Long-Run Rewards for Markov Automata

Yuliya Butkova[1(✉)], Ralf Wimmer[2], and Holger Hermanns[1]

[1] Saarland University, Saarbrücken, Germany
{butkova,hermanns}@depend.uni-saarland.de
[2] Albert-Ludwigs-Universität Freiburg, Freiburg im Breisgau, Germany
wimmer@informatik.uni-freiburg.de

Abstract. Markov automata are a powerful formalism for modelling systems which exhibit nondeterminism, probabilistic choices and continuous stochastic timing. We consider the computation of long-run average rewards, the most classical problem in continuous-time Markov model analysis. We propose an algorithm based on value iteration. It improves the state of the art by orders of magnitude. The contribution is rooted in a fresh look on Markov automata, namely by treating them as an efficient encoding of CTMDPs with – in the worst case – exponentially more transitions.

1 Introduction

The need for automated verification is becoming more and more pertinent with the complexity of systems growing day by day. Estimating the expected cost of system maintenance, maximising the expected profit, evaluating the availability of the system in the long run – all these questions can be answered by quantitative model checking.

Quantitative model checking of models such as continuous-time Markov chains (CTMCs) and continuous-time Markov decision processes (CTMDPs) has been studied extensively. Unfortunately, modelling complex systems requires a formalism that admits compositionality, which neither CTMCs nor CTMDPs can offer. The most general compositional formalism available to date are Markov automata [5]. Markov automata can model controllable (via nondeterministic choices) systems running in continuous time that are prone to random phenomena.

Enriching Markov automata with rewards enables the assessment of system performance, dependability and more generally quality of service (QoS) [10]. *State rewards* represent costs that are accumulated over time, for instance, related to energy consumption. Costs associated with executing a certain step or policy, e.g. a deliberate violation of QoS, are modelled by means of *action rewards*.

This work is partly supported by the ERC Advanced Grant 695614 (POWVER), by the German Research Council (DFG) as part of the Cluster of Excellence BrainLinks/BrainTools (EXC 1086) and by the Sino-German Center for Research Promotion as part of the project CAP (GZ 1023).

A. Legay and T. Margaria (Eds.): TACAS 2017, Part II, LNCS 10206, pp. 188–203, 2017.
DOI: 10.1007/978-3-662-54580-5_11

The long-run behaviour of a model is by far the most prominent and most often studied property in the general context of continuous-time Markov models [13,14]. We discuss the corresponding problem for Markov automata with rewards, namely the computation of long-run average reward properties. Thus far, this problem is solved by reducing it to linear programming (LP) [10]. LP solvers, despite the abundance of options as well as numerous techniques improving their efficiency, tend to scale poorly with the size of the model.

In this paper we develop the Bellman equation [1] for long-run average reward properties. This characterisation enables the use of value or policy iteration approaches, which on other Markov models are known to scale considerably better than algorithms based on linear programming. This characterisation is made possible by considering a Markov automaton as a compact representation of a CTMDP with – in the worst case – exponentially more transitions. To arrive there, we do not consider probabilistic states as first-class objects, but rather as auxiliary states that encode the CTMDP's transitions compactly. From this new perspective, the analysis of Markov automata does not require designing new techniques, but lets us adopt those used for CTMDPs. However, a trivial adaptation of CTMDP algorithms to an exponentially larger model obtained from a Markov automaton would obviously induce exponential runtime. We manage to avoid this issue by a dedicated treatment of exponentiality via dynamic programming. As a result, considering the problem from a different angle enables us to design a simple, yet very efficient algorithm. Its building blocks are algorithms that have been known for a long time – relative value iteration for CTMDPs and dynamic programming for classical finite horizon problems.

The original LP-based algorithm is available in the IMCA tool [10]. We have implemented our algorithm in IMCA as well and evaluated both approaches on a number of benchmarks. The runtime of our algorithm for long-run average reward is several orders of magnitude better than the LP-based approach. The latter can outperform our algorithm on small models, but it scales far worse, which makes our algorithm the clearly preferred solution for real-world models.

2 Foundations

Given a finite or countable set S, a *probability distribution* over S is a function $\mu : S \to [0,1]$ such that $\sum_{s \in S} \mu(s) = 1$. We denote the set of all probability distributions over S by $\mathrm{Dist}(S)$. We set $\mu(S') := \sum_{s \in S'} \mu(s)$ for $S' \subseteq S$.

Definition 1. *A* (closed) Markov reward automaton (MRA) $\mathcal{M}$ *is a tuple* $\mathcal{M} = (S, s_0, Act, \hookrightarrow, \rightsquigarrow, \mathrm{r}, \rho)$ *such that*

- S *is a finite set of states;*
- $s_0 \in S$ *is the initial state;*
- Act *is a finite set of actions;*
- $\hookrightarrow \subseteq S \times Act \times \mathrm{Dist}(S)$ *is a finite* probabilistic transition relation*;*
- $\rightsquigarrow \subseteq S \times \mathbb{R}^{>0} \times S$ *is a finite* Markovian transition relation*;*
- $\mathrm{r} : S \times Act \to \mathbb{R}_{\geqslant 0}$ *is a* transient reward function*;*
- $\rho : S \to \mathbb{R}_{\geqslant 0}$ *is a* state reward function.

We often abbreviate $(s, \alpha, \mu) \in \hookrightarrow$ by $s \overset{\alpha}{\hookrightarrow} \mu$ and write $s \overset{\lambda}{\rightsquigarrow} s'$ instead of $(s, \lambda, s') \in \rightsquigarrow$. $Act(s) = \{\alpha \in Act \mid \exists \mu \in \mathrm{Dist}(S) : s \overset{\alpha}{\hookrightarrow} \mu\}$ denotes the set of actions that are enabled in state $s \in S$. A state s is *probabilistic (Markovian)*, if it has at least one probabilistic transition $s \overset{\alpha}{\hookrightarrow} \mu$ (Markovian transition $s \overset{\lambda}{\rightsquigarrow} s'$, resp.). States can be both probabilistic and Markovian. We denote the set of probabilistic states by $PS_{\mathcal{M}}$ and the Markovian states by $MS_{\mathcal{M}}$. To simplify notation, we assume w. l. o. g. that actions of probabilistic transitions of a state are pairwise different (this can be achieved by renaming them).

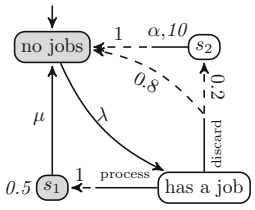


Fig. 1. An example MRA.

Example 1. Figure 1 shows an example MRA of a lazy server. Grey and white coloring of states indicate the sets $MS_{\mathcal{M}}$, respectively $PS_{\mathcal{M}}$ (their intersection being disjoint here). Transitions labelled as *discard*, *process* or α are actions enabled in a state. Dashed transitions associated with an action represent the distribution assigned to the action. Purely solid transitions are Markovian. The server has to process jobs, which arrive at rate λ; this is modelled by a Markovian transition with a corresponding rate. Whenever there is a job to process, the server chooses either to *process* or to *discard* it. These decisions are modelled by probabilistic transitions with corresponding actions. A job is processed by the server with rate μ and requires energy. We model energy consumption as a state reward 0.5 for state s_1. Discarding a job doesn't cost any energy, but with a 20% chance leads to a complaint and associated costs. These costs are modelled as an action reward 10 of state s_2 and action α.

For a Markovian state $s \in MS_{\mathcal{M}}$, the value $R(s, s') := \sum_{(s,\lambda,s') \in \rightsquigarrow} \lambda$ is called the *transition rate* from s to s'. The *exit rate* of a Markovian state s is $E(s) := \sum_{s' \in S} R(s, s')$. We require $E(s) < \infty$ for all $s \in MS_{\mathcal{M}}$.

For a probabilistic state s, s.t. $s \overset{\alpha}{\hookrightarrow} \mu$ for some α, the value $\mathbb{P}[s, \alpha, s'] := \mu(s')$. For a Markovian state s with $E(s) > 0$, the branching probability distribution when leaving the state through a Markovian transition is denoted by $\mathbb{P}[s, \cdot] \in \mathrm{Dist}(S)$ and defined by $\mathbb{P}[s, s'] := R(s, s')/E(s)$.

The Markovian transitions are governed by an exponential distributions, i. e. the probability of leaving $s \in MS_{\mathcal{M}}$ within $t \geq 0$ time units is given by $1 - \mathrm{e}^{-E(s) \cdot t}$, after which the next state is chosen according to $\mathbb{P}[s, \cdot]$.

In this paper we consider *closed* MRA, i. e. probabilistic transitions cannot be delayed by further compositions. Therefore we can make the usual *urgency assumption* that probabilistic transitions happen instantaneously. Whenever the system is in state s with $Act(s) \neq \emptyset$ and an action $\alpha \in Act(s)$ is chosen, the successor s' is selected according to the distribution $\mathbb{P}[s, \alpha, \cdot]$ and the system moves instantaneously from s to s'. The residence time in probabilistic states is therefore always 0. As the execution of a probabilistic transition is instantaneous and because the probability that a Markovian transition is triggered immediately is 0, we can assume that the probabilistic transitions take precedence over the Markovian transitions. We therefore assume $PS_{\mathcal{M}} \cap MS_{\mathcal{M}} = \emptyset$.

Additionally, we make the following non-Zenoness assumption, as in [9]. An MRA is *non-Zeno* iff no *maximal end component* [9] of only probabilistic states is reachable with probability > 0. This excludes models in which there is a chance to get trapped in an infinite number of transitions occurring in finite time.

Paths, Rewards and Schedulers. A *(timed) path* in $\mathcal{M}$ is a finite or infinite sequence $\pi = s_0 \xrightarrow{\alpha_0,t_0} s_1 \xrightarrow{\alpha_1,t_1} \cdots \xrightarrow{\alpha_k,t_k} s_{k+1} \xrightarrow{\alpha_{k+1},t_{k+1}} \cdots$. Here $s_i \xrightarrow{\alpha_i,0} s_{i+1}$ s.t. $\alpha_i \in Act(s_i)$ is a probabilistic transition via action α_i, and $s_i \xrightarrow{\perp,t_i} s_{i+1}$, s.t. $t_i > 0$ and $s_i \overset{\lambda}{\rightsquigarrow} s_{i+1}$, denotes a Markovian transition with sojourn time t_i in state s_i. The set of all finite (infinite) paths of $\mathcal{M}$ is denoted by $Paths^*_{\mathcal{M}}$ ($Paths_{\mathcal{M}}$). An *untimed* path $\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_k} s_{k+1} \xrightarrow{\alpha_{k+1}} \cdots$ is a path containing no timing information. We use $\text{prefix}(\pi, t)$ to denote the prefix of path π until time t, i. e. $\text{prefix}(\pi, t) = s_0 \xrightarrow{\alpha_0,t_0} s_1 \xrightarrow{\alpha_1,t_1} \cdots \xrightarrow{\alpha_k,t_k} s_{k+1}$, s.t. $\sum_{i=0}^{k} t_i \leqslant t$ and $\sum_{i=0}^{k+1} t_i > t$. If $\pi = s_0 \xrightarrow{\alpha_0,t_0} s_1 \xrightarrow{\alpha_1,t_1} \cdots \xrightarrow{\alpha_{k-1},t_{k-1}} s_k$ is finite, we define $|\pi| := k$ and $\pi\downarrow := s_k$.

Let π be a finite path, we define the *accumulated reward* of π as follows:

$$\text{rew}(\pi) := \sum_{i=0}^{|\pi|-1} \rho(s_i) \cdot t_i + \text{r}(s_i, \alpha_i).$$

For an infinite path π, $\text{rew}(\pi, t) := \text{rew}\big(\text{prefix}(\pi, t)\big)$ denotes the reward collected until time t. The following two assumptions can be made without restricting reward expressiveness: (i) the state reward of probabilistic states is always 0 (since residence time in probabilistic states is 0); (ii) if $s \in MS_{\mathcal{M}}$ then $\text{r}(s, \cdot) = 0$ (due to the absence of outgoing probabilistic transitions in Markovian states).

In order to resolve the nondeterminism in probabilistic states of an MRA we need the notion of a scheduler. A *scheduler* (or *policy*) $D : Paths^*_{\mathcal{M}} \to \text{Dist}(\hookrightarrow)$ is a measurable function, s.t. $D(\pi)$ assigns positive probability only to transitions $(\pi\downarrow, \alpha, \mu) \in \hookrightarrow$, for some α, μ. The set of all measurable schedulers is denoted by $GM_{\mathcal{M}}$. A *(deterministic) stationary scheduler* is a function $D : PS_{\mathcal{M}} \to \hookrightarrow$, s.t. $D(s)$ chooses only from transitions $(s, \alpha, \mu) \in \hookrightarrow$, for some α, μ.

An initial state s_0 and a fixed scheduler D induce a stochastic process on $\mathcal{M}$. For a stationary scheduler this process is a continuous-time Markov chain (CTMC). A CTMC is called a *unichain* (*multichain*) if it has only 1 (>1) recurrence class [3] plus possibly some transient states. We say that an MRA $\mathcal{M}$ is a *unichain* if all stationary schedulers induce a unichain CTMC on $\mathcal{M}$, and a *multichain* otherwise.

3 Long-Run Average Reward Property

In this section, we introduce the long-run average reward property on Markov reward automata and discuss the only available algorithm for this problem.

Let $\mathcal{M} = (S, s_0, Act, \hookrightarrow, \rightsquigarrow, \text{r}, \rho)$ be a Markov reward automaton and π an infinite path in $\mathcal{M}$. The random variable $\mathcal{L}_{\mathcal{M}} : Paths_{\mathcal{M}} \to \mathbb{R}_{\geqslant 0}$ such that

$$\mathcal{L}_{\mathcal{M}}(\pi) := \lim_{t \to \infty} \frac{1}{t} \text{rew}(\pi, t)$$

denotes the long-run average reward over a path π in $\mathcal{M}$. We now define the *optimal expected long-run average reward* on $\mathcal{M}$ with initial state s as follows:

$$\mathrm{aR}^{\mathrm{opt}}_{\mathcal{M}}(s) := \operatorname*{opt}_{D \in GM_{\mathcal{M}}} \mathbf{E}_{s,D}[\mathcal{L}_{\mathcal{M}}] = \operatorname*{opt}_{D \in GM_{\mathcal{M}}} \int_{Paths_{\mathcal{M}}} \mathcal{L}_{\mathcal{M}}(\pi) \mathrm{Pr}_{s,D}[\mathrm{d}\pi],$$

where $\mathrm{opt} \in \{\sup, \inf\}$. In the following, we use $\mathrm{aR}^{\mathrm{opt}}_{\mathcal{M}}$ instead of $\mathrm{aR}^{\mathrm{opt}}_{\mathcal{M}}(s)$, whenever the value does not depend on the initial state. Furthermore, $\mathrm{aR}^{D}_{\mathcal{M}}(s)$ denotes the long-run average reward gathered when following the policy D.

Guck et al. [10] show that under the assumptions mentioned in Sect. 2 there is always an optimal scheduler for the $\mathrm{aR}^{\mathrm{opt}}$ problem that is stationary. From now on we therefore consider only stationary schedulers.

Quantification. We will present now the only available solution for the quantification of $\mathrm{aR}^{\mathrm{opt}}$ [10]. The computation is split into three steps:

1. *Find all maximal end components of* $\mathcal{M}$. A *maximal end component (MEC)* of a MRA can be seen as a maximal sub-MRA whose underlying graph is strongly connected. An MRA may have multiple MECs. The problem of finding all MECs of an MRA is equivalent to decomposing a graph into strongly connected components. This problem admits efficient solutions [4].
2. *Compute* $\mathrm{aR}^{\mathrm{opt}}_{\mathcal{M}}$ *for each maximal end component.* An optimal scheduler for $\mathrm{aR}^{\mathrm{opt}}$ on an MEC induces a unichain on this MEC [10]. A solution for unichain MRA is therefore needed for this step. The solution provided by Guck et al. [10] is based on a reduction of the $\mathrm{aR}^{\mathrm{opt}}$ computation to the solution of a linear optimisation problem. The latter in turn can be solved by any of the available linear programming solvers.
3. *Compute a stochastic shortest path (SSP) problem.* Having the optimal values $\mathrm{aR}^{\mathrm{opt}}_{\mathcal{M}_j}$ for maximal end components $\mathcal{M}_j$, the following holds [9,10]:

$$\mathrm{aR}^{\mathrm{opt}}_{\mathcal{M}}(s) = \sup_{D \in GM} \sum_{j=1}^{k} \mathrm{Pr}_{s,D}[\Diamond\Box S_j] \cdot \mathrm{aR}^{\mathrm{opt}}_{\mathcal{M}_j},$$

   where $\mathrm{Pr}_{s,D}[\Diamond\Box S_j]$ denotes the probability to eventually reach and then stay in the MEC $\mathcal{M}_j$ starting from state s and using the scheduler D. S_j is the state space of $\mathcal{M}_j$. The authors reduce this problem to a well-established SSP problem on Markov decision processes [13], that admits efficient solutions, such as value or policy iteration [2].

One can see that steps 1 and 3 of this algorithm admit efficient solutions, while the algorithm for step 2 is based on linear programming. The algorithms for linear programming are, unfortunately, known to not scale well with the size of the problem in the context of Markov decision processes, relative to iterative algorithms based on value or policy iteration. So far, however, no iterative algorithm is known for long-run average rewards on Markov automata. In this work we fill this gap and design an iterative algorithm for the computation of long-run average rewards on MRA.

4 An Iterative Approach to Long-Run Average Rewards

In this section, we present our approach for quantifying the long-run average reward on Markov reward automata. Recall that the original algorithm, described in the previous section, is efficient in all the steps except for step 2 – the computation of the long-run average reward for unichain MRA. We therefore target this specific sub-problem and present our algorithm for unichain MRA. Having an arbitrary MRA $\mathcal{M}$, one can quantify $\mathrm{aR}^{\mathrm{opt}}$ by applying steps 1 and 3 of the original algorithm and using our solution for unichain MRA for step 2.

Effective Analysis of Unichain MRA. The core of our approach lies in the following observation: a Markov reward automaton can be considered as a compact representation of a possibly exponentially larger continuous-time Markov decision process (CTMDP). This observation enables us to use efficient algorithms available for CTMDPs [13] to compute long-run average rewards. But since that CTMDP, in the worst case, has exponentially more transitions, this naïve approach does not seem promising. We circumvent this problem by means of classical dynamic programming, and thereby arrive at an efficient solution that avoids the construction of the large CTMDP.

For the rest of this section, $\mathcal{M} = (S, s_0, Act, \hookrightarrow, \rightsquigarrow, \mathrm{r}, \rho)$ denotes a unichain Markov reward automaton. Guck et al. [10] show that $\mathrm{aR}^{\mathrm{opt}}$ for a unichain MRA does not depend on the initial state, i.e. $\forall s, s' : \mathrm{aR}^{\mathrm{opt}}_{\mathcal{M}}(s) = \mathrm{aR}^{\mathrm{opt}}_{\mathcal{M}}(s')$. We will therefore refer to this value as $\mathrm{aR}^{\mathrm{opt}}_{\mathcal{M}}$.

4.1 CTMDP Preserving $\mathrm{aR}^{\mathrm{opt}}$

We will now present a transformation from a unichain MRA to a CTMDP that preserves the long-run average reward property.

Definition 2. *A* continuous-time Markov decision process *(CTMDP) is a tuple* $\mathcal{C} = (S, Act, \mathrm{R})$*, where* S *is a finite set of* states*,* Act *is a finite set of* actions*, and* $\mathrm{R} : S \times Act \times S \to \mathbb{R}_{\geq 0}$ *is a rate function.*

The set $Act(s) = \{\alpha \in Act \mid \exists s' \in S : \mathrm{R}(s, \alpha, s') > 0\}$ is the set of *enabled actions* in state s. A path in a CTMDP is a finite or infinite sequence $\pi = s_0 \xrightarrow{\alpha_0, t_0} s_1 \xrightarrow{\alpha_1, t_1} \cdots \xrightarrow{\alpha_{k-1}, t_{k-1}} s_k \cdots$, where $\alpha_i \in Act(s_i)$ and t_i denotes the residence time of the system in state s_i. $E(s, \alpha) := \sum_{s' \in S} \mathrm{R}(s, \alpha, s')$ and $\mathbb{P}_{\mathcal{C}}[s, \alpha, s'] := \frac{\mathrm{R}(s,\alpha,s')}{E(s,\alpha)}$. The notions of $Paths^*_{\mathcal{C}}$, $Paths_{\mathcal{C}}$, $\mathrm{prefix}(\pi, t)$, $|\pi|$, $\pi\!\downarrow$, schedulers and unichain CTMDP are defined analogously to corresponding definitions for an MRA (see Sect. 2).

A *reward structure* on a CTMDP $\mathcal{C}$ is a tuple $(\rho_{\mathcal{C}}, \mathrm{r}_{\mathcal{C}})$, where $\rho_{\mathcal{C}} : S \to \mathbb{R}_{\geqslant 0}$ and $\mathrm{r}_{\mathcal{C}} : S \times Act \to \mathbb{R}_{\geqslant 0}$. The reward of a finite path π is defined as follows:

$$\mathrm{rew}_{\mathcal{C}}(\pi) := \sum_{i=0}^{|\pi|-1} \rho_{\mathcal{C}}(s_i) \cdot t_i + \mathrm{r}_{\mathcal{C}}(s_i, \alpha_i)$$

The optimal expected long-run average reward $\mathrm{aR}_{\mathcal{C}}^{\mathrm{opt}}$ of a CTMDP $\mathcal{C}$ is defined analogously to $\mathrm{aR}_{\mathcal{M}}^{\mathrm{opt}}$ on MRA (see Sect. 2). As shown in [13], for a unichain CTMDP $\mathcal{C}$ we have $\forall s, s' \in S : \mathrm{aR}_{\mathcal{C}}^{\mathrm{opt}}(s) = \mathrm{aR}_{\mathcal{C}}^{\mathrm{opt}}(s')$. In the future we will refer to this value as $\mathrm{aR}_{\mathcal{C}}^{\mathrm{opt}}$.

Transformation to Continuous-Time MDP. Let $\mathcal{M}$ be a unichain MRA. We construct the CTMDP $\mathcal{C}_{\mathcal{M}} = (S_{\mathcal{C}}, Act_{\mathcal{C}}, \mathrm{R}_{\mathcal{C}})$ with reward structure $(\rho_{\mathcal{C}}, \mathrm{r}_{\mathcal{C}})$ as follows:

- $S_{\mathcal{C}} := MS_{\mathcal{M}}$;
- The set $Act_{\mathcal{C}}$ is obtained as follows. Let $s \in MS_{\mathcal{M}}$, then we denote as PS_s the set of all probabilistic states $s' \in PS_{\mathcal{M}}$ reachable from s via the transition relation $\hookrightarrow$. Let A_s be a function $A_s : PS_s \to Act$, s.t. $A_s(s') \in Act(s')$. Then the set of all enabled actions $Act_{\mathcal{C}}(s)$ for state s in $\mathcal{C}_{\mathcal{M}}$ is the set of all possible functions A_s, and $Act_{\mathcal{C}} = \bigcup_{s \in MS_{\mathcal{M}}} Act_{\mathcal{C}}(s)$.
- Next, we define the transition matrix $\mathrm{R}_{\mathcal{C}}$. Let $s, s' \in MS_{\mathcal{M}}$, and $\Pi_{PS}(s, A_s, s')$ be the set of all untimed paths in $\mathcal{M}$ from s to s' via only probabilistic states and choosing those actions in the probabilistic states that are defined by A_s. Then $\mathrm{R}_{\mathcal{C}}(s, A_s, s') := E(s) \cdot \sum_{\pi \in \Pi_{PS}(s, A_s, s')} \Pr_{\mathcal{M}}[\pi]$, where $\pi = s \xrightarrow{\perp} s_1 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_k} s'$ and $\Pr_{\mathcal{M}}[\pi] = \mathbb{P}[s, s_1] \cdot \mathbb{P}[s_1, \alpha_1, s_2] \cdots \mathbb{P}[s_k, \alpha_k, s']$.
- $\rho_{\mathcal{C}}(s) := \rho(s)$;
- $\mathrm{r}_{\mathcal{C}}(s, A_s) := \sum_{s' \in S_{\mathcal{C}}} \sum_{\pi \in \Pi_{PS}(s, A_s, s')} \Pr_{\mathcal{M}}[\pi] \cdot \mathrm{r}_{\mathcal{M}}(\pi)$, where $\pi = s \xrightarrow{\perp} s_1 \xrightarrow{\alpha_1} \cdots s_k \xrightarrow{\alpha_k} s'$ and $\mathrm{r}_{\mathcal{M}}(\pi) = \sum_{i=1}^{k} \mathrm{r}(s_i, A_s(s_i))$. The action reward for state s and action A_s in $\mathcal{C}$ is therefore the expected accumulated action reward over all successors s'(in $\mathcal{C}$) of state s and over all paths from s to s'.

An example of this transformation is depicted in Fig. 2. One can already see that even in small examples the amount of transitions of the CTMDP corresponding to a MRA can grow extremely fast. If every probabilistic successor $s' \in PS_s$ of a state s in $\mathcal{M}$ has 2 enabled actions, the set of enabled actions $Act_{\mathcal{C}}(s)$ of s in $\mathcal{C}_{\mathcal{M}}$ is $2^{|PS_s|}$. This growth is therefore exponential in the worst-case, and the worst case occurs frequently, due to cascades of probabilistic states.

Remark. It is obvious that this transformation if applied to a unichain MRA yields a unichain CTMDP. Moreover, at each state s of the resulting CTMDP the exit rate is the same across all actions enabled. We therefore refer to this exit rate as $E(s)$.

Theorem 1. $\mathrm{aR}_{\mathcal{C}_{\mathcal{M}}}^{\mathrm{opt}} = \mathrm{aR}_{\mathcal{M}}^{\mathrm{opt}}$

4.2 Dealing with Exponentiality

In this section, we will develop a simple yet efficient solution to cope with exponentiality, harvesting the Bellman equation for CTMDPs [13] together with the structure of $\mathcal{M}$. A naïve direct application to $\mathcal{C}_{\mathcal{M}}$ yields:

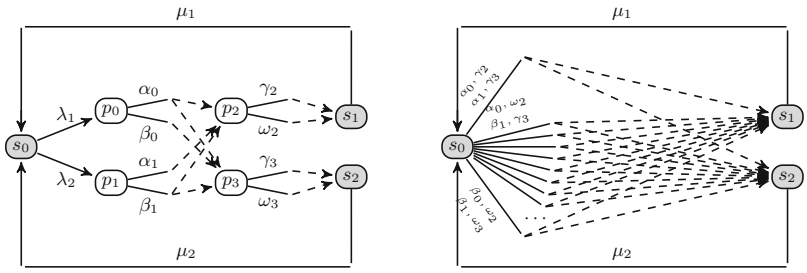


Fig. 2. An example of $\mathcal{M} \to \mathcal{C}_{\mathcal{M}}$ transformation. The MRA $\mathcal{M}$ is depicted on the left and the resulting CTMDP $\mathcal{C}_{\mathcal{M}}$ on the right. In this picture we omitted the probabilities of the probabilistic transitions. If distributions $\mathbb{P}[p_0, \alpha_0, \cdot]$ and $\mathbb{P}[p_0, \alpha_1, \cdot]$ are uniform, then $\mathrm{R}_{\mathcal{C}}(s_0, \frac{\alpha_0,\gamma_2}{\alpha_1,\gamma_3}, s_1) = (\lambda_1{+}\lambda_2){\cdot}\Big[\frac{\lambda_1}{\lambda_1+\lambda_2}(0.5 \cdot 1 + 0.5 \cdot 0) + \frac{\lambda_2}{\lambda_1+\lambda_2}(1 \cdot 1)\Big] = 0.5{\cdot}\lambda_1{+}\lambda_2$.

Theorem 2 (Bellman equation. Inefficient way). *Let* $\mathcal{C}_{\mathcal{M}} = (S_{\mathcal{C}}, Act_{\mathcal{C}}, \mathrm{R}_{\mathcal{C}})$ *and* $(\rho_{\mathcal{C}}, \mathrm{r}_{\mathcal{C}})$ *be a CTMDP and a reward structure obtained through the above transformation. Let* $\mathrm{opt} \in \{\sup, \inf\}$*, then there exists a vector* $h \in \mathbb{R}^{|S_{\mathcal{C}}|}$ *and a unique value* $\mathrm{aR}^{\mathrm{opt}}_{\mathcal{M}} \in \mathbb{R}_{\geqslant 0}$ *that are a solution to the Bellman equation:* $\forall s \in MS_{\mathcal{M}}$:

$$\frac{\mathrm{aR}^{\mathrm{opt}}_{\mathcal{M}}}{E(s)} + h(s) = \operatorname*{opt}_{\alpha \in Act(s)} \left\{ \mathrm{r}_{\mathcal{C}}(s,\alpha) + \frac{\rho_{\mathcal{C}}(s)}{E(s)} + \sum_{s' \in S_{\mathcal{C}}} \mathbb{P}_{\mathcal{C}}[s,\alpha,s'] \cdot h(s') \right\} \tag{1}$$

It is easy to see that the only source of inefficiency in this case is the optimisation operation on the right-hand side, performed over possibly exponentially many actions. Left untreated, this operation in essence is a brute force check of optimality of each action. We will now show how to avoid this problem by working with $\mathcal{M}$ itself and not with $\mathcal{C}_{\mathcal{M}}$. Informally, we will show that the right-hand side optimisation problem on $\mathcal{C}_{\mathcal{M}}$ is nothing more than a *total expected reward* problem on a *discrete-time Markov decision process*. Knowing this, we can apply well-known dynamic programming techniques to solve this problem.

MDPs and Total Expected Reward. We will first need to briefly introduce Markov decision processes and the total expected reward problem.

Definition 3. *A* Markov decision process *(MDP) is a tuple* $\mathcal{D} = (S_{\mathcal{D}}, s_0, Act_{\mathcal{D}}, \mathbb{P}_{\mathcal{D}})$ *where* $S_{\mathcal{D}}$ *is a finite set of* states*,* s_0 *is the initial state,* $Act_{\mathcal{D}}$ *is a finite set of* actions*, and* $\mathbb{P}_{\mathcal{D}} : S_{\mathcal{D}} \times Act_{\mathcal{D}} \to \mathrm{Dist}(S_{\mathcal{D}})$ *is a probabilistic transition matrix.*

The definitions of paths, schedulers and other related notions are analogous to those of CTMDP. In contrast to CTMDPs and MRA, MDPs run in discrete time. A reward structure on an MDP is a function $\mathrm{r}_{\mathcal{D}} : S_{\mathcal{D}} \times Act_{\mathcal{D}} \to \mathbb{R}_{\geqslant 0}$.

Let X_i^s, Y_i^s be random variables denoting the state occupied by $\mathcal{D}$ and the action chosen at step i starting from state s. Then the value

$$\mathrm{tR}^{\mathrm{opt}}_{\mathcal{D},\mathrm{r}_\mathcal{D}}(s) := \operatorname*{opt}_{D \in GM_\mathcal{D}} \mathbf{E}_{s,D}\left[\lim_{N\to\infty} \sum_{i=0}^{N-1} \mathrm{r}_\mathcal{D}(X_i^s, Y_i^s)\right],$$

where $\mathrm{opt} \in \{\sup, \inf\}$, denotes the *optimal total expected reward* on $\mathcal{D}$ with reward structure $\mathrm{r}_\mathcal{D}$, starting from state s [2].

The total expected reward problem on MDPs is a well-established problem that admits policy-iteration and LP-based algorithms [13]. Moreover, for acyclic MDPs it can be computed by the classical finite horizon dynamic programming approach [2], in which each state has to be visited only once. We will present now the iterative scheme that can be used to compute $\mathrm{tR}^{\mathrm{opt}}$ on an acyclic MDP.

A state of an MDP is a *terminal state* if all its outgoing transitions are self-loops with probability 1 and reward 0. We call an MDP *acyclic* if the self-loops of terminal states are its only loops. We say that a non-terminal state s *has maximal depth* i, or $d(s) = i$, if the longest path π from s until a terminal state has length $|\pi| = i$. We define $d(t) := 0$. The following is the iterative scheme to compute the value $\mathrm{tR}^{\mathrm{opt}}$ on $\mathcal{D}$:

$$v_{d(s)}(s) = \begin{cases} 0 & d(s) = 0 \\ \operatorname*{opt}_{\alpha \in Act} \left\{ \mathrm{rew}_\mathcal{D}(s,\alpha) + \sum_{s' \in S} \mathbb{P}[s,\alpha,s'] v_{d(s')}(s') \right\} & d(s) > 0 \end{cases} \tag{2}$$

Theorem 3. $\mathrm{tR}^{\mathrm{opt}}(s) = v_{d(s)}(s)$

Transformation to Discrete-Time MDP. Let $E^{\max}_\mathcal{M}$ be the maximal exit rate among all the Markovian states of $\mathcal{M}$ and $\lambda > E^{\max}_\mathcal{M}$. We will present now a linear transformation from $\mathcal{M}$ to the *terminal MDP* $\mathcal{D}^\lambda_\mathcal{M}$:

1. At first we obtain the MDP $\mathcal{D}_\lambda = (S, s_0, Act', \mathbb{P}_\lambda)$ with $Act' = Act \dot\cup \{\bot\}$. This MDP contains all probabilistic states of $\mathcal{M}$ and their actions. Additionally, we add the Markovian states by making them probabilistic. In each Markovian state only action $\bot$ is enabled. The probability distribution for this action is obtained by *uniformising* the states. Uniformisation with rate λ fixes the means of the residence times (which are discrete quantities, as opposed to the CTMDP formulation) in all Markovian states s to $\frac{1}{\lambda}$ instead of $\frac{1}{E(s)}$. This is achieved by introducing self-loops [13].

$$\mathbb{P}_\lambda[s,\alpha,s'] := \begin{cases} \mathbb{P}[s,\alpha,s'] & \text{for } s \in PS_\mathcal{M},\ \alpha \in Act'(s) \\ \frac{R(s,s')}{\lambda} & \text{for } s \in MS_\mathcal{M},\ \alpha = \bot,\ s' \neq s \\ 1 - \frac{E(s) - R(s,s)}{\lambda} & \text{for } s \in MS_\mathcal{M},\ \alpha = \bot,\ s' = s \end{cases}$$

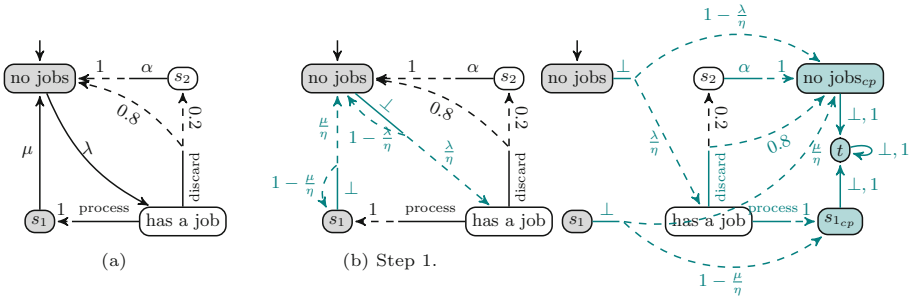


Fig. 3. Transformation to terminal MDP with uniformisation rate η. Figure (a) depicts the original MRA from Fig. 1. The result of the first step of the transformation is shown in figure (b), and the second step is depicted in (c).

2. Next, for each Markovian state, we introduce a copy state and redirect all the transitions leading to Markovian states to these new copy states. Additionally, we introduce a terminal state t, that has only self-loop transitions. Let $\mathcal{D}_\lambda = (S, s_0, Act', \mathbb{P}_\lambda)$ be the MDP obtained in the previous step, then we build $\mathcal{D}^\lambda_\mathcal{M} = (S_\mathcal{D}, s_0, Act', \mathbb{P}_\mathcal{D})$, where $S_{cp} = \{s_{cp} \mid s \in MS_\mathcal{M}\}$, $S_\mathcal{D} = S \dot\cup S_{cp} \dot\cup \{t\}$ and

$$\mathbb{P}'_\mathcal{D}[s,\alpha,s'] = \begin{cases} \mathbb{P}_\lambda[s,\alpha,p] & \text{for } s' = p_{cp} \in S_{cp} \\ \mathbb{P}_\lambda[s,\alpha,s'] & \text{for } s' \in PS_\mathcal{M} \\ 1 & \text{for } s \in S_{cp}, s' = t, \alpha = \perp \\ 1 & \text{for } s, s' = t, \alpha = \perp \end{cases}$$

Figure 3 depicts both steps of the transformation. The resulting MDP is the one that we will use to compute the total expected reward sub-problem.

Efficient Characterisation. We can now present an efficient characterisation of the long-run average reward on unichain MRA.

Let $\mathcal{D}^\lambda_\mathcal{M} = (S_\mathcal{D}, s_0, Act', \mathbb{P}_\mathcal{D})$ be the terminal MDP for $\mathcal{M}$ and $v : S_\mathcal{D} \to \mathbb{R}$. We define the reward structure $\mathrm{rew}_{\mathcal{D},v}$ for $\mathcal{D}^\lambda_\mathcal{M}$ as follows:

$$\mathrm{rew}_{\mathcal{D},v}(s,\alpha) := \begin{cases} \mathrm{r}(s,\alpha) & \text{for } s \in PS_\mathcal{M}, \ \alpha \in Act'(s) \\ \frac{\rho(s)}{\lambda} & \text{for } s \in MS_\mathcal{M}, \alpha = \perp \\ v(s) & \text{for } s \in S_{cp}, \quad \alpha = \perp \\ 0 & \text{for } s = t, \quad \alpha = \perp \end{cases}$$

Theorem 4 (Bellman equation. Efficient way). *There exists a vector $h \in \mathbf{R}^{|MS_\mathcal{M}|}$ and a unique value $\mathrm{aR}^{\mathrm{opt}}_\mathcal{M} \in \mathbf{R}_{\geqslant 0}$ that are a solution to the system:*

$$\forall s \in MS_\mathcal{M} : \quad \frac{\mathrm{aR}^{\mathrm{opt}}_\mathcal{M}}{\lambda} + h(s) = \mathrm{tR}^{\mathrm{opt}}_{\mathcal{D}^\lambda_\mathcal{M}, \mathrm{rew}_{\mathcal{D},h}}(s)$$

The difference between this characterisation and the one derived in Theorem 2 is the right-hand side of the equations. The brute force traversal of exponentially many actions of the former is changed to a total expected reward computed over a linear-sized MDP in the latter.

The correctness of the approach is rooted in two facts. First of all, as a consequence of Theorems 1 and 2 the computation of the long-run average reward of an MRA can be reduced to the same problem on a *continuous-time* MDP. By the results of [13] the latter in turn can be reduced to the long-run average reward problem on its *uniformised discrete-time* MDP. This explains the uniformisation of Markovian states in step 1 of the above transformation, and it explains the reward value $\frac{\rho(s)}{\lambda}$ of the Markovian states. The second observation is more technical. For a Markovian state s the right-hand side of Eq. (1) (Theorem 2) is the total expected reward collected when starting from s in the MDP from step 1, and finishing upon encountering a Markovian state for the second time (the first one being s itself). This explains the addition of copy states in step 2 that lead to a terminal state.

The above equation can be solved with many available techniques, e. g. by policy iteration [13]. This will naturally cover cases where the MDP $\mathcal{D}^{\lambda}_{\mathcal{M}}$ has inherited from $\mathcal{M}$ cycles visiting only probabilistic states (without a chance of getting trapped there, since non-Zenoness is assumed). Such a cycle of probabilistic transitions almost never happens in real-world applications and is usually considered a modelling mistake. In fact, we are not aware of any practical example where that case occurs. We therefore treat separately the class of models that have no cycles of this type and call such MRA *PS-acyclic*.

Theorem 5 (Bellman equation for *PS*-acyclic $\mathcal{M}$). *Let $\mathcal{M}$ be a PS-acyclic unichain MRA. Then there exists a vector $h \in \mathbf{R}^{|MS_{\mathcal{M}}|}$ and a unique value $\mathrm{aR}^{\mathrm{opt}}_{\mathcal{M}} \in \mathbf{R}_{\geqslant 0}$ that are a solution to the Bellman equation:*

$$\forall s \in MS_{\mathcal{M}}: \quad \frac{\mathrm{aR}^{\mathrm{opt}}_{\mathcal{M}}}{\lambda} + h(s) = \frac{\rho(s)}{\lambda} + \sum_{s' \in PS_{\mathcal{M}}} \frac{R(s,s')}{\lambda} \cdot v_{d(s')}(s') + \sum_{\substack{s' \in MS_{\mathcal{M}} \\ s' \neq s}} \frac{R(s,s')}{\lambda} \cdot h(s') + \left(1 - \frac{E(s) - R(s,s)}{\lambda}\right) \cdot h(s)$$

$$\forall s \in PS_{\mathcal{M}}: \quad v_{d(s)}(s) = \operatorname*{opt}_{\alpha \in Act} \left\{ \mathrm{r}(s,\alpha) + \sum_{s' \in MS_{\mathcal{M}}} \mathbb{P}[s,\alpha,s'] \cdot h(s') + \sum_{s' \in PS_{\mathcal{M}}} \mathbb{P}[s,\alpha,s'] \cdot v_{d(s')}(s') \right\},$$

where λ is the uniformisation rate used to construct $\mathcal{D}^{\lambda}_{\mathcal{M}}$ and $d(s)$ denotes the depth of state s in $\mathcal{D}^{\lambda}_{\mathcal{M}}$.

Algorithm 1. `RelativeValueIteration`

input : Unichain MRA $\mathcal{M} = (S, s_0, Act, \hookrightarrow, \rightsquigarrow, \mathrm{r}, \rho)$, opt $\in \{\sup, \inf\}$, approximation error $\varepsilon > 0$
output : $\mathrm{aR}^{\varepsilon}_{\mathcal{M}}$ such that $\|\mathrm{aR}^{\varepsilon}_{\mathcal{M}} - \mathrm{aR}^{\mathrm{opt}}_{\mathcal{M}}\| \leqslant \varepsilon$

1 $\lambda \longleftarrow E^{\max}_{\mathcal{M}} + 1$;
2 $\mathcal{D}^{\lambda}_{\mathcal{M}} \longleftarrow$ terminal MDP obtained as described above;
3 $s^* \longleftarrow$ any Markovian state of $\mathcal{M}$;
4 $v_0 = \overline{0}, v_1 = \overline{1}$;
5 $w_0 = \overline{0}$;
6 **for** $(n = 0;\ sp(v_{n+1} - v_n) < \frac{\varepsilon}{\lambda};\ n++)$ **do**
7 $\quad v_{n+1} = \texttt{TotalExpectedReward}(\mathcal{D}^{\lambda}_{\mathcal{M}}, \mathrm{rew}_{\mathcal{D}^{\lambda}_{\mathcal{M}}, w_n}, \mathrm{opt})$;
8 $\quad w_{n+1} = v_{n+1} - v_{n+1}(s^*) \cdot e$; /* *e is the vector of ones* */
9 **return** $v_{n+1}(s^*) \cdot \lambda$;

4.3 Algorithmic Solution

In order to solve the efficient variant of the Bellman equation, standard value or policy iteration approaches are applicable. In this section, we present the relative value iteration algorithm[1] for this problem (Algorithm 1). This algorithm has two levels of computations: the standard MDP value iteration as an outer loop on Markovian states, and during each iteration of the value iteration we compute the total expected reward on the terminal MDP.

Here $sp(v) := \left| \max_{s \in MS_{\mathcal{M}}} \{v(s)\} - \min_{s \in MS_{\mathcal{M}}} \{v(s)\} \right|$ and `TotalExpectedReward` denotes the function that computes the total expected reward on an MDP.

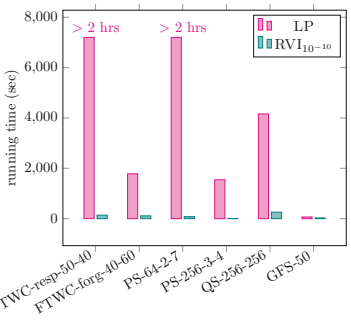
Theorem 6. *Algorithm 1 computes for all $\varepsilon > 0$ the value $\mathrm{aR}^{\varepsilon}_{\mathcal{M}}$, such that $\|\mathrm{aR}^{\varepsilon}_{\mathcal{M}} - \mathrm{aR}^{\mathrm{opt}}_{\mathcal{M}}\| \leqslant \varepsilon$.*

Remark. Notice that in order to obtain the ε-optimal policy that achieves the value $\mathrm{aR}^{\varepsilon}_{\mathcal{M}}$, one only needs to store the optimising actions, computed during the `TotalExpectedReward` phase.

In case $\mathcal{M}$ is *PS*-acyclic, Theorem 5 applies, and instead of the general algorithm computing the total expected reward (Algorithm 1, line **7**), one can resort to its optimised version, that computes the values (2) as defined in Sect. 4.2.

Remark. Needless to say, the CTMDP for a MRA does not necessarily grow exponentially large. So, an alternative approach would be to first build the CTMDP as described in Sect. 4.1 and then, provided that model is small enough, analyse it with standard algorithms for long-run average reward [13]. Since our approach can directly work on the MRA we did not explore this alternative route.

[1] Classical value iteration is also possible, but is known to be numerically unstable.



	$\lvert S\rvert$	$\lvert PS_{\mathcal{M}}\rvert$	$\lvert MS_{\mathcal{M}}\rvert$	$\lvert\pi_{\hookrightarrow}\rvert^{\mathcal{M}}$	$\lambda_{\wedge}^{\mathcal{M}}$
FTWC-resp-50-40	92,819	20,806	72,013	1	5
FTWC-forg-40-60	185,634	20,807	164,827	1	6
PS-64-2-7	454,667	324,616	130,051	2	2
PS-256-3-4	131,529	87,605	43,924	2	3
QS-256-256	465,177	398,096	67,081	4	2
GFS-50	44,916	14,955	29,961	1	2

Fig. 4. Running time comparison of the LP and RVI. The table on the right presents the general data of the models used.

5 Experiments

In this section, we will present the empirical evaluation of the discussed algorithms.

Benchmarks. Our primary interest is to evaluate our approach on real-world examples. We therefore do not consider synthetic benchmarks but rather assess the algorithm on published ones. For this reason the model parameters we can vary is limited. Additionally the degree of variation of some parameters is restricted by the runtime/space requirements of the tool SCOOP [15], used to generate those models. The following is the collection of published benchmark models used to perform the experiments:

PS-S-J-K. The *Polling System* case study [8,16] consists of S servers that process requests of J types, stored in two queues of size K. We enriched this benchmark with rewards denoting maintenance costs. Maintaining a queue yields state reward proportional to its occupancy and processing a request of type j has an action reward dependent on the request type.

QS-K_1-K_2. The *Queuing System* [11] stores requests into two queues of size K_1 and K_2, that are later processed by a server attached to the queue. This model has only state-rewards proportional, which are to the size of the queue.

GFS-N. The *Google File System* [6,7] splits files into chunks, which are maintained by N chunk servers. The system is functioning if it is backed up and for each chunk at least one copy is available. We assign state reward 1 to all the functioning states thus computing the long-run availability of the system.

FTWC-B-N_1-N_2. The *Fault Tolerant Workstation Cluster* [12] models two networks of N_1 and N_2 workstations, interconnected by a switch. The two switches communicate via a backbone. The system is managed by a repairman, his behaviour (B) can be either *responsible*, *forgetful* or *lazy*. Rewards assigned to states and actions denote the cost of repairs, energy consumption and QoS violation.

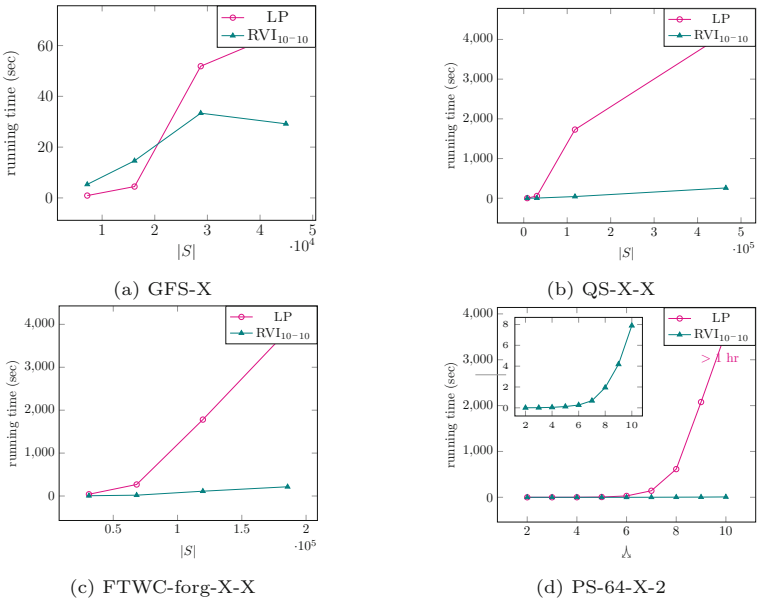


Fig. 5. Runtime complexity of LP and RVI w.r.t. the increase of the model size.

Implementation/Hardware Aspects. We have implemented our approach as a part of the IMCA/MAMA toolset [8], the only toolset providing quantification of long-run average rewards on MRA. IMCA 1.6 contains the implementation of the aR$^{\text{opt}}$ algorithm from [10] that we have discussed in Sect. 3. It uses the SoPlex LP-solver [17] for the solution of linear optimisation problems with the primal and dual feasibility tolerance parameters set to 10^{-6}. All experiments were run on a single core of Intel Core i7-4790 with 8 GB of RAM.

Empirical Evaluation. The space complexity of both the algorithms is polynomial. Therefore, we have used two measures to evaluate the algorithms: running time w.r.t. the increase of precision and model size.

All the models we tested have only one MEC. We will denote the size of this MEC as $|\mathcal{M}|$, and $PS_{\mathcal{M}}$ ($MS_{\mathcal{M}}$) represents the number of probabilistic (Markovian) states of this MEC. We use the symbol $|\pi_{\hookrightarrow}|^{\mathcal{M}}$ to denote the length of the longest path π (in $\mathcal{M}$) that contains only probabilistic states, and $\not\downarrow^{\mathcal{M}}$ stands for the maximal number of enabled actions in probabilistic states of $\mathcal{M}$. RVI$_\varepsilon$ denotes that Algorithm 1 ran with precision ε and LP the LP-based algorithm from [9]. We use the symbol "X" whenever the varying parameter of the experiment is a part of the model name, e.g. PS-2-X.

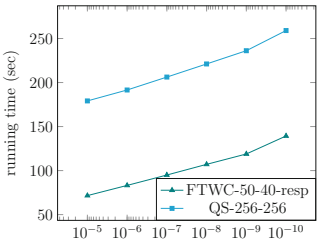


Fig. 6. Observed dependency of RVI on the precision parameter ε in reversed logarithmic x-axis.

Long-Run Average Reward

Efficiency. Figure 4 depicts the comparison of running times of RVI (with precision 10^{-10}) and LP. The running time of RVI on performed experiments is *several orders of magnitude better* than the running time of LP.

Precision. Figure 6 shows the dependency of the computation time of our approach on the precision parameter ε. We observed in all the experiments significant growth of the computation time with the decrease of ε.

Model size. Figure 5 shows the running time comparison of the two algorithms w.r.t. the increase of the model size. In the experiments shown in Fig. 5a–c, both algorithms show a more or less linear dependency on the state space size. The general observation here is that RVI scales much better with the increase of model size than LP. Figure 5a shows that the LP can be better on smaller models, but on larger models RVI takes over. Figure 5d shows the dependency not only on the state space size but also on the maximal number of enabled actions. In this case both algorithms exhibit quadratic dependency with RVI scaling much better than LP.

Remark. All the models we considered (and all case studies we know of) are *PS*-acyclic (which is stronger than our base non-Zenoness assumption). Therefore, Theorem 5 applies that computes the $\mathrm{aR}^{\mathrm{opt}}$ value for *PS*-acyclic MRA. The original LP approach we compare with is, however, not optimised for *PS*-acyclicity.

6 Conclusion

We have presented a novel algorithm for long-run expected rewards for Markov automata. It considers the automaton as a compact representation of a possibly exponentially larger CTMDP. We circumvent exponentiality by applying available algorithms for dynamic programming and for total expected rewards on discrete-time MDPs, derived from the Markov automaton using uniformisation. Experiments on a series of case studies have demonstrated that our algorithm outperforms the available LP-based algorithm by several orders of magnitude. We consider this a genuine breakthrough in Markov automata applicability, in light of the importance of long-run evaluations in performance, dependability and quality-of-service analysis, together with the fact that MAs provide the semantic foundation for engineering frameworks such as (dynamic) fault trees, generalised stochastic Petri nets, and the Architecture Analysis & Design Language (AADL). The general approach we developed is particularly efficient if restricted to Markov automata free of cycles of probabilistic states, which are the only models occurring in practice. Whether or not one should consider all models with such loops as instances of Zeno behaviour is an open question. In fact, a profound understanding of all aspects of Zenoness in Markov automata is not yet developed. It is on our research agenda.

References

1. Bellman, R.E.: Dynamic Programming. Princeton University Press, Princeton (1957)
2. Bertsekas, D.P.: Dynamic Programming and Optimal Control, 2nd edn. Athena Scientific, Belmont (2000)
3. Bhattacharya, R.N., Waymire, E.C.: Stochastic Processes with Applications. SIAM, Philadelphia (2009)
4. Chatterjee, K., Henzinger, M.: Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification. In: Proceedings of SODA, pp. 1318–1336, January 2011
5. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: Proceedings of LICS, pp. 342–351. IEEE CS (2010)
6. Ghemawat, S., Gobioff, H., Leung, S.: The Google file system. In: Scott, M.L., Peterson, L.L. (eds.) Proceedings of SOSP, Bolton Landing, NY, USA, pp. 29–43. ACM, October 2003
7. Guck, D.: Quantitative Analysis of Markov Automata. Master's thesis, RWTH Aachen University, June 2012
8. Guck, D., Hatefi, H., Hermanns, H., Katoen, J.-P., Timmer, M.: Modelling, reduction and analysis of Markov automata. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 55–71. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40196-1_5
9. Guck, D., Hatefi, H., Hermanns, H., Katoen, J., Timmer, M.: Analysis of timed and long-run objectives for Markov automata. Log. Methods Comput. Sci. **10**(3) (2014)
10. Guck, D., Timmer, M., Hatefi, H., Ruijters, E., Stoelinga, M.: Modelling and analysis of Markov reward automata. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 168–184. Springer, Heidelberg (2014). doi:10.1007/978-3-319-11936-6_13
11. Hatefi, H., Hermanns, H.: Model checking algorithms for Markov automata. ECE ASST **53** (2012). http://journal.ub.tu-berlin.de/eceasst/article/view/783
12. Haverkort, B.R., Hermanns, H., Katoen, J.: On the use of model checking techniques for dependability evaluation. In: Proceedings of SRDS, Nürnberg, Germany, pp. 228–237. IEEE CS, October 2000
13. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming, 1st edn. Wiley, New York (1994)
14. Stewart, W.J.: Introduction to the Numerical Solution of Markov Chains. Princeton University Press, Princeton (1994)
15. Timmer, M.: SCOOP: a tool for symbolic optimisations of probabilistic processes. In: Proceedings of QEST, Aachen, Germany, pp. 149–150. IEEE CS, September 2011
16. Timmer, M., Pol, J., Stoelinga, M.I.A.: Confluence reduction for Markov automata. In: Braberman, V., Fribourg, L. (eds.) FORMATS 2013. LNCS, vol. 8053, pp. 243–257. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40229-6_17
17. Wunderling, R.: Paralleler und objektorientierter Simplex-Algorithmus. Ph.D. thesis, Berlin Institute of Technology (1996). http://d-nb.info/950219444