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

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

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

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

1 Introduction

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

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

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

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

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

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

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

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

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

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

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

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

applied, and within MECs VI is applied on-demand (i.e. only when there is a requirement for more precise value bounds). The resulting algorithm furthermore is an *anytime* algorithm, i.e. it can be stopped at any time and give an upper and lower bounds on the result.
- *Experimental results.* We compare our new algorithms to the state-of-the-art tool MultiGain [BCFK15] on various models. The experiments show that MultiGain is vastly outperformed by our methods on nearly every model. Furthermore, we compare several variants of our methods and investigate the different domains of applicability.

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

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

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

2 Preliminaries

2.1 Markov Decision Processes

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

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

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

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

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

End Components. A pair (T, A), where $\emptyset \neq T \subseteq S$ and $\emptyset \neq A \subseteq \bigcup_{s \in T} \mathsf{Av}(s)$, is an *end component* of an MDP $\mathcal{M}$ if (i) for all $s \in T, a \in A \cap \mathsf{Av}(s)$ we have $\mathrm{supp}(\Delta(s,a)) \subseteq T$, and (ii) for all $s, s' \in T$ there is a finite path $w = sa_0 \dots a_n s' \in (T \times A)^* \times T$, i.e. w starts in s, ends in s', stays inside T and only uses actions in A.[1] Intuitively, an end component describes a set of states for which a particular strategy exists such that all possible paths remain inside these states and all of those states are visited infinitely often almost surely. An end component (T, A) is a *maximal end component (MEC)* if there is no other end component (T', A') such that $T \subseteq T'$ and $A \subseteq A'$. Given an MDP $\mathcal{M}$, the set of its MECs is denoted by $\mathsf{MEC}(\mathcal{M})$. With these definitions, every state of an MDP belongs to at most one MEC and each MDP has at least one MEC.

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

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

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

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

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

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

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

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

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

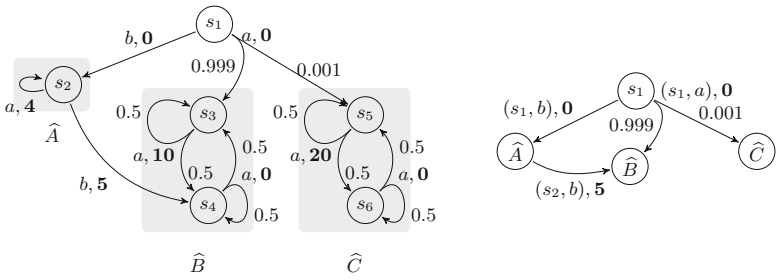
– *Finally, for* $\widehat{s} \in \widehat{S}$, $(s,a) \in \widehat{\mathsf{Av}}(\widehat{s})$, *we define* $\widehat{r}(s,(s,a)) = r(s,a)$.

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Algorithm 1. ValueIteration

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

3 Value Iteration Solutions

3.1 Naive Value Iteration

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

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

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

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

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

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

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

span semi-norm, which is defined as the difference between the maximum and minimum element of a vector $\boldsymbol{w}$

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

The stopping criterion then is given by the condition

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

When the criterion (SC1) is satisfied we have that

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

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

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

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

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

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

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

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

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

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

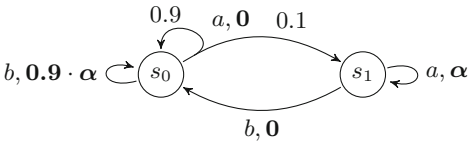


Fig. 2. A communicating MDP parametrized by the value α.

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

3.2 Local Value Iteration

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

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

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

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

intuition, we show in Theorem 2 that the problem of computing the average reward is reduced to computing the value of each MEC and determining the maximum probability of reaching the state s_+ in the weighted MEC quotient.

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

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

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

- Δ^f *is defined as*

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

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

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

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

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

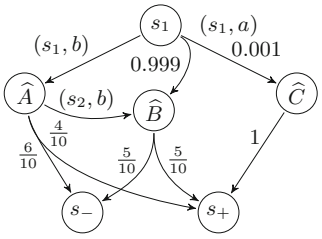


Fig. 3. The weighted quotient of the MDP in Fig. 1a and function $f = \{\widehat{A} \mapsto \frac{4}{10}, \widehat{B} \mapsto \frac{5}{10}, \widehat{C} \mapsto \frac{10}{10}\}$. Rewards and stay action labels omitted for readability.

Algorithm 2. LocalVI

Input: MDP $\mathcal{M} = (S, s_{\text{init}}, Act, \mathsf{Av}, \Delta, r)$, precision $\varepsilon > 0$
Output: w, s.t. $|w - v(s_{\text{init}})| < \varepsilon$

$f = \emptyset$
for $M_i = (T_i, A_i) \in \mathsf{MEC}(\mathcal{M})$ **do** ▷ Determine values for MECs
  Compute the average reward $w(M_i)$ on M, such that $|w(M_i) - v(M_i)| < \frac{1}{2}\varepsilon$,
  $f(\hat{s}_i) \leftarrow \frac{1}{r_{\max}} w(M_i)$
$\mathcal{M}^f \leftarrow$ the weighted MEC quotient of $\mathcal{M}$ and f
Compute p s.t. $|p - \sup_{\pi \in \Pi} \mathbb{P}^{\pi}_{\mathcal{M}^f, s^f_{\text{init}}}(\Diamond s_+)| < \frac{1}{2r_{\max}}\varepsilon$ ▷ Determine reachability
return $r_{\max} \cdot p$

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

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

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

3.3 On-Demand Value Iteration

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

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

Further, suppose that $\widehat{A}$ was a more complicated MEC, but after a few iterations the criterion (SC1) already shows that the value of $\widehat{A}$ is at most 4.4. Similarly, after several iterations in $\widehat{B}$, we might see that the value of $\widehat{B}$ is

greater than 4.5. In this situation, there is no point in further approximating the value of $\widehat{A}$ since the action b leading to it will not be optimal anyway, and its precise value will not be reflected in the result. △

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

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

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

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

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

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

Algorithm 3. OnDemandVI

Input: MDP $\mathcal{M} = (S, s_{\text{init}}, Act, \mathsf{Av}, \Delta, r)$, precision $\varepsilon > 0$, threshold $k \geq 2$
Output: w, s.t. $|w - v(s_{\text{init}})| < \varepsilon$

1: Set $u(\cdot,\cdot) \leftarrow 1$, $u(s_-,\cdot) \leftarrow 0$; $l(\cdot,\cdot) \leftarrow 0$, $l(s_+,\cdot) \leftarrow 1$ ▷ Initialize
2: Let $A(s) := \arg\max_{a \in \mathsf{Av}^{l,u}(s)} u(s,a)$
3: Let $u(s) := \max_{a \in A(s)} u(s,a)$ and $l(s) := \max_{a \in A(s)} l(s,a)$
4: **repeat**
5: $s \leftarrow s_{\text{init}}^{l,u}, w \leftarrow s$ ▷ Generate path
6: **repeat**
7: $a \leftarrow$ sampled uniformly from $A(s)$
8: $s \leftarrow$ sampled according to $\Delta^{l,u}(s,a)$
9: $w \leftarrow w, a, s$
10: **until** $s \in \{s_+, s_-, s_?\}$ or $\mathsf{Appear}(s,w) = k$ ▷ Terminate path
11: **if** $\text{pop}(w) = s_?$ **then** ▷ Refine MEC in which stay was taken
12: $\text{pop}(w)$
13: $\widehat{q} \leftarrow \text{top}(w)$
14: Run VI on $\widehat{q}$, updating u and l, until $u - l$ is halved
15: Update $\Delta^{l,u}(\widehat{q}, \mathsf{stay})$ according to Definition 5
16: **else if** $\mathsf{Appear}(s,w) = k$ **then** ▷ Update EC-collapsing
17: OnTheFlyEc
18: **repeat** ▷ Back-propagate values
19: $a \leftarrow \text{pop}(w)$, $s \leftarrow \text{pop}(w)$
20: $u(s,a) \leftarrow \sum_{s' \in S} \Delta(s,a,s') \cdot u(s')$
21: $l(s,a) \leftarrow \sum_{s' \in S} \Delta(s,a,s') \cdot l(s')$
22: **until** $w = \emptyset$
23: **until** $u(s_{\text{init}}) - l(s_{\text{init}}) < \frac{2\varepsilon}{r_{\max}}$ ▷ Terminate
24: **return** $r_{\max} \cdot \frac{1}{2}(u(s_{\text{init}}) + l(s_{\text{init}}))$

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

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

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

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

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

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

Procedure 4. OnTheFlyEc

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

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

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

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

4 Implementation and Experimental Results

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

4.1 Models

First, we briefly explain the examples used for evaluation. **virus** [KNPV09] models a virus spreading through a network. We reward each attack carried out by an infected machine. Note that in this model, no machine can "purge" the

virus, hence eventually all machines will be infected. **cs_nfail** [KPC12] models a client-server mutual exclusion protocol with probabilistic failures of the clients. A reward is given for each successfully handled connection. **investor** [MM07,MM02] models an investor operating in a stock market. The investor can decide to sell his stocks and keep their value as a reward or hold them and wait to see how the market evolves. The rewards correspond to the value of the stocks when the investor decides to sell them, so maximizing the average reward corresponds to maximizing the expected selling value of the stocks. **phil_nofair** [DFP04] represents the (randomised) dining philosophers without fairness assumptions. We use two reward structures, one where a reward is granted each time a philosopher "thinks" or "eats", respectively. **rabin** [Rab82] is a well-known mutual exclusion protocol, where multiple processes repeatedly try to access a shared critical section. Each time a process successfully enters the critical section, a reward is given. **zeroconf** [KNPS06] is a network protocol designed to assign IP addresses to clients without the need of a central server while still avoiding address conflicts. We explain the reward assignment in the corresponding result section. **sensor** [KPC12] models a network of sensors sending values to a central processor over a lossy connection. A reward is granted for every *work* transition.

4.2 Tools

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

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

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

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

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

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

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

4.3 Results

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

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

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

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

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

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

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

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

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

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

These results show that for some models, `ODV` is able to obtain an ε-optimal estimate of the mean payoff while only exploring a tiny fraction of the state

space. This allows us to solve many problems which previously were intractable simply due to an enormous state space.

5 Conclusion

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

References

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

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

[BCC+14] Brázdil, T., Chatterjee, K., Chmelík, M., Forejt, V., Křetínský, J., Kwiatkowska, M., Parker, D., Ujma, M.: Verification of Markov decision processes using learning algorithms. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 98–114. Springer, Cham (2014). doi:10.1007/978-3-319-11936-6_8

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[KNP11] Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22110-1_47

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

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

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

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

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

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

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

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

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

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

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

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