard value iteration: a generalization for *multi-dimensional* objectives and a *"bounded"* one with an over-approximating sequence. Firstly, the multi-dimensional Bellman operator for reachability, e.g. [23],

$$\mathfrak{B} : \left(S \to 2^{[0,1]^n}\right) \to \left(S \to 2^{[0,1]^n}\right)$$

works with *sets* $X(s)$ of points achievable in s rather than single points:

[1] Our notion of Pareto frontier captures the whole surface in the first quadrant. Other definitions such as $\mathfrak{P}_{\mathcal{T}} = \{\vec{v} \mid \vec{v} \text{ is achievable} \wedge \forall \text{ achievable } \vec{u} : \vec{u} \not> \vec{v}\}$ only capture the Pareto optimal points. For example, if the set of achievable points in the three-dimensional space is the whole unit cube then our definition returns its three sides, while the other definition returns only the singleton with the Pareto optimal point $(1, 1, 1)$.

$$\mathfrak{B}(X)(\mathsf{s}) = \begin{cases} \bigcap_{a \in \mathsf{Av}(\mathsf{s})} X(\mathsf{s},\mathsf{a}) & \text{if } \mathsf{s} \in S_{\bigcirc} \\ conv(\bigcup_{a \in \mathsf{Av}(\mathsf{s})} X(\mathsf{s},\mathsf{a})) & \text{if } \mathsf{s} \in S_{\square} \end{cases}$$

where we define

$$X(\mathsf{s},\mathsf{a}) = \left(dwc(\{\mathbb{1}_{\mathcal{T}}(\mathsf{s})\}) + \sum_{\mathsf{s}' \in S} \delta(\mathsf{s},\mathsf{a},\mathsf{s}') \cdot X(\mathsf{s}') \right) \cap \mathbf{1}$$

and $\mathbb{1}_{\mathcal{T}}$ is the indicator vector function of target sets, i.e. $\mathbb{1}_{\mathcal{T}}(\mathsf{s})_i$ equals 1 if $\mathsf{s} \in T_i$ and 0 otherwise, and $\mathbf{1} = dwc(\{\vec{1}\})$ is the unit box.

Intuitively, the operator works as follows. Given what can be achieved from s using now an action a, we can compute the value for the minimizing state as the intersection over all actions since these points are achievable no matter what Minimizer does. For maximizing states, if there exists an action achieving a point then Maximizer can achieve it from here; moreover, we compute the convex hull since Maximizer can also randomize and, as opposed to the minimizing case with intersection, union of convex sets need not be convex. Once we have dealt with decision making on the first line, it remains to determine what can be achieved by each decision, on the second line. The achievable values are given by the weighted average of the successors' values, but additionally, the base case of targets must be handled. Namely, whenever a state is in a target set, all values up to 1 in that dimension are achievable (but not greater than 1).

This also gives rise to an algorithm approximating $\mathfrak{A}$, which is the least fixpoint of $\mathfrak{B}$ [23]. We initialize $\mathsf{L} : S \to 2^{[0,1]^n}$ to return $\{\vec{0}\}$ everywhere and iteratively apply the Bellman operator, yielding arbitrarily precise approximations of $\mathfrak{A}$ by $\mathfrak{B}^k(\mathsf{L})$ as $k \to \infty$ [23][2]. Moreover, for every state s it can be checked that the set $\mathfrak{B}^k(\mathsf{L})(s)$ is presented at each step k as a closed downward-closed convex polyhedron, i.e. a *finite* object. Thus we can effectively construct any desired approximation.

However, it is not known how to bound the difference of the actual achievable set $\mathfrak{A}$ and the approximation after k iterations. For that reason, [37] introduced for the single-dimensional case the *bounded value iteration* (named along the tradition of [41]), a way to compute also an over-approximating sequence. If we initialize U to return $dwc(\{\vec{1}\})$ everywhere[3], then $\lim_{k\to\infty} \mathfrak{B}^k(\mathsf{U})$ is a fixpoint, which is generally different from the least one. Hence [37] modifies $\mathfrak{B}$ so that it has a single fixpoint equal to the least one of the original $\mathfrak{B}$. Then both the sequence of lower bounds and of upper bounds converge to $\mathfrak{A}$, the value of the game. The modification is demonstrated in the next section, where we also illustrate the main ideas how to cope with the multi-dimensional case.

[2]Precisely, $\lim_{k\to\infty} \mathfrak{B}^k(\mathsf{L}) \subseteq \mathfrak{A} \subseteq dwc(\mathfrak{P}) = \mathrm{clos}(\lim_{k\to\infty} \mathfrak{B}^k(\mathsf{L}))$ where clos is the standard closure in $\mathbb{R}^n$.

[3]The same holds even if we initialize to 0 all the dimensions i in states from which there is no path to T_i, as is customary in MDP analysis. The solution of [37] is not sensitive to this and does not require this special treatment in the initialization of U.

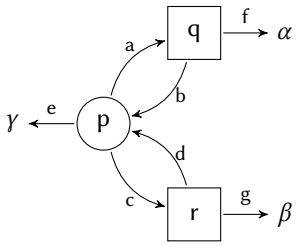


Figure 2. An example demonstrating the complications arising in an end component.

3 Example

In this section, we illustrate the issues preventing convergence of the upper bounds, as well as the solution of [37] and our extension of it. Value iteration converges if the SG is stopping, i.e. if the game reaches a designated sink with probability 1, or equivalently, if there are no end components (ECs). Hence the difficulty in solving reachability SG is rooted in ECs, as it is possible to cycle in its states infinitely long. As a running example, consider the EC in Fig. 2 with states p, q, r and actions a, . . . , g. The symbols α, β and γ are placeholders; in the single dimensional case, they represent a real number; in the multi dimensional case, a Pareto frontier. One can make this game a standard SG in the single dimensional case by, for example, replacing α with a transition that reaches the target with probability α and the sink with probability $1 - \alpha$. The multi-dimensional case is a straightforward extension.

We start by considering the single-reachability objective. The standard Bellman update procedure as described in Section 2.6 reduces to the following equations, where intersections become minima and unions become maxima. We write U_k as short for $\mathfrak{B}^k(\mathsf{U})$.

$$\begin{aligned} \mathsf{U}_{i+1}(\mathsf{p}) &= \min\{\mathsf{U}_i(\mathsf{q}), \mathsf{U}_i(\mathsf{r}), \gamma\} \\ \mathsf{U}_{i+1}(\mathsf{q}) &= \max\{\mathsf{U}_i(\mathsf{p}), \alpha\} \\ \mathsf{U}_{i+1}(\mathsf{r}) &= \max\{\mathsf{U}_i(\mathsf{p}), \beta\} \end{aligned}$$

By replacing U with L, we get the update equations for the lower bound. Recall that we initialize L_0 to return 0 everywhere and U_0 to return 1 everywhere.

3.1 MDP

Firstly, let us briefly mention the solution of [11, 35] for MDP. Suppose that all states belonged to the maximizing player, i.e. p was also maximizing. Then, the initialization $\mathsf{U}_0 = \vec{1}$ is already a fixpoint, although the true value of all three states is $\max\{\alpha, \beta, \gamma\}$. Intuitively, the reason for this is that the equations create a cyclic dependency: the process of finding the value by "asking neighbours" is not well-founded and all states

falsely believe that they can achieve the higher value 1. [37] calls such an EC *bloated*, having an unjustifiably large (bloated) upper bound. The solution of [11, 35] is to detect that this is an EC and collapse it into a single state, eliminating the cycle. Only outgoing actions α, β, γ of the EC are kept, and in the next iteration, the Bellman operator correctly sets the value of the collapsed state to $\max\{\alpha, \beta, \gamma\}$, thus converging to the true value. The solution of [37] captures this idea from a different perspective: It does not change the underlying graph, but instead realizes that all three states can reach the *"best exit"* of the EC, i.e. the state with an action exiting the EC and having the highest value. Then the algorithm reduces the upper bounds of the states of the EC to that of the best exit. This is called *deflating*, as the "internal higher pressure" of bloated upper bounds is "relieved", equalizing with the best exit.

3.2 Single-reachability SG

Secondly, for single-reachability *SG*, the EC cannot in general be collapsed, since the values of the states differ, and it is not clear a priori which states share a value. They depend on the *ordering* of the values of the exits, i.e. on the ordering of α, β and γ.

Case 1: If $\gamma < \min(\alpha, \beta)$, then after the first iteration we have $U_1(p) = \gamma$, $U_1(q) = 1$ and $U_1(r) = 1$. After the next iteration, $U_2(p) = \gamma$, $U_2(q) = \alpha$ and $U_2(r) = \beta$. These are the true values, as observable in Figure 2. In this case U_k converges to the value. However, note that the values of the states in the same EC are different.

Case 2: If $\gamma \geq \min(\alpha, \beta)$, and say $\alpha > \beta$, then the values of p and r are β and that of q is α. This is the case, because p will always play action c, not allowing state r to achieve anything but the smallest value β. However, U_k does not converge to these values. In the first iteration, $U_1(p) = \gamma$, $U_1(q) = 1$ and $U_1(r) = 1$. After the next iteration, $U_2(p) = U_2(q) = U_2(r) = \gamma$. After this, the upper bounds do not change any more, because we have the problem of cyclic dependencies as described in Section 3.1. If we fix the strategy of the Minimizer to c as that is the best choice, only $\{p, r\}$ forms an EC. The value of both p and r is β, as that is the best exit that the Maximizer can achieve, given that Minimizer does not play the suboptimal action a. Such an EC where all states share the same value is called simple end component (SEC) [37]. It is simple, because after fixing the strategy of Minimizer to be optimal, this player cannot influence the play anymore (as the SG locally becomes an MDP). In the SEC, Maximizer can direct the play to the best exit and almost surely achieve the value of it. Deflating the SEC $\{p, r\}$, i.e. setting the upper bound for all states in the SEC to that of the best exit, correctly updates the bounds to β. Afterwards, the upper bound of q is correctly set to α in the next iteration. So one would like to find and deflate all SECs.

However, which states form a SEC depends on the relative ordering of the exits' values and the corresponding choices that Minimizer makes (recall we had to fix the strategy of p to the optimal action c in order to realize which states form the SEC). Indeed, in the case with $\alpha < \beta$, a different SEC ($\{p, q\}$) should be deflated and if $\alpha = \beta$ then all three states form a SEC. Since we do not know the values of the exits, the algorithm uses the approximations (L_i) to guess which actions are suboptimal for the Minimizer, and hence which states form a SEC. As the lower approximation converges to the value, the true SECs are eventually detected and correctly deflated. However, when L_i is not yet close enough to the value, the computation of SECs can be wrong, e.g. if $\alpha < \beta$, but for the first few iterations of the algorithm the lower bound on β is smaller than that on α. Then, for these first iterations, the algorithm believes $\{p, r\}$ to be the SEC, and only afterwards realizes that it actually is $\{p, q\}$. Hence, the operation we perform on the SEC has to be conservative, i.e. sound even if it is given a set of states that actually do not form a SEC. This is why deflating was introduced, as it is sound for any EC, even ones that are not SECs [37, Lemma 3]. In contrast, modifying the underlying graph by collapsing as in [11, 35] would commit to the detected SEC-candidate and thereby possibly make the wrong choice. Note that we never know that we have correctly detected a SEC, we just know that in the limit we will eventually detect it.

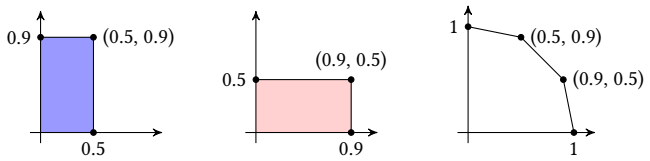


Figure 3. Pareto frontiers of α (left), β (center) and γ (right) in a 2-objective setting. X-axis represents the value along the first objective and Y-axis represents the value along the second objective.

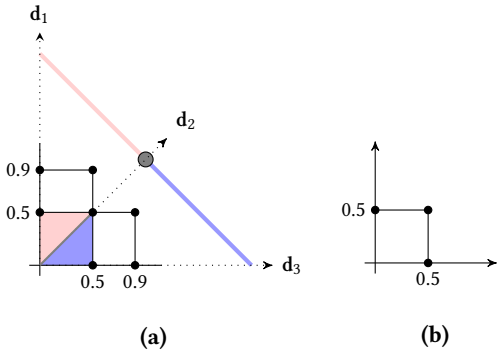


Figure 4. (a) Visualizing the regions for state p; and (b) the result of deflating.

3.3 Generalized-reachability SG

Here we intuitively describe and illustrate the main elements of our solution. The formal definitions of the key concepts only follow in the next section.

Regions. Consider again the example of Fig. 2. In the multi-dimensional case, instead of α, β and γ being reals, they are sets of achievable vectors. Let them be given as in Fig. 3, so e.g. $\alpha = dwc(\{(0.5, 0.9)\})$. Here γ gives the highest values, so it is the best one for Maximizer, and hence Minimizer will not play the corresponding e (as in Case 2 in Section 3.2). α and β, however, cannot be compared. Depending on the trade-off (corresponding to a direction) that Maximizer wants to achieve, α or β might be better than the other. To this end, let $\mathbf{d}$ be the direction in which Maximizer wants to maximize. Depending on $\mathbf{d}$, Minimizer's behaviour changes. If the objective along the x-axis is more important, then Minimizer chooses action a. This way, the value of the more important objective is restricted to 0.5. If on the other hand, the objective along y-axis is more important, then the Minimizer chooses action c. The Minimizer, for each direction $\mathbf{d}$, decides on the action to be chosen by comparing α and β evaluated in that direction; in other words, by computing the minimum of $\alpha[\mathbf{d}]$ and $\beta[\mathbf{d}]$.

Our algorithm identifies finitely many *regions* where the Minimizer has the same preference ordering over actions and then we deflate each region separately. In our example, we can identify three regions, as shown in Fig. 4a. Between the directions $\mathbf{d}_1$ and $\mathbf{d}_2$ (red ● region), Minimizer's best choice is action c; between $\mathbf{d}_2$ and $\mathbf{d}_3$ (blue ● region), Minimizer's best choice is action a; and along $\mathbf{d}_2$ (grey ● line), Minimizer is indifferent.

Deflating regional SECs. Once restricting to a region fixes the preference ordering over Minimizer's actions, we can proceed as in the single-dimensional case: We fix Minimizer's optimal strategy based on the lower bounds, identify SEC-candidates and deflate them. That means we update the Pareto frontier in the region to that of the *best exit* from the SEC. The whole Pareto frontier is constructed piece by piece, region by region.

Returning to our running example, we have already identified the three regions in the Pareto frontier for state p in Figure 4a. The SECs depending on the regions are as follows: In the blue ● region it is {p, q}, in the red ● region it is {p, r}, and along $\mathbf{d}_2$ all three states form a SEC. Deflating the blue ● region, we see that the best exit from the SEC has value α, so between 0° and 45° the value of p is set to the corresponding part of α. Doing the same for the other two regions results in the Pareto frontier depicted in Figure 4b. This result is also intuitively expected, as depending on which direction Maximizer prefers, Minimizer can always restrict the play to the other exit. Note that for the sake of example here we always talked about the true values, while the algorithm does not know these precisely. Therefore, deflating cannot update the values based on the value of α, but only on its approximation. Being on the safe side, the values will be decreased only to its over-approximation.

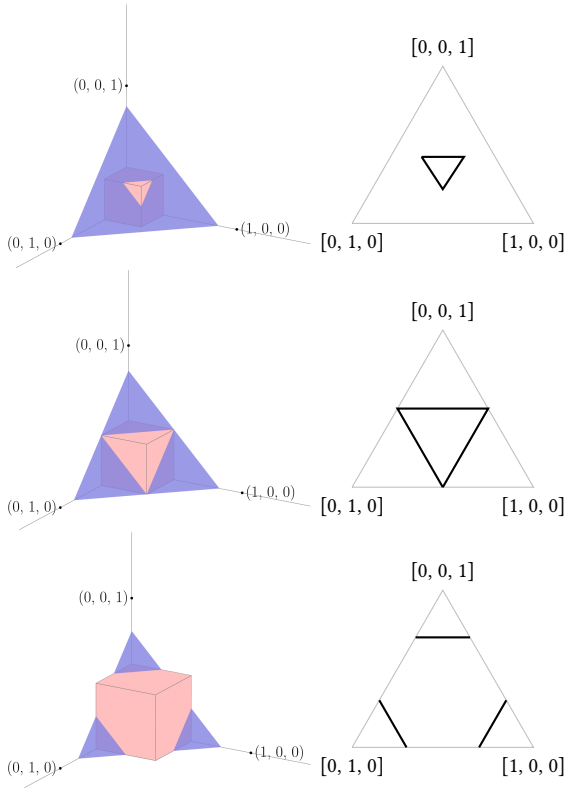


Figure 5. The left column shows Pareto frontiers; there is the blue tetrahedon and a pink box of varying size. The right column shows the projection of the intersection onto the projective hyperplane.

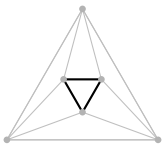
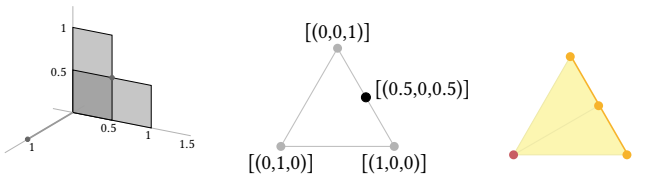


Figure 6. Triangulation of the top right of Fig. 5

Computing and representing regions. As explained above, a region depends on the preference ordering of actions. To compute regions where this ordering is constant, we use geometric methods. In the example of Fig. 4a, the point where the preference ordering changes is $(0.5, 0.5)$, which is where the two Pareto frontiers intersect. So, intuitively, by drawing the Pareto frontiers and finding the points of intersection, we can identify the regions (sets of the corresponding directions) where the preference ordering over actions is constant.

In Figure 5, we give a set of three examples to illustrate the construction of regions. The left picture in each row of the figure shows two Pareto frontiers: One is the blue tetrahedron, generated by Maximizer's free, but exclusive choice between target sets. The other is a red box of different sizes, generated by the possibility to reach a state in all target sets with a given probability. From top to bottom, we increase this



(a) Three sets of achievable points

(b) Projection of the intersections

(c) Convex regions as a simplicial complex

Figure 7. Projections of intersections of Pareto frontiers to the projection plane, which in 3D is the triangle formed by the points $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$. In Fig. 7b, labels represent directions and not individual vectors.

probability, thereby increasing the size of the box, yielding three different examples. We define regions as sets of directions. In order to draw directions, it is useful to consider the so-called *projective hyperplane*. It is the set of all directions and can be drawn (in our case with non-negative vectors only) as a triangle with corners $[1,0,0]$, $[0,1,0]$, $[0,0,1]$, capturing all directions. When a point (vector) $\vec{v}$ is projected into its direction $[\vec{v}]$, it intuitively corresponds to drawing a ray from the origin through the point $\vec{v}$. If we identify the projective hyperplane with the hyperplane passing through the *points* $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$ (or more precisely with this triangle) then the intersection of the ray and the hyperplane, say point $\vec{p}_v$, is the *projection* of $\vec{v}$ to the projective hyperplane. In our example, the right side of the figure shows the projection of the intersection of the Pareto frontiers onto the projective hyperplane. This gives rise to three regions, each with different preference ordering: the inner open triangle, its boundary and the outer triangle with the hole. Minimizer prefers the red action in the outside triangle, the blue one in the inside triangle, and is indifferent on the boundary. As these regions are hard to describe (as well as possibly not even convex and connected), we triangulate the projections to get smaller regions which are convex and generated by finitely many points. The triangulation of the top right of Figure 5 is depicted in Figure 6. Further note that while the preference ordering of actions is constant in each region, the faces of a region represent turning points of the preference ordering; hence these faces need to be separate regions like is customary for timed automata [1]. Hence in order to represent the regions, we thus decompose the triangle (generally, in higher dimensions, a *simplex*) into open triangles, open line segments and points (in general into a *simplicial complex*, i.e. the simplex together with its faces and recursively their faces).

As another example of the projection to the projective hyperplane and the triangulation, consider Figure 7a with three achievable sets: two rectangles – $dwc(\{(1,0,0.5)\})$ and $dwc(\{(0.5,0,1)\})$; and one line – $dwc(\{(0,1,0)\})$. The frontiers of the sets generate only one non-empty intersection[4], namely the point $(0.5,0,0.5)$. Its projection is represented by its direction, $[(0.5,0,0.5)]$ in Figure 7b. In order to keep the representation of regions effective, we again triangulate regions into finer ones, which are convex and generated by finitely many points, see Fig. 7c. Finally, note that Pareto frontiers of smaller dimensions may induce regions that are faces of the projective hyperplane (triangle). In this example, the vertex at $[(0,1,0)]$ is its own region, as it is the only direction where playing the line-action is not optimal for Minimizer. We can also see that in Fig. 4b: the red vertex corresponds to Minimizer choosing one of the "rectangular" actions (as the other action is suboptimal), the orange region to choosing the action yielding $(0,1,0)$, and in the yellow Minimizer is indifferent between all actions. Since these cases only arise on faces of the projective hyperplane, the decomposition into the simplicial complex of the projective hyperplane (triangle) caters for these corner cases. Note that for identifying the regions, we considered the point $[(0.5,0,0.5)]$, which is the turning point of preference between the two rectangles. As both of them are suboptimal in this direction, this is not necessary to get the coarsest partition. However, it is not a problem to use a finer partition (splitting the orange line and the yellow triangle), as we still have the invariant that in every region the strategy of Minimizer is constant.

4 Algorithm

4.1 Lifting the concepts from the single-dimensional case

Before giving the algorithm, we have to define extensions of the concepts of best exit and simple end component (SEC) introduced in [37] to the multi-objective setting, as intuitively discussed in the previous section. To this end, we also introduce the concept of *regions*.

Best exits. In the single-dimensional case, the best exit of an EC was just the best exiting action for the Maximizer. In the multi-dimensional setting, Maximizer cannot only pick the best exit, but first visits all targets inside the (S)EC and then use any combination of exits to achieve any desired tradeoff. The definition of best exit depends on a parameter f. This function is used to calculate the set of achievable points from an exit. We can instantiate it with $\mathfrak{A}$ to denote the actual set of achievable points, as well as with the over-approximation U; in the algorithm, we do the latter, as we do not know $\mathfrak{A}$.

Thus we define the best exit in the multi-dimensional setting (similarly as $X(s,a)$ in Section 2.6):

$$\mathsf{BE}^f(T) := \left(dwc\Big(\sum_{s\in T}\{\mathbb{1}_{\mathcal{T}}(s)\}\Big) + conv\Big(\bigcup_{(s,a)\in \mathsf{Exits}(T_\square)} f(s,a)\Big)\right) \cap \mathbf{1}$$

[4]The neutral element $\{\vec{0}\}$ is not considered a non-empty intersection.

The first part ensures that, if a target is in the EC, all states in the EC have probability 1 to reach it; the second part takes the convex hull of the union of (Pareto sets of) all of Maximizer's exits, corresponding to randomizing over the exiting actions. For general ECs, this may give a strict over-approximation since Minimizer might prevent Maximizer from freely visiting all states and combining all actions. However, for SECs the expression is later shown exact. Note that here we use the convention $\bigcup_{\emptyset}(\cdot) = \{\vec{0}\}$, which is a neutral and minimal element. This solves the corner case of an EC without any exit.

Regions. The extension of SEC works only when partitioning the set of all possible directions into *regions*, and then applying the same ideas as in the single-dimensional case in each region separately.

Definition 4.1 (Region)**.** A *region* is a subset $R \subseteq \mathsf{D}$ of directions.

To keep the presentation simple, we rely on a very general definition of regions at this point. We will see later in Section 4.3 how we can restrict to handling only regions that correspond to a finitely generated cone. In the following, slightly abusing notation, we sometimes view a region R as the set of points it contains, i.e. $\{v \in [0,1]^n \mid \exists \mathbf{d} \in R : [v] = \mathbf{d}\}$.

Simple ECs. In the single-dimensional case, the idea of SEC is the following: If Minimizer fixes their strategy to the optimal strategy (i.e. ignores all suboptimal actions), and in the remaining game there still exists an EC, then this EC is simple. It is the best choice of Minimizer to allow Maximizer to roam around freely in the SEC and pick the best exit. Thus, all states in the SEC have the same value, namely that of the best exit (recall, best for Maximizer).

In the multi-dimensional case, the optimal strategy of Minimizer depends on the tradeoffs between the different goals. This is why, to generalize the concept of SEC, we need to add the restriction that a set of states is a SEC for some region R, as the trade-offs between the goals are resolved in the same way in the whole region, or in other words: where the optimal strategy of Minimizer is the same for all directions in R. Formally:

Definition 4.2 (Regional *SEC*)**.** An EC T is *a regional simple end component for some region* R, if for every direction $\mathbf{d} \in R$ and all states $s \in T$, $\mathfrak{A}(s)[\mathbf{d}] = \mathsf{BE}^{\mathfrak{A}}(T)[\mathbf{d}]$.

Note that from this definition we also know that all states in the regional *SEC* have the same value. Moreover, as we shall see, the definition implies that on R, the optimal strategy of Minimizer should be the same in all directions. Lifting this to a set of regions we have the following property:

Definition 4.3 (Consistent Partition)**.** Let T be an EC and $f\colon S \to 2^{\mathbb{R}^n}$. A partition of the set D of directions into a set of regions $\mathcal{R}$ is called *consistent* w.r.t. T and f if for all $R \in \mathcal{R}$ and all $\mathbf{d}_1, \mathbf{d}_2 \in R$, $s \in T_{\bigcirc}$ and $a \in \mathsf{Av}(s)$ it holds that

$$f(s,a)[\mathbf{d}_1] = \min_{b \in \mathsf{Av}(s)} f(s,b)[\mathbf{d}_1] \iff f(s,a)[\mathbf{d}_2] = \min_{b \in \mathsf{Av}(s)} f(s,b)[\mathbf{d}_2].$$

In the other direction, we shall see that every possible regional SEC can be defined on regions of an arbitrary consistent partition. Hence, algorithmically, we shall be looking for such partitions first and then for regional SECs.

Algorithm 1 Multi-Objective Bounded Value Iteration

Input:
SG G, generalized-reach. objective $\mathcal{T}$, precision ε
Output:
$\mathcal{L}, \mathcal{U}$ such that $\forall \mathbf{d} \in \mathsf{D} : \mathcal{L}[\mathbf{d}] \leq \mathfrak{P}[\mathbf{d}] \leq \mathcal{U}[\mathbf{d}]$ and $\mathcal{U}[\mathbf{d}] - \mathcal{L}[\mathbf{d}] < \varepsilon$

1: **procedure** MO-BVI(G,$\mathcal{T}$,ε)
2: **for** each s $\in S$ **do** ▷ Initialization
3: L(s) $\leftarrow \{\vec{0}\}$ ▷ to the least and
4: U(s) $\leftarrow dwc(\{\vec{1}\})$ ▷ the greatest values
5: **repeat** ▷ The new Bellman update $\overline{\mathfrak{B}}$
6: L $\leftarrow \mathfrak{B}$(L) ▷ Standard Bellman updates
7: U $\leftarrow \mathfrak{B}$(U)
8: U $\leftarrow$ DEFLATE_SECs(G, L, U) ▷ New treatment
9: **until** $\max_{\mathbf{d} \in \mathcal{D}} \mathsf{U}(s_0)[\mathbf{d}] - \mathsf{L}(s_0)[\mathbf{d}] < \varepsilon$ ▷ ε-approximate
10: **return** $(\mathsf{L}(s_0), \mathsf{U}(s_0))$

4.2 Algorithms

We present our overall bounded VI procedure as Algorithm 1. In the following, we provide intuitive explanations of the algorithm and its sub-procedures, as well as the proofs for the lemmata on correctness of the sub-procedures. The correctness of the whole algorithm is proven in Section 5. Section 4.3 gives more details on the effectiveness of the computation in Algorithm 3, as that pseudocode is rather mathematical and it is not trivial to see that it is indeed effectively computable and yields an effective approximation.

Algorithm 1 (MO-BVI). initializes the under- and over-approximations L and U and updates them using the new Bellman update operator $\overline{\mathfrak{B}}$. This operator first performs the standard Bellman updates and then calls the procedure DEFLATE_SECs, which we exemplified in Section 3. The intuition of the whole algorithm is, that as the under-approximation converges, eventually the correct regional SECs are found and deflated. When all regional SECs are deflated, the over-approximation approaches the true set of achievable vectors in the limit. Note that the stopping criterion can be evaluated, as the under- and over-approximation are at all times described by finitely many points, for details see Section 4.3.

Algorithm 2 Deflate candidate SECs

Input:
SG G, functions L and U such that for all states $s : \mathsf{L}(s) \subseteq \mathfrak{A}(s) \subseteq \mathsf{U}(s)$
Output:
Updated upper bound U′
1: **procedure** DEFLATE_SECs(G, L, U)
▷ In each MEC, we compute relevant regions, find all candidate *SEC*s and decrease their upper bounds
2: $\mathsf{U}'(s) \leftarrow \{\vec{0}\}$ for all $s \in S$ ▷ Result variable
3: $\mathcal{M} \leftarrow \mathsf{MEC}(\mathsf{G})$ ▷ MEC decomposition of the game
4: **for** each $T \in \mathcal{M}$ **do**
5: $\mathcal{R} \leftarrow$ GET_REGIONS(T, L)
6: **for** each $R \in \mathcal{R}$ **do**
7: $\mathcal{S} \leftarrow$ FIND_SECs(T, L, R) ▷ Candidate SECs
8: **for** each $s \in T$ **do**
▷ If in candidate SEC, deflate
9: **if** $s \in C$ for some $C \in \mathcal{S}$ **then**
10: $\mathsf{U}'(s) \leftarrow \mathsf{U}'(s) \cup (\mathsf{U}(s) \cap \mathsf{BE}^{\mathsf{U}}(C) \cap R)$
▷ Otherwise, keep estimate in this region
11: **else**
12: $\mathsf{U}'(s) \leftarrow \mathsf{U}'(s) \cup (\mathsf{U}(s) \cap R)$
13: **return** U′

Algorithm 3 Compute consistent partition into regions

Input:
MEC $T \subseteq S$, function L such that for all states $s : \mathsf{L}(s) \subseteq \mathfrak{A}(s)$
Output:
Consistent partition $\mathcal{R}$ w.r.t. T and L
1: **procedure** GET_REGIONS(T, L)
2: $\mathcal{R} \leftarrow \{\mathsf{D}\}$ ▷ initialize with trivial partition
3: **for** $s \in T_{\bigcirc}$ **do**
4: **for** each $B \subseteq \mathsf{Av}(s)$ **do**
5: $R_B \leftarrow \{\mathbf{d} \in \mathsf{D} \mid B = \arg\min_{a \in \mathsf{Av}(s)} \mathsf{L}(s, a)[\mathbf{d}]\}$
6: $\mathcal{R}' = \{R_B \mid B \subseteq \mathsf{Av}(s), R_B \neq \emptyset\}$
7: $\mathcal{R} \leftarrow$ common refinement of $\mathcal{R}$ and $\mathcal{R}'$
8: **return** $\mathcal{R}$

Algorithm 2 (DEFLATE_SECs***).*** is the heart of our new algorithm. It implements the correct handling of end components, ensuring convergence of the upper and the lower approximation to the *same* fixpoint. As every SEC is an EC and every EC is a subset of a MEC, the algorithm first computes the MEC-decomposition. Then, for each MEC we compute a consistent partition of the set of directions into regions using Algorithm 3. Finally, Algorithm 2 updates the over-approximation of every state in the considered MECs. It does so piece by piece, region by region; this is why in Lines 10 and 12 we always intersect with R, restricting the update to

Algorithm 4 Find candidate SECs

Input: Under-approximation L, EC T, region R from a consistent partition w.r.t. T and L
Output: Set of Regional *SEC*s for R, according to L
1: **procedure** FIND_SECs(L,T,R) $T \subseteq S$, L, region R
2: $\mathbf{d} \leftarrow$ arbitrary element of R
3: $\mathsf{Av}' \leftarrow \mathsf{Av}$
4: **for** each $s \in T_{\bigcirc}$ **do**
▷ Keep only optimal Minimizer actions
5: $\mathsf{Av}'(s) \leftarrow \{a \in \mathsf{Av}(s) \mid \mathsf{L}(s, a)[\mathbf{d}] = \min_{b \in \mathsf{Av}(s)} \mathsf{L}(s, b)[\mathbf{d}]\}$
6: **return** $\mathsf{MEC}(T|_{\mathsf{Av}'})$ ▷ MEC decomposition on T with actions restricted to Av′

points in the current region, and take the union with the intermediate result U′, adding all the points from the previous iterations of the loop over $\mathcal{R}$. If a state is part of a regional SEC C (as detected by Algorithm 4), the upper bound in the current region is reduced to $\mathsf{BE}^{\mathsf{U}}(C)$, i.e. to the best exit from the regional SEC. If a state is not in a candidate SEC for the current region, its upper bound does not change. Note that the best exit depends on U, our current best over-approximation. The intersection with $\mathsf{U}(s)$ ensures that deflate is monotonic. Formally, we have the following:

Lemma 4.4 (DEFLATE is monotonic and sound)**.** *Given a game* G *with correct upper and lower bounds* U *and* L *(i.e.* $\forall s \in S : \mathsf{L}(s) \subseteq \mathfrak{A}(s) \subseteq \mathsf{U}(s)$*),* $\mathsf{U}' = \mathsf{DEFLATE\_SECs}(\mathsf{G}, \mathsf{L}, \mathsf{U})$ *has the following properties: For all states* $s \in S$*,*

- $\mathsf{U}'(s) \subseteq \mathsf{U}(s)$ *(Monotonicity),*
- $\mathfrak{A}(s) \subseteq \mathsf{U}'(s)$ *(Soundness),*

Proof. For monotonicity notice that due to line 10, $\mathsf{U}'(s)$ is obtained by intersecting $\mathsf{U}(s)$ with $\mathsf{BE}^{\mathsf{U}}(C)$ on each region $R \in \mathcal{R}$, which makes sure that $\mathsf{U}'(s) \subseteq \mathsf{U}(s)$ in the end. For the second item, we have that $\mathfrak{A}(s') \subseteq \mathsf{U}'(s')$ for all states s' by assumption. Recall that $\mathsf{BE}^{\mathsf{U}}(\mathfrak{A})$ is the set of points achievable from $s \in C$ assuming that Maximizer has control over all states in C. Clearly, $\mathfrak{A}(s) \subseteq \mathsf{BE}^{\mathfrak{A}}(C) \subseteq \mathsf{BE}^{\mathsf{U}}(\mathfrak{A})$, which proves soundness (see [3, Appendix A.1] for details). □

Algorithm 3 (GET_REGIONS***).*** has to return a consistent partition of the set of directions D, i.e. for all directions in a region, the optimal strategy of Minimizer needs to be the same. To do that, for every state in the given MEC, we partition the set of directions into regions according to the optimal strategy of Minimizer, i.e. which actions are optimal in the region[5]. Then we take the *common refinement* of all these partitions. The common refinement of two partitions $\mathcal{R}_1$ and $\mathcal{R}_2$ is defined as the coarsest partition $\mathcal{R}$ such that for all $R_1 \in \mathcal{R}_1$, $R_2 \in \mathcal{R}_2$

[5]The implementation suggested in Section 4.3 actually computes regions for all orderings of actions. It then describes the regions with the same optimal actions as a union of all regions where these actions are at the top of the ordering.

we have $R_1 \cap R_2 \in \mathcal{R}$. Notice that the common refinement of any number of consistent partitions (w.r.t. the same T and L) is again consistent. Intuitively, in every resulting region the strategy of all Minimizer states in the MEC is constant. Formally, we have the following lemma:

Lemma 4.5 (GET_REGIONS is sound)**.** *For any set of states T and bound function L, the set of regions $\mathcal{R}$ returned by procedure* GET_REGIONS*(T, L) is a consistent partition.*

Proof. We simply consider for every subset $B \subseteq \mathsf{Av}(s)$, $s \in T_{\bigcirc}$, the region R where the actions in B are all optimal. This yields a partition $\mathcal{R}' = \{R_B \mid B \subseteq \mathsf{Av}(s), R_B \neq \emptyset\}$ which is consistent w.r.t. $\{s\}$ and L. We repeat this for all $s \in T_{\bigcirc}$ and take the common refinement of all partitions obtained in this way, yielding a consistent partition for the whole EC T and L. See the next section on how to technically implement these operations effectively. □

Algorithm 4 (FIND_SECs***).*** is very similar to the single-dimensional case ([37, Alg. 2]). The difference is that in the multi-objective setting we cannot just fix the strategy of Minimizer and compute the ECs in the resulting SG. We have to pick a direction from the region and consider the strategy of Minimizer w.r.t. that direction. Since we know that the given region is from a consistent partition by assumption on the input (which is true due to Lemma 4.5), Minimizer's optimal strategy is the same for all directions in the input region. Thus the direction can be arbitrarily chosen from that region. We stress that FIND_SECs is called with the current under-approximation and returns only those state sets, which according to the current lower bound form regional SECs; these need not actually be regional SECs according to $\mathfrak{A}$. However, as sketched in the proof of Lemma 4.4, deflation is so conservative that it is sound given any EC. The required property of FIND_SECs is that it eventually finds the correct regional SECs when L converges to $\mathfrak{A}$ close enough, or formally:

Lemma 4.6 (FIND_SECs is sound)**.** *For $T \subseteq S$ and a region R from a consistent partition, it holds that $X \in$* FIND_SECs*$(T, \mathfrak{A}, R)$ if and only if X is an inclusion-maximal SEC for region R.*

Proof. Since R is from a consistent partition, we can pick any direction $\mathbf{d} \in R$ and identify Minimizer's optimal actions for the whole region R as in line 5. Let X be a MEC returned by FIND_SECs. Then within this EC, Minimizer only has optimal actions for region R and thus, it does not matter how exactly these choices are resolved – in particular, it does not make a difference if Maximizer takes over control of Minimizer's states as explained earlier. But then, from each $s \in X$, Maximizer can achieve precisely $\mathsf{BE}^{\mathfrak{A}}(X)$. Thus X is an inclusion-maximal SEC for region R. □

4.3 Effectiveness of GET_REGIONS

In this section we describe GET_REGIONS in more detail and argue why the computation is effective. As discussed in Section 3, regions in our context bear some resemblance to regions of timed automata [1]. We first recall some geometric notions from e.g. [36] that are necessary to talk about the representation of the considered objects:

A *(k)-simplex* is a k-dimensional polytope given as the convex hull of $k+1$ affinely independent vertices. Intuitively, a simplex is a point, line segment, triangle, tetrahedron etc. For example, considering Figure 7b, the point $(0.5, 0, 0.5)$ is a 0-simplex, the line between this point and $(1, 0, 0)$ is a 1-simplex, and the whole triangle is a 2-simplex. A *face* of a k-dimensional simplex is the convex hull of a non-empty subset of the $k+1$ points making up the simplex. A *facet* of a k-dimensional simplex is a natural face, i.e. a face that uses exactly k points. For example, for a 2-simplex which is a triangle, the triangle itself, the 3 edges and 3 vertices are all faces. Each face is also a simplex. Only the three edges are facets.

A *simplicial complex (SC)* is a set of simplices closed under taking faces, i.e. every face of a simplex in the SC is also part of the SC. It also satisfies the property that a non-empty intersection of any two simplices in the SC is a face of both the simplices. Using Figure 7b again: Consider the SC containing the two lines (1-simplices) between $(0.5, 0, 0.5)$ and $(1, 0, 0)$ as well as between $(0.5, 0, 0.5)$ and $(0, 0, 1)$. It also has to contain the point (0-simplex) $(0.5, 0, 0.5)$, as that is the intersection of the lines. Additionally, the points $(1, 0, 0)$ and $(0, 0, 1)$ need to be in the SC, as they are the faces of the lines. In order to represent (i) partitions (disjoint decompositions) and (ii) open regions, as discussed already in Section 3.3, we consider the open version: we subtract from each simplex all its facets and, abusing the notation, call them simplices and their union SC.

The invariant of our computation is that all partitions into regions as well as the Pareto frontiers are represented as finite unions of SCs. The partitions decompose (triangulate) D, the part of the projective hyperplane that is in the non-negative orthant (n-dimensional analog of the first quadrant), which can thus itself be seen as $(n-1)$-simplex; the Pareto frontiers are given by linear functions on the areas defined by regions, hence consists of SCs in the non-negative orthant (of course, generally not arranged in a hyperplane). Altogether, since simplices can be stored as the set of their vertices, we can effectively represent these partitions and frontiers by finite sets of finite sets of points.

For the computation of GET_REGIONS on Line 5, we can compute the intersections of all pairs of Pareto frontiers of the available actions as they are piece-wise linear with finitely many pieces, and we obtain a finite partition. The projected intersections then become k-simplices for some $k > 0$ (the intersection of Pareto frontiers can be points, lines, planes and so on as seen in Fig. 7 and 5). Similarly, on Line 7, starting from two finite partitions, their common refinement after the respective triangulation, as e.g. in Figure 6, is also finite and an SC. Recall the base case for the partition is the SC of the projective $n-1$-simplex.

Finally, the resulting approximation of $\mathfrak{A}$ is effective since, given a direction $\mathbf{d}$, we can identify its region and the respective simplex on the Pareto frontiers L and U and their value in the intersection with $\mathbf{d}$. For effectiveness of the stopping criterion on Line 9 of Algorithm 1, we additionally note that we only need to test for each simplex the differences in its generating points (more precisely the limits as the simplex is open) since the difference is a linear function on each of the finitely many pieces of the approximation.

5 Correctness Proof of Algorithm MO-BVI

Our new Bellman operator $\overline{\mathfrak{B}}$ defined as one application of the loop body of Algorithm 1 is a higher order operator transforming pairs of the estimate functions: the two estimate functions $\mathsf{L}, \mathsf{U} \in S \to 2^{[0,1]^n}$ for the under-/over-approximation are transformed into a pair with the modified under- and over-approximation. It can thus be seen as a function of type

$$\overline{\mathfrak{B}} \;:\; \left(S \to 2^{[0,1]^n} \times 2^{[0,1]^n}\right) \;\to\; \left(S \to 2^{[0,1]^n} \times 2^{[0,1]^n}\right).$$

We fix an SG $\mathsf{G} = (S, S_\Box, S_\bigcirc, \mathsf{s}_0, \mathsf{A}, \mathsf{Av}, \delta)$ and a generalized-reachability objective $\mathcal{T}$ for the following proofs and implicitly use them as parameters of $\overline{\mathfrak{B}}$. Note that for all states $\mathsf{s} \in S$, $\mathsf{U}_0(\mathsf{s}) = dwc(\{\vec{1}\})$ respectively $\mathsf{L}_0(\mathsf{s}) = \{\vec{0}\}$ are set by the initialization.

We consider the sequence $(\mathsf{L}_i, \mathsf{U}_i) := \overline{\mathfrak{B}}^i(\mathsf{L}_0, \mathsf{U}_0)$, $i \in \mathbb{N}$, output by our algorithm. We also use the notation $\mathsf{L}_\infty := \lim_{i\to\infty} \mathsf{L}_i := \bigcup_{i\geq 0} \mathsf{L}_i$ and $\mathsf{U}_\infty := \lim_{i\to\infty} \mathsf{U}_i := \bigcap_{i\geq 0} \mathsf{U}_i$.

Proposition 5.1. ***Soundness***
Algorithm 1 computes for each state $\mathsf{s} \in S$ *a sequence of monotonic over- and under-approximations of* $\mathfrak{A}(\mathsf{s})$*, i.e.* $\forall i \in \mathbb{N} : \mathsf{L}_i(\mathsf{s}) \subseteq \mathfrak{A}(\mathsf{s}) \subseteq \mathsf{U}_i(\mathsf{s})$ *and for* $i < j, \mathsf{L}_i(\mathsf{s}) \subseteq \mathsf{L}_j(\mathsf{s})$ *as well as* $\mathsf{U}_i(\mathsf{s}) \supseteq \mathsf{U}_j(\mathsf{s})$.

Proposition 5.2. ***Convergence from below***
$\forall$ *states* $\mathsf{s} \in S$ *and all directions* $\mathbf{d} \in \mathsf{D} : \mathsf{L}_\infty(\mathsf{s})[\mathbf{d}] = \mathfrak{A}(\mathsf{s})[\mathbf{d}]$.

Proposition 5.3. ***Convergence from above***
$\forall$ *states* $\mathsf{s} \in S$ *and all directions* $\mathbf{d} \in \mathsf{D} : \mathsf{U}_\infty(\mathsf{s})[\mathbf{d}] = \mathfrak{A}(\mathsf{s})[\mathbf{d}]$.

Note that for all directions $\mathbf{d}$ and for all $\mathsf{s} \in S$ by definition $\mathfrak{A}(\mathsf{s})[\mathbf{d}] = \mathfrak{P}(\mathsf{s})[\mathbf{d}]$. Using this and the three propositions, we can prove the main theorem.

Theorem 5.4. *Algorithm 1 computes convergent monotonic over- and under-approximations of* $\mathfrak{P}(\mathsf{s})$ *for each* $\mathsf{s} \in S$*. Since it is convergent, for every* $\epsilon > 0$ *there exists an* i*, such that for every* $\mathsf{s} \in S$ *and direction* $\mathbf{d} \in \mathsf{D} : \mathsf{U}_i(\mathsf{s})[\mathbf{d}] - \mathsf{L}_i(\mathsf{s})[\mathbf{d}] < \epsilon$*. So by instantiating* s *with* s_0*, we solve the problem posed in Section 2.5.*

Proof of Propositions 5.1 and 5.2. Note that for all $i \in \mathbb{N}$ it holds that $\mathsf{L}_i = \mathfrak{B}^i(\mathsf{L}_0)$, since DEFLATE_SECs does not change the under-approximation. [9, Proposition 8] proves that $\mathfrak{B}$ is order-preserving, i.e. monotonic, and that it converges to the unique least fixpoint $\mathfrak{A}$ when repeatedly applied to the bottom element of a complete partial order. The least possible lower bound assigns $\vec{0}$ to all S, since there is no smaller vector that can be assigned to a state. This is exactly the definition of L_0, which implies that for all $\mathsf{s} \in S$, the closure of $\mathsf{L}_\infty(\mathsf{s})$ equals the closure of $\mathfrak{A}(\mathsf{s})$, which implies Proposition 5.2.

For the soundness of the over-approximation we require that the additional operation, namely DEFLATE_SECs, performed by $\overline{\mathfrak{B}}$ is sound (proven in Lemma 4.4). The monotonicity of the under- and over-approximation follows from the monotonicity of $\mathfrak{B}$ [9, Proposition 8] and of DEFLATE_SECs (Lemma 4.4) Thus we can deduce Proposition 5.1. □

It only remains to show Proposition 5.3. As a key ingredient for the proof we will use the following:

Lemma 5.5 (Fixpoint)**.** $\overline{\mathfrak{B}}(\mathsf{L}_\infty, \mathsf{U}_\infty) = (\mathsf{L}_\infty, \mathsf{U}_\infty)$*, i.e. the limit of* $\overline{\mathfrak{B}}$ *is also a fixpoint.*

Proof idea. We only need to argue about the second component U_∞. If we did not have a fixpoint, then a further application of $\overline{\mathfrak{B}}$ would find a SEC T for some region R and decrease the over-approximation U_∞ for some $\mathbf{d} \in R$ and $s \in T$, i.e. $\mathsf{BE}^{\mathsf{U}_\infty}(T)[\mathbf{d}] < \mathsf{U}_\infty(s)[\mathbf{d}]$. The key idea is that since the lower approximations L_i converge to $\mathfrak{A}$, the SEC T is detected and deflated infinitely many times before convergence. But this means that $\mathsf{BE}^{\mathsf{U}_\infty}(T)[\mathbf{d}] = \mathsf{U}_\infty(s)[\mathbf{d}]$, contradiction. For more details, see [3, Appendix A.2]. □

Proof of Proposition 5.3. We will use the fixpoint property from Lemma 5.5 to derive a contradiction. We assume for contradiction that there is a state $\mathsf{s} \in S$ and a direction $\mathbf{d} \in \mathsf{D}$ such that $\mathsf{U}_\infty(\mathsf{s})[\mathbf{d}] \neq \mathfrak{A}(\mathsf{s})[\mathbf{d}]$. Applying the Bellman operator once more to $(\mathsf{L}_\infty, \mathsf{U}_\infty)$ results in a new upper bound U'. We will show that $\mathsf{U}' \subsetneq \mathsf{U}_\infty$. In other words, applying the loop once more decreases the over-approximation. This is a contradiction to U_∞ being a fixpoint and proves our goal.

1. Assume for contradiction, that $\exists t \in S, \mathbf{d} \in \mathsf{D} : \mathsf{U}_\infty(t)[\mathbf{d}] \neq \mathfrak{A}(t)[\mathbf{d}]$ and thus $\exists \mathbf{d}\; \mathsf{U}_\infty(t)[\mathbf{d}] > \mathfrak{A}(t)[\mathbf{d}]$ with Prop. 5.1. We fix this direction $\mathbf{d}$ and t for the rest of the proof.
2. Let $X := \{\mathsf{s} \in S \mid \Delta(\mathsf{s}) = \max_{t\in S} \Delta(t)\}$, where $\Delta(\mathsf{s}) := \mathsf{U}_\infty(\mathsf{s})[\mathbf{d}] - \mathfrak{A}(\mathsf{s})[\mathbf{d}]$ is the difference between over-approximation $\mathsf{U}_\infty(\mathsf{s})$ and achievable set $\mathfrak{A}$ in $\mathbf{d}$.
   a. We also define $\Delta(\mathsf{s}, \mathsf{a}) := \mathsf{U}_\infty(\mathsf{s}, \mathsf{a})[\mathbf{d}] - \mathfrak{A}(\mathsf{s}, \mathsf{a})[\mathbf{d}]$ for an action $\mathsf{a} \in \mathsf{Av}(s)$.
   b. By assumption, $X \neq \emptyset$ and for all $\mathsf{s} \in X : \Delta(\mathsf{s}) > 0$.
   Note: $\Delta(s)$, $\Delta(s, a)$ and X are all defined w.r.t. the fixed direction $\mathbf{d}$ (not indicated in notation to avoid clutter).
3. $\Delta(s) > 0$ implies that $dwc(\mathbb{1}_{\mathcal{T}}(\mathsf{s}))[\mathbf{d}] = 0$, i.e. s is not contained in target sets "aligned" in direction $\mathbf{d}$ because otherwise, $\mathsf{U}_\infty(\mathsf{s})[\mathbf{d}] = \mathfrak{A}(\mathsf{s})[\mathbf{d}] = dwc(\mathbb{1}_{\mathcal{T}}(\mathsf{s}))[\mathbf{d}]$.
4. For all (s, a) exits X it holds that $\Delta(\mathsf{s}, \mathsf{a}) < \Delta(\mathsf{s})$.
   Reason: If (s, a) exits X, then $\exists \mathsf{s}' \in \mathsf{Post}(\mathsf{s}, \mathsf{a}) \setminus X$. Note

that $\Delta(s') < \Delta(s)$ by construction of X

$$\begin{aligned}\Delta(s,a) &= \mathsf{U}_\infty(s,a)[\mathbf{d}] - \mathfrak{A}(s,a)[\mathbf{d}] \quad \text{(Definition of } \Delta(s,a)\text{)}\\ &= \sum_{s' \in \mathrm{Post}(s,a)} \delta(s,a,s') \left(\mathsf{U}_\infty(s')[\mathbf{d}] - \mathfrak{A}(s')[\mathbf{d}]\right)\\ &\qquad \text{(Definition of } \mathfrak{A}(s,a) \text{ and Step 3)}\\ &= \sum_{s' \in \mathrm{Post}(s,a)} \delta(s,a,s')\Delta(s') \qquad \text{(Definition of } \Delta(s)\text{)}\\ &< \Delta(s) \qquad \text{(since } t \in \mathrm{Post}(s,a) \text{ and } \Delta(t) < \Delta(s)\text{)}\end{aligned}$$

5. No state in X depends on a leaving action. Formally:
   (a) $\forall s \in X_\bigcirc, (s,a)\ \text{exits}\ X : \mathfrak{A}(s)[\mathbf{d}] < \mathfrak{A}(s,a)[\mathbf{d}]$, i.e. a leaving action leaving X cannot be optimal for Minimizer in direction $\mathbf{d}$.
   Reason: Since s is a state of Minimizer, $\forall a \in \mathsf{Av}(s) : \mathsf{U}_\infty(s)[\mathbf{d}] \leq \mathsf{U}_\infty(s,a)[\mathbf{d}]$. From this and the inequality from the previous Step 4, we get that:

$$\begin{aligned}&\mathsf{U}_\infty(s,a)[\mathbf{d}] - \mathfrak{A}(s,a)[\mathbf{d}]\\ &= \Delta(s,a) && \text{(Definition of } \Delta\text{)}\\ &< \Delta(s) && \text{(Step 4)}\\ &= \mathsf{U}_\infty(s)[\mathbf{d}] - \mathfrak{A}(s)[\mathbf{d}] && \text{(Definition of } \Delta\text{)}\\ &\leq \mathsf{U}_\infty(s,a)[\mathbf{d}] - \mathfrak{A}(s)[\mathbf{d}] && (s \in X_\bigcirc)\end{aligned}$$

   Subtracting $\mathsf{U}_\infty(s,a)[\mathbf{d}]$ and multiplying by (-1) yields the claim.
   (b) $\forall s \in X_\Box$, $\mathsf{U}_\infty(s)[\mathbf{d}] > \sum_{a \in \mathsf{Av}(s,a)} w_a \cdot \mathsf{U}_\infty(s,a)$ if $w_a > 0$ for some action a exiting X. Intuitively, this means that Maximizer cannot assign positive weight to any action leaving X. The proof is similar to part (a).
6. X contains an EC because if not, then $\exists s \in X : \forall a \in \mathsf{Av}(s) : (s,a)\ \text{exits}\ X$. But then s necessarily depends on a leaving action in the sense of the previous Step 5, contradiction.
7. Using that X contains an EC, we can show that X even contains a regional *simple* EC $Z \subseteq X$ w.r.t. to the region $\{\mathbf{d}\}$. Applying $\overline{\mathfrak{B}}$ once more to $(\mathfrak{A}, \mathsf{U}_\infty)$, the over-approximation decreases.

   Reason: We only give high-level intuition here, as the proof is very technical. The formal details are in [3, Appendix A.3]. We prove by a large case distinction that X contains a regional SEC Z for the region $\{\mathbf{d}\}$. Since $\mathsf{L}_\infty = \mathfrak{A}$, by Lemma 4.6 this regional SEC is found and deflated. By construction of Z, then its value is set to "depend on the outside", i.e. it assigns a positive weight on an action leaving X. Then, by Step 5, the over-approximation is reduced and we arrive at a contradiction. □

6 Conclusion

For a given $\varepsilon > 0$ and a generalized-reachability stochastic game, we compute an ε-approximation of its Pareto frontier. Our algorithm can be run as an anytime algorithm, reporting the under- and over-approximations on the frontier, due to an extended version of value iteration. We have suggested the name "bounded value iteration" as it better generalizes to higher dimensions than "interval iteration". We conjecture that this technique can be generalized to other models, such as concurrent games, and more complex objectives, such as total reward. Finally, while decidability remains open, the approximation algorithms are practically more relevant even in the single-dimensional case. Note that approximative value iteration is the default technique for analysis of MDP, although there is an exact and polynomial solution by linear programming. The reason is that the theoretical worst-case complexity of value iteration is practically not too relevant. Consequently, an efficient implementation, possibly exploring only a part of the state space using learning, as e.g. in [11, 37], may be an interesting future direction.

Acknowledgments

Pranav Ashok, Jan Křetínský and Maximilian Weininger were funded in part by TUM IGSSE Grant 10.06 (PARSEC) and the German Research Foundation (DFG) project KR 4890/2-1 "Statistical Unbounded Verification". Krishnendu Chatterjee was supported by the ERC CoG 863818 (ForM-SMArt) and Vienna Science and Technology Fund (WWTF) Project ICT15-003. Tobias Winkler was supported by the RTG 2236 UnRAVeL.

References

[1] Rajeev Alur and David L. Dill. 1994. A Theory of Timed Automata. *Theor. Comput. Sci.* 126, 2 (1994), 183–235.
[2] Pranav Ashok, Krishnendu Chatterjee, Przemyslaw Daca, Jan Kretínský, and Tobias Meggendorfer. 2017. Value Iteration for Long-Run Average Reward in Markov Decision Processes. In *CAV*. 201–221. https://doi.org/10.1007/978-3-319-63387-9_10
[3] Pranav Ashok, Krishnendu Chatterjee, Jan Kretinsky, Maximilian Weininger, and Tobias Winkler. 2020. Approximating Values of Generalized-Reachability Stochastic Games. *CoRR* abs/1908.05106 (2020).
[4] Christel Baier, Marcus Daum, Clemens Dubslaff, Joachim Klein, and Sascha Klüppelholz. 2014. Energy-Utility Quantiles. In *NASA Formal Methods*. 285–299.
[5] Christel Baier, Clemens Dubslaff, and Sascha Klüppelholz. 2014. Trade-off analysis meets probabilistic model checking. In *CSL-LICS*. 1:1–1:10.
[6] Christel Baier, Clemens Dubslaff, Sascha Klüppelholz, Marcus Daum, Joachim Klein, Steffen Märcker, and Sascha Wunderlich. 2014. Probabilistic Model Checking and Non-standard Multi-objective Reasoning. In *FASE*. 1–16.
[7] Christel Baier and Joost-Pieter Katoen. 2008. Principles of Model Checking.
[8] Nicolas Basset, Marta Z. Kwiatkowska, Ufuk Topcu, and Clemens Wiltsche. 2015. Strategy Synthesis for Stochastic Games with Multiple Long-Run Objectives. In *TACAS (Lecture Notes in Computer Science)*, Vol. 9035. Springer, 256–271.
[9] Nicolas Basset, Marta Z. Kwiatkowska, and Clemens Wiltsche. 2018. Compositional strategy synthesis for stochastic games with multiple objectives. *Inf. Comput.* 261, Part (2018), 536–587.

[10] Tomás Brázdil, Václav Brozek, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. 2014. Two Views on Multiple Mean-Payoff Objectives in Markov Decision Processes. *LMCS* 10, 1 (2014). https://doi.org/10.2168/LMCS-10(1:13)2014

[11] Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Kretínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. 2014. Verification of Markov Decision Processes Using Learning Algorithms. In *ATVA (Lecture Notes in Computer Science)*, Vol. 8837. Springer, 98–114.

[12] Tomás Brázdil, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. 2013. Trading Performance for Stability in Markov Decision Processes. In *LICS*. 331–340.

[13] Romain Brenguier and Vojtech Forejt. 2016. Decidability Results for Multi-objective Stochastic Games. In *ATVA (Lecture Notes in Computer Science)*, Vol. 9938. 227–243.

[14] Romain Brenguier and Jean-François Raskin. 2015. Pareto Curves of Multidimensional Mean-Payoff Games. In *CAV (2) (Lecture Notes in Computer Science)*, Vol. 9207. Springer, 251–267.

[15] Krishnendu Chatterjee. 2007. Markov Decision Processes with Multiple Long-Run Average Objectives. In *FSTTCS (Lecture Notes in Computer Science)*, Vol. 4855. Springer, 473–484.

[16] Krishnendu Chatterjee and Laurent Doyen. 2016. Perfect-Information Stochastic Games with Generalized Mean-Payoff Objectives. In *LICS*. ACM, 247–256.

[17] Krishnendu Chatterjee and Nathanaël Fijalkow. 2011. A reduction from parity games to simple stochastic games. In *GandALF*. 74–86. https://doi.org/10.4204/EPTCS.54.6

[18] Krishnendu Chatterjee, Vojtech Forejt, and Dominik Wojtczak. 2013. Multi-objective Discounted Reward Verification in Graphs and MDPs. In *LPAR*. 228–242.

[19] Krishnendu Chatterjee and Thomas A Henzinger. 2008. Value iteration. In *25 Years of Model Checking*. Springer, 107–138.

[20] Krishnendu Chatterjee, Thomas A. Henzinger, Barbara Jobstmann, and Arjun Radhakrishna. 2010. Gist: A Solver for Probabilistic Games. In *CAV*. 665–669. https://doi.org/10.1007/978-3-642-14295-6_57

[21] Krishnendu Chatterjee, Zuzana Kretínská, and Jan Kretínský. 2017. Unifying Two Views on Multiple Mean-Payoff Objectives in Markov Decision Processes. *Logical Methods in Computer Science* 13, 2 (2017).

[22] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, David Parker, and Aistis Simaitis. 2013. PRISM-games: A Model Checker for Stochastic Multi-Player Games. In *TACAS (Lecture Notes in Computer Science)*, Vol. 7795. Springer, 185–191.

[23] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. 2013. On Stochastic Games with Multiple Objectives. In *MFCS (Lecture Notes in Computer Science)*, Vol. 8087. Springer, 266–277.

[24] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. 2013. *On Stochastic Games with Multiple Objectives*. Technical Report. 266–277 pages.

[25] Taolue Chen, Marta Z. Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. 2013. Synthesis for Multi-objective Stochastic Games: An Application to Autonomous Urban Driving. In *QEST*. 322–337. https://doi.org/10.1007/978-3-642-40196-1_28

[26] Chih-Hong Cheng, Alois Knoll, Michael Luttenberger, and Christian Buckl. 2011. GAVS+: An Open Platform for the Research of Algorithmic Game Solving. In *ETAPS*. 258–261. https://doi.org/10.1007/978-3-642-19835-9_22

[27] Anne Condon. 1992. The complexity of stochastic games. *Information and Computation* 96, 2 (1992), 203–224.

[28] Anne Condon. 1993. On Algorithms for Simple Stochastic Games. In *Advances in Computational Complexity Theory, volume 13 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 51–73.

[29] Costas Courcoubetis and Mihalis Yannakakis. 1995. The Complexity of Probabilistic Verification. *J. ACM* 42, 4 (July 1995), 857–907.

[30] Kousha Etessami, Marta Z. Kwiatkowska, Moshe Y. Vardi, and Mihalis Yannakakis. 2008. Multi-Objective Model Checking of Markov Decision Processes. *Logical Methods in Computer Science* 4, 4 (2008).

[31] J.A. Filar, D. Krass, and K.W Ross. 1995. Percentile performance criteria for limiting average Markov decision processes. *Automatic Control, IEEE Transactions on* 40, 1 (Jan 1995), 2–10.

[32] Vojtech Forejt, Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. 2011. Quantitative Multi-objective Verification for Probabilistic Systems. In *TACAS*. 112–127. https://doi.org/10.1007/978-3-642-19835-9_11

[33] Vojtech Forejt, Marta Z. Kwiatkowska, and David Parker. 2012. Pareto Curves for Probabilistic Model Checking. In *ATVA (Lecture Notes in Computer Science)*, Vol. 7561. Springer, 317–332.

[34] Christoph Haase, Stefan Kiefer, and Markus Lohrey. 2017. Computing quantiles in Markov chains with multi-dimensional costs. In *LICS*. 1–12.

[35] Serge Haddad and Benjamin Monmege. 2018. Interval iteration algorithm for MDPs and IMDPs. *Theor. Comput. Sci.* 735 (2018), 111–131.

[36] A. Hatcher. 2002. *Algebraic Topology*. Cambridge University Press. https://books.google.de/books?id=BjKs86kosqgC

[37] Edon Kelmendi, Julia Krämer, Jan Kretínský, and Maximilian Weininger. 2018. Value Iteration for Simple Stochastic Games: Stopping Criterion and Learning Algorithm. In *CAV*. https://doi.org/10.1007/978-3-319-96145-3_36

[38] Marta Kwiatkowska, David Parker, and Clemens Wiltsche. 2016. PRISM-Games 2.0: A Tool for Multi-objective Strategy Synthesis for Stochastic Games. In *TACAS (Lecture Notes in Computer Science)*, Vol. 9636. Springer, 560–566.

[39] Marta Kwiatkowska, David Parker, and Clemens Wiltsche. 2018. PRISM-games: verification and strategy synthesis for stochastic multi-player games with multiple objectives. *STTT* 20, 2 (2018), 195–210.

[40] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *CAV (Lecture Notes in Computer Science)*, Vol. 6806. Springer, 585–591.

[41] H. Brendan Mcmahan, Maxim Likhachev, and Geoffrey J. Gordon. 2005. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML 05*. 569–576.

[42] Christos H. Papadimitriou and Mihalis Yannakakis. 2000. On the Approximability of Trade-offs and Optimal Access of Web Sources. In *FOCS*. IEEE Computer Society, 86–92.

[43] Martin L. Puterman. 2014. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons.

[44] Mickael Randour, Jean-François Raskin, and Ocan Sankur. 2015. Percentile Queries in Multi-dimensional Markov Decision Processes. In *CAV (1) (Lecture Notes in Computer Science)*, Vol. 9206. Springer, 123–139.

[45] Mickael Randour, Jean-François Raskin, and Ocan Sankur. 2017. Percentile queries in multi-dimensional Markov decision processes. *Formal Methods in System Design* 50, 2-3 (2017), 207–248. https://doi.org/10.1007/s10703-016-0262-7

[46] María Svorenová and Marta Kwiatkowska. 2016. Quantitative verification and strategy synthesis for stochastic games. *Eur. J. Control* 30 (2016), 15–30. https://doi.org/10.1016/j.ejcon.2016.04.009

[47] Yaron Velner. 2015. Robust Multidimensional Mean-Payoff Games are Undecidable. In *FoSSaCS*. Springer, 312–327.