

Pareto Curves for Probabilistic Model Checking

Vojtěch Forejt¹, Marta Kwiatkowska¹, and David Parker²

¹ Department of Computer Science, University of Oxford, UK

² School of Computer Science, University of Birmingham, UK

Abstract. Multi-objective probabilistic model checking provides a way to verify several, possibly conflicting, quantitative properties of a stochastic system. It has useful applications in controller synthesis and compositional probabilistic verification. However, existing methods are based on linear programming, which limits the scale of systems that can be analysed and makes verification of time-bounded properties very difficult. We present a novel approach that addresses both of these shortcomings, based on the generation of successive approximations of the Pareto curve for a multi-objective model checking problem. We illustrate dramatic improvements in efficiency on a large set of benchmarks and show how the ability to visualise Pareto curves significantly enhances the quality of results obtained from current probabilistic verification tools.

1 Introduction

Probabilistic model checking is an automated technique for verifying systems that exhibit stochastic behaviour. This arises due to, for example, failures in physical components, unreliable communication media or randomisation. Systems are typically modelled as Markov chains or Markov decision processes (MDPs), and probabilistic temporal logics are used to specify quantitative properties to be verified such as “the probability of a message packet being lost is less than 0.05” or “the expected energy consumption is at most 100 mJ”.

It is often necessary to incorporate *nondeterminism* into system models, to represent, for example, the actions of an external controller or the order in which a scheduler chooses to interleave system components running in parallel. In these cases, systems are usually modelled as MDPs. Each possible way of resolving the nondeterminism in an MDP is represented by an *adversary* (also known as a strategy or policy). Properties to be verified against the MDP quantify over its adversaries, e.g., “the probability of a message packet being lost is less than 0.05 *for all possible adversaries*”. It is also common to use numerical queries, e.g., “what is the *maximum* expected energy consumption?”.

Model checking reduces to an *optimisation* problem, namely determining the maximum (or minimum) probability (or expected cost/reward) achievable by any adversary. For the most common classes of property (the probability of reaching a set of target states or the expected cumulated reward), model checking can be reduced to solving a linear programming (LP) problem. In practice, however, most probabilistic verification tools use (approximate) iterative numerical methods, such as *value iteration* [19], since they scale to much larger systems and are

amenable to symbolic (BDD-based) implementations. Value iteration can also be used for *time-bounded* (finite-horizon) properties, which is impractical with LP. Another alternative is policy iteration, but this is also impractical for time-bounded properties, and preliminary investigations in [10] showed no particular improvement over value iteration in the context of probabilistic verification.

There has recently been increased interest in *multi-objective* probabilistic model checking for MDPs [5,9,11,4], which can be used to analyse trade-offs between several, possibly conflicting, quantitative properties. Consider, for example, two events of interest, A and B , and let p_A^σ and p_B^σ be the probability that each occurs under an adversary σ of an MDP. In this paper, we study several kinds of multi-objective properties. *Achievability* queries ask, e.g., “is there an adversary σ satisfying the predicate $\psi = p_A^\sigma \geq x \wedge p_B^\sigma \geq y$?” and *numerical* queries ask, e.g., “what is maximum value of x such that ψ is achievable?”. We also consider the *Pareto curve* of *undominated* solution points: for this example, the set of pairs (x, y) such that ψ is achievable but any increase in either x or y would necessitate a decrease in the other.

Multi-objective techniques have natural applications to *controller synthesis* for MDPs (e.g., “how can we maximise the probability of successful message transmission, whilst keeping the expected energy usage below 100 mJ?”). They also form the basis of recent *compositional verification* techniques [15], which decompose model checking into separate tasks for each system component using assume-guarantee reasoning (e.g., “what is the maximum probability of a global system error, under the assumption that component 1 fails with probability at most 0.02?”). This approach has been successfully used to verify probabilistic systems too large to analyse without compositional techniques.

Existing multi-objective model checking methods [5,9,11,4] rely on a reduction to LP. The linear program solved, although of a rather different form to the standard (single objective) case, is still linear in the size of the MDP, yielding polynomial time complexity. As discussed above, though, LP-based probabilistic verification has several important weaknesses. In this paper, we present a novel approach to multi-objective model checking of probabilistic reachability and expected total reward properties. Our method is based on the generation of successive, increasingly precise approximations to the Pareto curve by optimising weighted sums of objectives using value iteration. On a large selection of benchmarks, we demonstrate the following benefits:

- (i) dramatic improvements in run-time *efficiency*, by factors of up to 150;
- (ii) significant *scalability* improvements: over an order of magnitude model size;
- (iii) the usefulness of visualising *Pareto curves* for verification problems;
- (iv) solution of *time-bounded* probabilistic reachability and cumulative reward.

The last of these also paves the way for the development of multi-objective techniques for richer, *timed* classes of models such as continuous-time MDPs.

An extended version of this paper, including proofs, is also available [12].

Related work. Multi-objective optimisation has been extensively studied in areas such as operations research, economics and stochastic control [6], including

its application to MDPs [1]. Many general approaches exist, based on, for example, *normalising* multiple objectives into a single weighted objective; *constrained* approaches optimising one objective while bounding the others; heuristic search using, e.g., *evolutionary algorithms* [7]; application of satisfiability/constraint solvers [17]; and stochastic search with restarts [16]. Several methods, including e.g. [16], iterate over weighted sums of objectives, as we do; the main difference is that our approach is tailored to the convex, linear problems derived from MDPs.

Multi-objective optimisation is routinely used in areas such as embedded systems design and a variety of general-purpose optimisation tools exist, e.g., PISA [2], ModeFRONTIER and libraries for MATLAB. Such tools tend to be targeted at much more complex (e.g., non-convex and non-linear) design spaces than the ones that we focus on in this paper. They are also typically used for *static* design problems, rather than our *dynamic* models of system behaviour.

A rigorous complexity analysis of multi-objective optimisation, in particular for approximating *Pareto curves*, was undertaken in the influential work of [18]. More recently [8], some of these results were improved for the simpler case of *convex* multi-objective problems but no practical investigations are undertaken.

Most relevant to the current work is the application of multi-objective optimisation to *probabilistic verification* [5,9,11,4]. In [5,9,4], discounted total reward, probabilistic ω -regular and long-run average properties are studied, respectively. In each case, algorithms are given using a reduction to LP, also showing the existence of methods to approximate Pareto curves using the results of [18], but implementations are not considered. The work of [11] adds expected total reward properties and provides an implementation, based on LP. As discussed above, the performance and scalability of our approach is significantly better, as we show in Section 5. None of the above consider time-bounded properties. In principle, for discrete-time models like MDPs, these can be reduced to unbounded properties using a finite counter but this is generally impractical in terms of scalability.

2 Background

Geometry. For a vector $\mathbf{x} \in \mathbb{R}^n$, we use x_i to denote its i -th component and say \mathbf{x} is a *weight vector* if $x_i \geq 0$ for all i and $\sum_{i=1}^n x_i = 1$. The *Euclidean inner product* of $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ is defined as $\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^n x_i \cdot y_i$. For a set of vectors $X = \{\mathbf{x}_1, \dots, \mathbf{x}_k\} \subseteq \mathbb{R}^n$, a *convex combination* is $\sum_{j=1}^k w_j \cdot \mathbf{x}_j$ for some weight vector $\mathbf{w} \in \mathbb{R}^k$. We use $\text{down}(X)$ to denote the *downward closure* of the convex hull of X , i.e. the set of vectors $\mathbf{z} \in \mathbb{R}^n$ that satisfy $\mathbf{z} \leq \mathbf{y}$ for some convex combination \mathbf{y} of X . Given a convex set Y , we say that a point $\mathbf{y} \in Y$ is on the *boundary* of Y if, for any $\varepsilon > 0$, there is a point $\mathbf{z} \notin Y$ such that the Euclidean distance between \mathbf{y} and \mathbf{z} is at most ε . From the *separating hyperplane* and *supporting hyperplane* theorems, we have the following.

Proposition 1 ([3]). *Let $Y \subseteq \mathbb{R}^n$ be a downward closed set of points. For any $\mathbf{p} \in \mathbb{R}^n$ not in Y , there is a weight vector $\mathbf{w} \in \mathbb{R}^n$ such that $\mathbf{w} \cdot \mathbf{p} > \mathbf{w} \cdot \mathbf{y}$ for all $\mathbf{y} \in Y$. Also, for any \mathbf{q} on the boundary of Y , there is a weight vector $\mathbf{w} \in \mathbb{R}^n$ such that $\mathbf{w} \cdot \mathbf{q} \geq \mathbf{w} \cdot \mathbf{y}$ for all $\mathbf{y} \in Y$. We say that \mathbf{w} separates \mathbf{q} from $\text{down}(Y)$.*

Markov decision processes (MDPs). MDPs are commonly used to model systems with probabilistic and nondeterministic behaviour. Denoting by $Dist(X)$ the set of probability distributions over a set X , an *MDP* takes the form $\mathcal{M} = (S, \bar{s}, \alpha, \delta)$, where S is a set of states, $\bar{s} \in S$ is an initial state, α is a set of actions and $\delta : S \times \alpha \rightarrow Dist(S)$ is a (partial) probabilistic transition function.

Each state s of an MDP \mathcal{M} has an associated set $A(s)$ of *enabled* actions, given by $A(s) \stackrel{\text{def}}{=} \{a \in \alpha \mid \delta(s, a) \text{ is defined}\}$. If action $a \in A(s)$ is taken in state s , then the next state is determined randomly according to the distribution $\delta(s, a)$, i.e., a transition to state s' occurs with probability $\delta(s, a)(s')$. A *path* through \mathcal{M} is a (finite or infinite) sequence $\pi = s_0 a_0 s_1 a_1 \dots$ where $s_0 = \bar{s}$, and $a_i \in A(s_i)$ and $\delta(s_i, a_i)(s_{i+1}) > 0$ for all i . We denote by *IPaths* (*FPaths*) the set of all infinite (finite) paths and, for finite path π , $last(\pi)$ is its last state.

An *adversary* $\sigma : FPaths \rightarrow Dist(\alpha)$ (also called a strategy or policy) of \mathcal{M} is a resolution of the choices of action in each state, based on its execution so far. In standard fashion [13], an adversary σ induces a probability measure $Pr_{\mathcal{M}}^{\sigma}$ over *IPaths*. An adversary σ is *deterministic* if $\sigma(\pi)$ is a Dirac distribution for all π (and *randomised* if not); it is *memoryless* if $\sigma(\pi)$ depends only on $last(\pi)$. The set of all adversaries for \mathcal{M} is $Adv_{\mathcal{M}}$.

A *reward structure* for \mathcal{M} is a function $\rho : S \times \alpha \rightarrow \mathbb{R}$ mapping actions to (positive or negative) reals. For an infinite path $\pi = s_0 a_0 s_1 a_1 \dots$ and a number $k \in \mathbb{N} \cup \{\infty\}$ the *total reward in k steps* for π over ρ is $\rho[k](\pi) \stackrel{\text{def}}{=} \sum_{i=0}^{k-1} \rho(s_i, a_i)$.

Model checking MDPs. In this paper, we focus on two key classes of properties for MDPs: the *probability of reaching a target* and the *expected total reward*. In each case, we consider both time-bounded and unbounded variants. We will later discuss generalisation to more expressive properties. In this and the following sections, we assume a fixed MDP $\mathcal{M} = (S, \bar{s}, \alpha, \delta)$.

Definition 1 (Reachability predicate). A reachability predicate $[T]_{\sim p}^{\leq k}$ comprises a set of target states $T \subseteq S$, a relational operator $\sim \in \{\geq, \leq\}$, a rational probability bound p and a time-bound $k \in \mathbb{N} \cup \{\infty\}$. It states that the probability of reaching T within k steps satisfies $\sim p$. Formally, satisfaction of $[T]_{\sim p}^{\leq k}$ by MDP \mathcal{M} , under adversary σ , denoted $\mathcal{M}, \sigma \models [T]_{\sim p}^{\leq k}$, is defined as follows:

$$\mathcal{M}, \sigma \models [T]_{\sim p}^{\leq k} \Leftrightarrow Pr_{\mathcal{M}}^{\sigma}(\{s_0 a_0 s_1 a_1 \dots \in IPaths \mid \exists i \leq k : s_i \in T\}) \sim p.$$

Definition 2 (Reward predicate). A reward predicate $[\rho]_{\sim r}^{\leq k}$ comprises a reward structure $\rho : S \times \alpha \rightarrow \mathbb{R}$, a relational operator $\sim \in \{\geq, \leq\}$, a rational reward bound r and a time bound $k \in \mathbb{N} \cup \{\infty\}$. It states that the expected total reward cumulated within k steps satisfies $\sim r$. Formally, satisfaction of $[\rho]_{\sim r}^{\leq k}$ by \mathcal{M} , under adversary σ , denoted $\mathcal{M}, \sigma \models [\rho]_{\sim r}^{\leq k}$, is defined as follows:

$$\mathcal{M}, \sigma \models [\rho]_{\sim r}^{\leq k} \Leftrightarrow ExpTot_{\mathcal{M}}^{\sigma, k}(\rho) \sim r \text{ where } ExpTot_{\mathcal{M}}^{\sigma, k}(\rho) \stackrel{\text{def}}{=} \int_{\pi} \rho[k](\pi) dPr_{\mathcal{M}}^{\sigma}.$$

For the *unbounded* forms of the notation above ($k = \infty$), we will often omit k , writing e.g. $[\rho]_{\sim r}$ instead of $[\rho]_{\sim r}^{\leq \infty}$ or $ExpTot_{\mathcal{M}}^{\sigma}(\rho)$ instead of $ExpTot_{\mathcal{M}}^{\sigma, \infty}(\rho)$.

For this paper, we also need to consider *weighted sums* of rewards.

Definition 3 (Weighted reward sum). Given a weight vector $\mathbf{w} \in \mathbb{R}^n$ and vectors of time bounds $\mathbf{k} = (k_1, \dots, k_n) \in (\mathbb{N} \cup \{\infty\})^n$ and reward structures $\boldsymbol{\rho} = (\rho_1, \dots, \rho_n)$ for MDP \mathcal{M} , the weighted reward sum $\mathbf{w} \cdot \boldsymbol{\rho}[\mathbf{k}]$ over a path π is defined as $\mathbf{w} \cdot \boldsymbol{\rho}[\mathbf{k}](\pi) \stackrel{\text{def}}{=} \sum_{i=1}^n w_i \rho_i[\mathbf{k}](\pi)$. The expected total weighted sum is then: $\text{ExpTot}_{\mathcal{M}}^{\sigma, \mathbf{k}}(\mathbf{w} \cdot \boldsymbol{\rho}) \stackrel{\text{def}}{=} \int_{\pi} \mathbf{w} \cdot \boldsymbol{\rho}[\mathbf{k}](\pi) dPr_{\mathcal{M}}^{\sigma}$. For any adversary σ , we have: $\text{ExpTot}_{\mathcal{M}}^{\sigma, \mathbf{k}}(\mathbf{w} \cdot \boldsymbol{\rho}) = \sum_{i=1}^n w_i \text{ExpTot}_{\mathcal{M}}^{\sigma, k_i}(\rho_i)$.

Notice that satisfaction of reachability and reward predicates is defined above with respect to a specific adversary σ of an MDP \mathcal{M} . When performing model checking on the MDP, the most common approach is to verify that such a predicate is satisfied *for all* adversaries $\sigma \in \text{Adv}_{\mathcal{M}}$. An alternative, often described as *controller synthesis*, is to ask the dual question: whether *there exists* an adversary σ satisfying the predicate. In either case, model checking reduces to computing the maximum or minimum reachability probability or expected reward. For the unbounded cases, this can be done by solving an LP problem, using policy iteration, or with value iteration, an approximate iterative numerical method [19]. For time-bounded properties, only value iteration is applicable.

3 Multi-objective Queries

We now describe how to formalise multi-objective queries for MDPs. In the following section, we will present novel, efficient algorithms for their verification. We formulate our queries in a similar style to the one taken in [11], but with two key additions. Firstly, we include the ability to specify *time-bounded* reachability and reward properties. Secondly, we consider *Pareto curves*.

The essence of multi-objective properties for MDPs is that they require multiple predicates to be satisfied concurrently for the same adversary.

Definition 4 (Multi-objective predicate). A multi-objective predicate is a vector $\boldsymbol{\psi} = (\psi_1, \dots, \psi_n)$ of reachability or reward predicates. We say that $\boldsymbol{\psi}$ is satisfied by MDP \mathcal{M} under adversary σ , denoted $\mathcal{M}, \sigma \models \boldsymbol{\psi}$, if $\mathcal{M}, \sigma \models \psi_i$ for all $1 \leq i \leq n$. We call $\boldsymbol{\psi}$ a basic multi-objective predicate if it is of the form $([\rho_1]_{\geq r_1}^{\leq k_1}, \dots, [\rho_n]_{\geq r_n}^{\leq k_n})$, i.e. it comprises only lower-bounded reward predicates.

We define three ways to formulate multi-objective queries for an MDP: *achievability queries*, which check for the existence of an adversary satisfying a multi-objective predicate $\boldsymbol{\psi}$; *numerical queries*, which maximise or minimise a reachability/reward objective over the set of adversaries satisfying $\boldsymbol{\psi}$; and *Pareto queries*, which determine the Pareto curve for a set of objectives.

Definition 5 (Achievability query). For MDP \mathcal{M} and multi-objective predicate $\boldsymbol{\psi}$, an achievability query asks if $\boldsymbol{\psi}$ is satisfiable (or achievable), i.e. whether there exists an adversary $\sigma \in \text{Adv}_{\mathcal{M}}$ such that $\mathcal{M}, \sigma \models \boldsymbol{\psi}$.

Definition 6 (Numerical query). For MDP \mathcal{M} , a numerical query is of the form $\text{num}([o_1]_{\star}^{\leq k_1}, (\psi_2 \dots, \psi_n))$, comprising an $n-1$ -sized multi-objective predicate $(\psi_2 \dots, \psi_n)$ and an objective $[o_1]_{\star}^{\leq k_1}$, where o_1 is a reward structure ρ_1 or

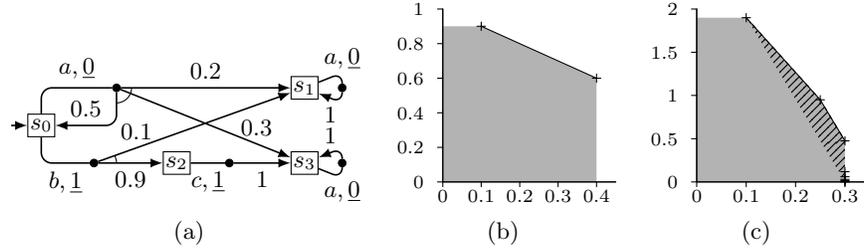


Fig. 1. Example MDP (a), graphs for $(\{\{s_1\}\}_{\geq x}, \{\{s_3\}\}_{\geq y})$ (b) and $(\{\{s_1\}\}_{\leq 2}, [\rho]_{\geq y})$ (c).

target set T_1 , $k_1 \in \mathbb{N} \cup \{\infty\}$ is a time bound and $\star \in \{\min, \max\}$. We define:

$$\begin{aligned} \text{num}([o_1]_{\min}^{\leq k_1}, (\psi_2, \dots, \psi_n)) &\stackrel{\text{def}}{=} \inf\{x \in \mathbb{R} \mid ([o_1]_{\leq x}^{\leq k_1}, \psi_2, \dots, \psi_n) \text{ is satisfiable}\}. \\ \text{num}([o_1]_{\max}^{\leq k_1}, (\psi_2, \dots, \psi_n)) &\stackrel{\text{def}}{=} \sup\{x \in \mathbb{R} \mid ([o_1]_{\geq x}^{\leq k_1}, \psi_2, \dots, \psi_n) \text{ is satisfiable}\}. \end{aligned}$$

Definition 7 (Pareto query). For MDP \mathcal{M} , a Pareto query takes the form $\text{pareto}([o_1]_{\star_1}^{\leq k_1}, \dots, [o_n]_{\star_n}^{\leq k_n})$, where each $[o_i]_{\star_i}^{\leq k_i}$ is an objective as in Defn. 6. The set of achievable values is $A = \{\mathbf{x} \in \mathbb{R}^n \mid ([o_1]_{\star_1}^{\leq k_1}, \dots, [o_n]_{\star_n}^{\leq k_n}) \text{ is satisfiable}\}$ where $\sim_i = \geq$ if $\star_i = \max$ and $\sim_i = \leq$ if $\star_i = \min$. We say, for points $\mathbf{x}, \mathbf{y} \in A$, that \mathbf{x} dominates \mathbf{y} if $x_i \sim_i y_i$ for all i and $x_j > y_j$ for some j . Then:

$$\text{pareto}([o_1]_{\star_1}^{\leq k_1}, \dots, [o_n]_{\star_n}^{\leq k_n}) \stackrel{\text{def}}{=} \{\mathbf{x} \in A \mid \mathbf{x} \text{ is not dominated by any } \mathbf{y} \in A\}.$$

Convexity. A fundamental property of the multi-objective optimisation problems solved in this paper (and on MDPs in general) is their *convexity*. More precisely, consider target sets T_1, \dots, T_n , reward structures ρ_1, \dots, ρ_m and time-bounds $k_1, \dots, k_n, l_1, \dots, l_m \in \mathbb{N} \cup \{\infty\}$. Let $\mathbf{x}^\sigma \in \mathbb{R}^{n+m}$ be the vector defined such that $x_i = \text{Pr}_{\mathcal{M}}^\sigma(\diamond^{\leq k_i} T_i)$ for $1 \leq i \leq n$ and $x_{n+j} = \text{ExpTot}_{\mathcal{M}}^{\sigma, k_j}(\rho_j)$ for $1 \leq j \leq m$, where $\text{Pr}_{\mathcal{M}}^\sigma(\diamond^{\leq k_i} T_i)$ denotes the probability of reaching T_i in k_i steps under σ . Then, the set $\{\mathbf{x}^\sigma \mid \sigma \in \text{Adv}_{\mathcal{M}}\}$ forms a convex polytope [9,11]¹. As a direct consequence of this, the set of *achievable values* for a Pareto query is also convex.

Example 1. Fig. 1(a) shows an MDP with accompanying reward structure ρ indicated by underlined numbers. Consider first the multi-objective predicate $\psi = (\{s_1\} \geq x, \{s_3\} \geq y)$, which imposes lower bounds on the probabilities of reaching states s_1 and s_3 . The grey area in Fig. 1(b) shows the values of x and y for which ψ is satisfiable. The two points on the graph marked as + correspond to the two possible *memoryless deterministic* adversaries in \mathcal{M} . The line joining them (their convex closure) represents the points for all possible adversaries. For this example, this line also constitutes the Pareto curve. Achievability queries on ψ for $(x, y) = (0.2, 0.7)$ and $(0.4, 0.7)$ return true and false, respectively. Numerical query $\text{num}(\{s_1\}_{\max}, (\{s_3\}_{\geq 0.7}))$ returns 0.3.

Consider a second predicate $\psi' = (\{s_1\} \leq 2, [\rho] \geq y)$, now with a time-bounded reachability and reward predicate. Fig. 1(c) depicts (by +) points for some of the

¹ Strictly speaking, this requires finiteness of rewards, which we discuss below.

deterministic adversaries of \mathcal{M} , of which there are infinitely many. Their convex combination, the dashed area, marks the points achievable by *all* (randomised) adversaries, and its downward closure, in grey, shows the values of x and y for which ψ' is satisfiable. The Pareto curve is the black line along the top edge.

Assumptions. For the purposes of model checking the queries described in this section, we need to impose certain restrictions on the use of rewards. For clarity, we describe these in terms of achievability queries but they apply to all three classes. We first need the following definition.

Definition 8 (Reward-finiteness). *Let \mathcal{M} be an MDP and consider an achievability query $\psi = ([T_1]_{\sim_1 p_1}^{\leq k_1}, \dots, [T_n]_{\sim_n p_n}^{\leq k_n}, [\rho_1]_{\bowtie_1 r_1}^{\leq l_1}, \dots, [\rho_m]_{\bowtie_m r_m}^{\leq l_m})$ for \mathcal{M} . We say that ψ is reward-finite if, for each $1 \leq j \leq m$ such that $l_j = \infty$ and $\bowtie_j = \geq$, we have: $\sup\{\text{ExpTot}_{\mathcal{M}}^{\sigma, l_j}(\rho_j) \mid \mathcal{M}, \sigma \models ([T_1]_{\sim_1 p_1}^{\leq k_1}, \dots, [T_n]_{\sim_n p_n}^{\leq k_n})\} < \infty$ and is fully reward-finite if, for each $1 \leq j \leq m$ such that $l_j = \infty$ and $\bowtie_j = \geq$, we have: $\sup\{\text{ExpTot}_{\mathcal{M}}^{\sigma, l_j}(\rho_j) \mid \sigma \in \text{Adv}_{\mathcal{M}}\} < \infty$.*

Let \mathcal{M} and ψ be as in Defn. 8. To model check ψ on \mathcal{M} , we require that: (i) each reward structure ρ_i assigns only non-negative values; (ii) ψ is reward-finite; and (iii) for indices $1 \leq j \leq m$ such that $l_j = \infty$, either all \bowtie_j s are \geq or all are \leq .

Condition (ii) imposes natural restrictions on finiteness of rewards. Notice that we only require finiteness for adversaries which satisfy the probabilistic predicates contained in ψ . We adopt this approach from [11] where, in addition, algorithms are given to check that ψ is reward-finite and to construct a modified MDP that is equivalent (in terms of satisfiability of ψ) but for which ψ is *fully* reward-finite. This can be checked by a simpler multi-objective query containing only probabilistic predicates. Thus, in the remainder of this paper, we assume that all queries are fully reward-finite.

Condition (iii) ensures that the algorithms we define in the next section do not need to compute *unbounded expected total rewards* for MDPs with *both positive and negative rewards*, which is unsound. For unbounded reachability predicates, again using methods from [11], we can easily invert their bounds (to match those of any reward predicates) by making a simple change to the MDP.

Extensions. We also remark that the class of multi-objective properties outlined in this section can be extended in several respects. In particular, as shown in [11], we can add support for probabilistic ω -regular (e.g. LTL) properties via reduction to probabilistic reachability on a product of the MDP and one or more deterministic Rabin automata. That work also allows arbitrary Boolean combinations of predicates, which are reduced to disjunctive normal form and treated separately. Both of these extensions can be adapted to our setting; the former we have implemented and used for our experiments in Section 5.

4 Multi-objective Probabilistic Model Checking

We now present efficient algorithms for checking the multi-objective queries defined in the previous section. Proofs of correctness can be found in [12].

Input: MDP \mathcal{M} , multi-objective predicate $\psi = ([\rho_1]_{\geq r_1}^{\leq k_1}, \dots, [\rho_n]_{\geq r_n}^{\leq k_n})$
Output: true if ψ is achievable, false if not

- 1 $X := \emptyset$; $\rho = (\rho_1, \dots, \rho_n)$; $\mathbf{k} = (k_1, \dots, k_n)$; $\mathbf{r} = (r_1, \dots, r_n)$;
- 2 **do**
- 3 Find \mathbf{w} separating \mathbf{r} from $\text{down}(X)$;
- 4 Find adversary σ maximising $\text{ExpTot}_{\mathcal{M}}^{\sigma, \mathbf{k}}(\mathbf{w} \cdot \rho)$;
- 5 $\mathbf{q} := (\text{ExpTot}_{\mathcal{M}}^{\sigma, k_i}(\rho_i))_{1 \leq i \leq n}$;
- 6 **if** $\mathbf{w} \cdot \mathbf{q} < \mathbf{w} \cdot \mathbf{r}$ **then return false**;
- 7 $X := X \cup \{\mathbf{q}\}$;
- 8 **while** $\mathbf{r} \notin \text{down}(X)$;
- 9 **return true**;

Alg. 1. Basic algorithm for checking achievability queries

Reduction to basic form. The first step when checking any type of query is to reduce the problem to one over a *basic* predicate on a modified MDP. We do so by converting reachability predicates into reward predicates (by adding a one-off reward of 1 upon reaching the target) and then negating objectives for predicates with upper bounds. Formally, we do the following.

Proposition 2. *Let $\mathcal{M} = (S, \bar{s}, \alpha, \delta)$ be an MDP and $\psi = ([T_1]_{\sim_{1p_1}}^{\leq k_1}, \dots, [T_n]_{\sim_{np_n}}^{\leq k_n}, [\rho_1]_{\bowtie_{1r_1}}^{\leq l_1}, \dots, [\rho_m]_{\bowtie_{mr_m}}^{\leq l_m})$ be a multi-objective predicate. Let $\mathcal{M}' = (S', (\bar{s}, \emptyset), \alpha', \delta')$ be the MDP defined as follows: $S' = S \times 2^{\{1, \dots, n\}}$, $\alpha' = \alpha \times 2^{\{1, \dots, n\}}$ and, for all $s, s' \in S$, $a \in \alpha$ and $c \subseteq \{1, \dots, n\}$:*

- $\delta'((s, c), (a, c'))((s', c \cup c')) = \delta(s, a)(s')$ where $c' = \{i \mid s \in T_i\} \setminus c$;
- $\delta'((s, c), a')((s', c')) = 0$ for all other c, c' and a' .

Now, let ψ' be $([\rho_{T_1}]_{\geq p_1}^{\leq k_1+1}, \dots, [\rho_{T_n}]_{\geq p_n}^{\leq k_n+1}, [\bar{\rho}_1]_{\geq r_1}^{\leq l_1}, \dots, [\bar{\rho}_m]_{\geq r_m}^{\leq l_m})$, where: reward $\rho_{T_i}((s, c), (a, c'))$ is equal to 1 if $i \in c'$ and $\sim_i = \geq$, to -1 if $\sim_i = \leq$, and to 0 otherwise; and $\bar{\rho}_i((s, c), (a, c'))$ is equal to $\rho_i(s, a)$ if $\bowtie_i = \geq$ and to $-\rho_i(s, a)$ if $\sim_i = \leq$. Then ψ is satisfiable in \mathcal{M} if and only if ψ' is satisfiable in \mathcal{M}' .

Notice that the reduction described above results in reward structures with both positive and negative rewards. For time-bounded properties, this is not a concern. For unbounded ones, we must take care that they are all either non-negative or non-positive, as mentioned earlier in our discussion of condition (iii).

Achievability queries. We begin with achievability queries. We first give an outline of the overall algorithm; subsequently, we will describe in more detail how it is implemented in practice using value iteration.

By applying the reduction described above, we only need to consider the case of a *basic* multi-objective predicate $\psi = ([\rho_1]_{\geq r_1}^{\leq k_1}, \dots, [\rho_n]_{\geq r_n}^{\leq k_n})$. Alg. 1 shows how to check if ψ is satisfiable. It works by generating a sequence of weight vectors \mathbf{w} and optimising a \mathbf{w} -weighted sum of the n objectives. A resulting optimal adversary σ is then used to generate a point \mathbf{q} which is guaranteed to be contained on the Pareto curve for ψ , and a collection X of such points is assembled. Each new weight vector \mathbf{w} is identified by finding a separating hyperplane between $\text{down}(X)$ and $\mathbf{r} = (r_1, \dots, r_n)$. Once \mathbf{r} is found to be contained in $\text{down}(X)$, we

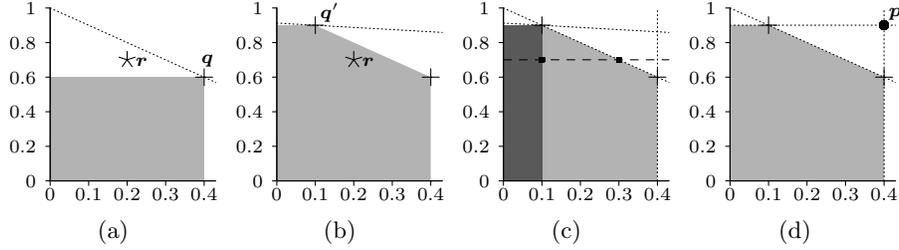


Fig. 2. Example executions of Algorithms 1, 3 and 4 (see Examples 2, 3 and 4).

know that ψ is achievable. If, on the other hand, $\mathbf{w} \cdot \mathbf{q} < \mathbf{w} \cdot \mathbf{r}$, then we know that ψ cannot possibly be achievable.

Correctness of Alg. 1 is proved in [12]. Termination is guaranteed by the fact that each iteration of the loop identifies a point \mathbf{q} on a unique face of the Pareto curve. In the worst case, the number of faces is exponential in $|\mathcal{M}|$, k and n [9]; however, our experimental results (in Section 5) show the number of steps is usually small on practical examples. The individual model checking problems solved in each step (lines 4-5 of Alg. 1) require time polynomial in $|\mathcal{M}|$. We describe their practical implementation in the next section.

Example 2. We illustrate the execution of Alg. 1 on the MDP from Example 1 and the achievability query $(\{s_1\} \geq 0.2, \{s_3\} \geq 0.7)$. Let us assume that we have already applied Proposition 2 so that we have an equivalent reward predicate $([\rho_1] \geq 0.2, [\rho_2] \geq 0.7)$ (the full reduction is in [12]). As a first (arbitrary) weight vector, we pick $\mathbf{w} = (0.5, 0.5)$ and then maximise $\mathbf{w} \cdot \boldsymbol{\rho}$. The resulting optimal adversary σ (which chooses a in s_0) gives $\mathbf{q} = (\text{ExpTot}_{\mathcal{M}}^{\sigma}(\rho_1), \text{ExpTot}_{\mathcal{M}}^{\sigma}(\rho_2)) = (0.4, 0.6)$ and we have $X = \{\mathbf{q}\}$. Fig. 2(a) shows the point \mathbf{q} (as $+$) and the target point $\mathbf{r} = (0.2, 0.7)$ (as \star). The dotted line represents the hyperplane with orientation \mathbf{w} passing through \mathbf{q} (i.e. the points \mathbf{x} for which $\mathbf{w} \cdot \mathbf{x} = \mathbf{w} \cdot \mathbf{q} = 0.5$), the points above which correspond to unachievable value pairs. The grey region is $\text{down}(X)$, in which all points are achievable. Since $\mathbf{r} \notin \text{down}(X)$, we continue.

Next, we pick a weight $\mathbf{w}' = (0.1, 0.8)$. Maximising $\mathbf{w}' \cdot \boldsymbol{\rho}$ results in adversary σ' (which chooses b in s_0) giving $\mathbf{q}' = (0.1, 0.9)$, which we add to X . Fig. 2(b) again shows both points in X , $\text{down}(X)$ and \mathbf{r} . It also plots points \mathbf{x} for which $\mathbf{w}' \cdot \mathbf{x} = \mathbf{w}' \cdot \mathbf{q}' = 0.73$. Since \mathbf{r} is now in $\text{down}(X)$, the algorithm returns true.

Value iteration. The most expensive part of Alg. 1 in practice is the combination of lines 4-5, which computes the maximum possible value for a weighted sum of reward objectives, determines a corresponding optimal adversary σ , and then finds the value for the n individual objectives under σ .

Alg. 2 shows how to perform all these tasks using a value iteration-style computation. One key difference between this algorithm and standard value iteration is that it needs to optimise a combination of unbounded and bounded properties. This is done in three phases (lines 3-8, 9-13 and 14-20). The first two correspond to the unbounded part; the third to the bounded part.

Another important difference is that the algorithm performs the optimisation of the weighted sum $\mathbf{w} \cdot \boldsymbol{\rho}[\mathbf{k}]$ and the computation of the vector of individual objective values $\mathbf{q} = (\text{ExpTot}_{\mathcal{M}}^{\sigma, k_i}(\rho_i))_{1 \leq i \leq n}$ simultaneously: the former in phases

```

Input: MDP  $\mathcal{M}=(S, \bar{s}, \alpha, \delta)$ , weight vect.  $\mathbf{w}$ , reward structures  $\boldsymbol{\rho}=(\rho_1, \dots, \rho_n)$ ,
vector of time bounds  $\mathbf{k} \in (\mathbb{N} \cup \{\infty\})^n$ , convergence threshold  $\varepsilon$ 
Output: Adv.  $\sigma$  maximising  $ExpTot_{\mathcal{M}}^{\sigma, \mathbf{k}}(\mathbf{w} \cdot \boldsymbol{\rho})$ ,  $\mathbf{q} = (ExpTot_{\mathcal{M}}^{\sigma, k_i}(\rho_i))_{1 \leq i \leq n}$ 
1  $\mathbf{x} := \mathbf{0}; \mathbf{x}^1 := \mathbf{0}; \dots; \mathbf{x}^n := \mathbf{0}; \mathbf{y} = \mathbf{0}; \mathbf{y}^1 := \mathbf{0}; \dots; \mathbf{y}^n := \mathbf{0};$ 
2  $\sigma^\infty(s) = \perp$  for all  $s \in S$ ;
3 do
4   foreach  $s \in S$  do
5      $y_s := \max_{a \in A(s)} (\sum_{\{i | k_i = \infty\}} w_i \cdot \rho_i(s, a) + \sum_{s' \in S} \delta(s, a)(s') \cdot x_{s'});$ 
6      $\sigma^\infty(s) := \arg \max_{a \in A(s)} (\sum_{\{i | k_i = \infty\}} w_i \cdot \rho_i(s, a) + \sum_{s' \in S} \delta(s, a)(s') \cdot x_{s'});$ 
7    $\delta := \max_{s \in S} (y_s - x_s); \mathbf{x} := \mathbf{y};$ 
8 while  $\delta > \varepsilon;$ 
9 do
10  foreach  $s \in S$  and  $i \in \{1, \dots, n\}$  where  $k_i = \infty$  do
11     $y_s^i := \rho_i(s, \sigma^\infty(s)) + \sum_{s' \in S} \delta(s, \sigma^\infty(s))(s') \cdot x_{s'}^i;$ 
12   $\delta := \max_{i=1}^n \max_{s \in S} (y_s^i - x_s^i); \mathbf{x}^1 := \mathbf{y}^1; \dots; \mathbf{x}^n := \mathbf{y}^n;$ 
13 while  $\delta > \varepsilon;$ 
14 for  $j = \max\{k_\ell < \infty \mid \ell \in \{1, \dots, n\}\}$  down to 1 do
15   foreach  $s \in S$  do
16      $y_s := \max_{a \in A(s)} (\sum_{\{i | k_i \geq j\}} w_i \cdot \rho_i(s, a) + \sum_{s' \in S} \delta(s, a)(s') \cdot x_{s'});$ 
17      $\sigma^j(s) := \arg \max_{a \in A(s)} (\sum_{\{i | k_i \geq j\}} w_i \cdot \rho_i(s, a) + \sum_{s' \in S} \delta(s, a)(s') \cdot x_{s'});$ 
18     foreach  $i \in \{1, \dots, n\}$  where  $k_i \geq j$  do
19        $y_s^i := \rho_i(s, \sigma^j(s)) + \sum_{s' \in S} \delta(s, \sigma^j(s))(s') \cdot x_{s'}^i;$ 
20      $\mathbf{x} := \mathbf{y}; \mathbf{x}^1 := \mathbf{y}^1; \dots; \mathbf{x}^n := \mathbf{y}^n;$ 
21 foreach  $i \in \{1, \dots, n\}$  do  $q_i := y_s^i;$ 
22  $\sigma$  behaves as  $\sigma^j$  in  $j$ -th step when  $j < \max_{i \in \{1, \dots, n\}} k_i$ , and as  $\sigma^\infty$  afterwards;
23 return  $\sigma, \mathbf{q}$ 

```

Alg. 2. Value iteration-based algorithm for lines 4-5 of Alg. 1.

1 and 3; the latter in phases 2 and 3. Consider first the optimisation of $\mathbf{w} \cdot \boldsymbol{\rho}[\mathbf{k}]$. The values, for all states $s \in S$, are computed as a sequence of increasingly precise approximations, stored in a pair of vectors, \mathbf{x} and \mathbf{y} . Each new approximation is stored in \mathbf{y} (line 5); then, \mathbf{x} and \mathbf{y} are compared for convergence and \mathbf{x} is set to \mathbf{y} (line 7) before proceeding to the next iteration. Computation of the bounded part of $\mathbf{w} \cdot \boldsymbol{\rho}[\mathbf{k}]$ continues in phase 3 in similar fashion (although no convergence check is needed). During optimisation of $\mathbf{w} \cdot \boldsymbol{\rho}[\mathbf{k}]$, a corresponding optimal adversary is also determined, with the unbounded and bounded fragments stored in σ^∞ and σ^j , respectively. The choices made by this adversary are used to compute the value q_i for each of the n individual objectives, the values for which are also stored in pairs of vectors $(\mathbf{x}^i, \mathbf{y}^i)$ for each q_i .

In practice, storing multiple $|S|$ -sized vectors $(\mathbf{x}, \mathbf{y}, \mathbf{x}^i, \text{ and } \mathbf{y}^i)$ is relatively expensive. We discuss later how the algorithm's memory usage can be improved. We include an example of the execution of Alg. 2 in [12].

Numerical queries. We now turn our attention to numerical queries. Alg. 3 shows how Alg. 1 can be adapted to check these. Like Alg. 1, it generates points \mathbf{q} on the Pareto curve from a sequence of weight functions \mathbf{w} . For the objective ρ_1 that is being optimised, we generate a sequence of lower bounds r_1 that are

Input: MDP \mathcal{M} , objective $[\rho_1]_{\max}^{\leq k_1}$, predicate $([\rho_2]_{\geq r_2}^{\leq k_2}, \dots, [\rho_n]_{\geq r_n}^{\leq k_n})$
Output: Value of $num([\rho_1]_{\max}^{\leq k_1}, ([\rho_2]_{\geq r_2}^{\leq k_2}, \dots, [\rho_n]_{\geq r_n}^{\leq k_n}))$

- 1 $X := \emptyset; \boldsymbol{\rho} := (\rho_1, \dots, \rho_n); \mathbf{k} := (k_1, \dots, k_n); \mathbf{r} := (\min_{\sigma \in Adv_M} ExpTot_{\mathcal{M}}^{\sigma, k_1}(\rho_1), r_2, \dots, r_n);$
- 2 **do**
- 3 Find \mathbf{w} separating \mathbf{r} from $down(X)$ such that $w_1 > 0$;
- 4 Find adversary σ maximising $ExpTot_{\mathcal{M}}^{\sigma, \mathbf{k}}(\mathbf{w} \cdot \boldsymbol{\rho})$;
- 5 $\mathbf{q} := (ExpTot_{\mathcal{M}}^{\sigma, k_i}(\rho_i))_{1 \leq i \leq n}$;
- 6 **if** $\mathbf{w} \cdot \mathbf{q} < \mathbf{w} \cdot \mathbf{r}$ **then return** \perp ;
- 7 $X := X \cup \{\mathbf{q}\}; r_1 := \max\{r_1, \max\{r' \mid (r', r_2, \dots, r_n) \in down(X)\}\}$;
- 8 **while** $\mathbf{r} \notin down(X)$ or $\mathbf{w} \cdot \mathbf{q} > \mathbf{w} \cdot \mathbf{r}$;
- 9 **return** r_1 ;

Alg. 3. Algorithm for checking numerical queries

used in the same fashion as Alg. 1. Initially, we take r_1 to be the minimum possible value for ρ_1 , which can be computed with a separate instance of value iteration. New (non-decreasing) values for r_1 are generated at each step based on the set of points X determined so far. The numerical computation for each step (lines 4-5 of Alg. 3) can again be carried out with Alg. 2. Correctness of Alg. 3 is proved in [12]. The bound on the number of steps needed is as for Alg. 1.

Example 3. We demonstrate Alg. 3 on the MDP from Example 1 and numerical query $([\{s_1\}]_{\max}, ([\{s_3\}]_{\geq 0.7}))$. Initially, $r_1=0.1$ and, with $\mathbf{w}=(0.1, 0.8)$, we get $\mathbf{q}=(0.1, 0.9)$. The resulting area $down(X)$ is shown as dark grey in Fig. 2(c). Next, r_1 remains as 0.1 and, with $\mathbf{w}=(1, 0)$, we get $\mathbf{q}=(0.4, 0.6)$. Adding this to X , $down(X)$ is enlarged by the light grey area. Finally, r_1 is set to 0.3, choosing $\mathbf{w}=(0.5, 0.5)$ yields $\mathbf{q}=(0.4, 0.6)$ again, and the loop ends. Fig. 2(c) also shows the points \mathbf{q} and \mathbf{r} (as + and ■). The final value returned is $r_1=0.3$.

Pareto curves. Next, we discuss Pareto queries. Generating and visualising Pareto curves (or their approximations) provides a much clearer view of the trade-offs between objectives. Our algorithm is implemented as a simple modification of our previous algorithms, and is presented as Alg. 4. For simplicity, we focus on the 2-objective case, which is most practical for visualisation. Our implementation, described later, also supports the 3-objective case and, in theory, this can be extended to an arbitrary number of objectives.

Alg. 4, like the earlier ones, builds a set X of points on a Pareto curve P using weights \mathbf{w} . Since P is convex, the surface of points X represents a *lower* approximation of P . Our algorithm also constructs an *upper* approximation Y using the generated weights \mathbf{w} . As illustrated in Example 2, for each point $\mathbf{q} \in X$, there is a corresponding hyperplane passing through \mathbf{q} and with orientation \mathbf{w} , above which no values are achievable. Hence these represent upper bounds on P and we store, in Y , any weight \mathbf{w} that resulted in each point $\mathbf{q} \in X$.

The sequence of weights \mathbf{w} is generated as follows. We construct an initial curve using weights $(1, 0)$ and $(0, 1)$. Then, we repeatedly: (i) sort the points in X ; (ii) for each successive pair $\mathbf{x}^i, \mathbf{x}^{i+1}$ in X , find the lowest point \mathbf{p} on the intersection of the hyperplanes stored in Y for \mathbf{x}^i and \mathbf{x}^{i+1} ; (iii) choose \mathbf{w} as a separating hyperplane between $down(X)$ and \mathbf{p} . The algorithm continues

<p>Input: MDP \mathcal{M}, reward structures $\rho = (\rho_1, \rho_2)$, time bounds (k_1, k_2), $\varepsilon_p \in \mathbb{R}_{>0}$</p> <p>Output: An ε_p-approximation of a Pareto curve</p> <pre style="margin: 0;"> 1 $X := \emptyset$; $Y : \mathbb{R}^2 \rightarrow 2^{\mathbb{R}^2}$, initially $Y(x) = \emptyset$ for all x; $\mathbf{w} = (1, 0)$; 2 Find adversary σ maximising $ExpTot_{\mathcal{M}}^{\sigma, k}(\mathbf{w} \cdot \rho)$; 3 $\mathbf{q} := (ExpTot_{\mathcal{M}}^{\sigma, k_1}(\rho_1), ExpTot_{\mathcal{M}}^{\sigma, k_2}(\rho_2))$; 4 $X := X \cup \{\mathbf{q}\}$; $Y(\mathbf{q}) := Y(\mathbf{q}) \cup \{\mathbf{w}\}$; $\mathbf{w} = (0, 1)$; 5 do 6 Find adversary σ maximising $ExpTot_{\mathcal{M}}^{\sigma, k}(\mathbf{w} \cdot \rho)$; 7 $\mathbf{q} := (ExpTot_{\mathcal{M}}^{\sigma, k_1}(\rho_1), ExpTot_{\mathcal{M}}^{\sigma, k_2}(\rho_2))$; 8 $X := X \cup \{\mathbf{q}\}$; $Y(\mathbf{q}) := Y(\mathbf{q}) \cup \{\mathbf{w}\}$; $\mathbf{w} = \perp$; 9 Order X to a sequence $\mathbf{x}^1, \dots, \mathbf{x}^m$ such that $\forall i: x_1^i \leq x_1^{i+1}$ and $x_2^i \geq x_2^{i+1}$; 10 foreach $i \in \{1, \dots, m-1\}$ do 11 Let \mathbf{u} be the element of $Y(\mathbf{x}^i)$ with maximal u_1; 12 Let \mathbf{u}' be the element of $Y(\mathbf{x}^{i+1})$ with minimal u'_1; 13 Find a point \mathbf{p} such that $\mathbf{u} \cdot \mathbf{p} = \mathbf{u} \cdot \mathbf{x}^i$ and $\mathbf{u}' \cdot \mathbf{p} = \mathbf{u}' \cdot \mathbf{x}^{i+1}$; 14 if distance of \mathbf{p} from $down(X)$ is $\geq \varepsilon_p$ then 15 Find \mathbf{w} separating $down(X)$ from \mathbf{p}, maximising $\mathbf{w} \cdot \mathbf{p} - \max_{\mathbf{x} \in down(X)} \mathbf{w} \cdot \mathbf{x}$; 16 while $\mathbf{w} \neq \perp$; 17 return X </pre>
--

Alg. 4. Algorithm for Pareto curve approximation for 2 objectives

until the maximum distance between the two approximations falls below some threshold ε_p . In principal, the algorithm can enumerate all faces of P . The reason for constructing an ε_p -approximation is two-fold: firstly, the number of faces is potentially large, whereas an approximation may suffice; secondly, computation of individual points (using value iteration), is already approximate.

Example 4. We illustrate Alg. 4 on the MDP from Example 1 with objectives $(\{s_1\}_{\max}, \{s_3\}_{\max})$. The first two weight vectors \mathbf{w} are $(1, 0)$ and $(0, 1)$, yielding points \mathbf{q} of $(0.4, 0.6)$ and $(0.1, 0.9)$, respectively (see Fig. 2(d)). The hyperplanes attached to each point are also shown, by dotted lines, as is their intersection $\mathbf{p} = (0.4, 0.9)$. We choose separating hyperplane $\mathbf{w} = (0.5, 0.5)$, indicated by the sloped dotted line. The algorithm then finds the intersection $(0.1, 0.9)$ of this with the horizontal line and, since this point is already in $down(X)$, terminates.

Adversary generation. Finally, we describe how to generate *optimal adversaries* for our multi-objective queries. We explain this for achievability queries, but it can easily be adapted to the other types too. Unlike standard (single-objective) MDP model checking, where deterministic adversaries always suffice to optimise reachability/reward objectives, multi-objective optimisation requires randomised adversaries. Alg. 1, when finding that bounds \mathbf{r} are achievable, generates points $\mathbf{q}^1, \dots, \mathbf{q}^m$ on the Pareto curve. Each corresponding call to Alg. 2 returns a (deterministic) adversary, say $\sigma_{\mathbf{q}^j}$ for the current point \mathbf{q}^j . The final adversary σ_{opt} is constructed from these and a weight vector $\mathbf{u} \in \mathbb{R}^m$ satisfying $r_i \leq \sum_{j=1}^m u_j \cdot q_i^j$ for all $1 \leq i \leq n$: it simply makes an initial one-off random choice of adversary $\sigma_{\mathbf{q}^j}$ to mimic (each with probability u_j).

5 Implementation and Results

We implemented our multi-objective model checking techniques in PRISM [14], also adding the automaton construction of [11] to support ω -regular properties. Value iteration is built on top of PRISM’s “sparse” engine. It would also be straightforward to adapt its symbolic (MTBDD) engine, which can improve scalability on models exhibiting regularity; but, for the current set of experiments, the sparse engine suffices to illustrate the benefits offered by our approach.

Heuristics and optimisations. Our core algorithms are based on generating weight vectors \mathbf{w} representing separating hyperplanes (e.g. at line 3 of Alg. 1). The choice of each \mathbf{w} is not unique and affects the number of steps needed by the algorithm. Based on our results, the following is an effective heuristic. For the first n vectors (assuming n objectives), choose \mathbf{w} with $w_i = 1$ for some i . Next, given point \mathbf{r} and set of points X , choose \mathbf{w} to maximise $\min_{\mathbf{x} \in X} (\mathbf{w} \cdot \mathbf{q} - \mathbf{w} \cdot \mathbf{x})$, i.e., pick the hyperplane with maximal Euclidean distance d from \mathbf{q} . This is done by solving the LP: “maximise d subject to $\sum_{i=1}^n w_i = 1$ and $w_i \cdot (q_i - x_i) \geq d$ for all $\mathbf{x} \in X$ ”. In practice, these problems are small and fast to solve.

We also apply various optimisations to the basic value iteration algorithms of Section 4. For unbounded properties, Gauss-Seidel value iteration [19] can be used to increase performance. Furthermore, we can significantly reduce the number of vectors stored with slight changes to Alg. 2; details are in [12].

Experimental results. We evaluated our techniques on benchmarks from several sources.² First, we used multi-objective problems resulting from the assume-guarantee framework of [15]. Second, we verified multi-objective properties on existing PRISM models: (i) a task-graph *scheduler* problem, minimising expected job completion time and expected energy consumption; (ii) a *team-formation* protocol, maximising the probability of completing two (separate) tasks and the expected size of the team that does so; (iii) a dynamic power management (*dpm*) controller, minimising over k steps both expected energy consumption and expected average queue size. Experiments were run on a 2.66GHz PC with 8GB of RAM. We used $\varepsilon = 10^{-6}$ for value iteration (this is the default in PRISM; smaller values led to very similar results) and $\varepsilon_p = 10^{-4}$ for Pareto curve generation.

The results are shown in Table 1: assume-guarantee problems at the top; the others below. For each model, we give the size (number of states), and details of the objectives in the query used. The middle part of the table compares the performance of our value iteration-based technique with the LP-based implementation of [11] on numerical queries. In our experiments, performance for achievability queries was very similar, so we omit them. The right part of the table shows times to compute Pareto curves for the same objectives on each model (which cannot be done with the implementation of [11]). For value iteration-based algorithms, we show the number of points (steps of the algorithm) needed; for LP-based, we show the size of the linear program solved.

Comparing the value-iteration and LP-based approaches, we see huge gains in run-time for our methods (up to approx. 150 times faster). There are also

² All models/properties used are at www.prismmodelchecker.org/files/atva12mo/.

Case study [parameters]	Num. states	Objectives		Numerical query				Pareto query			
		Num.	Types	LP ([11])		Val. iter.		Val. iter.			
				LP size	Time (s)	Pt.s	Time (s)	Pt.s	Time (s)		
<i>consensus</i> (2 proc.s) [R K]	3 2	691	2	$[T_1]_{\max}$	1026	0.57	3	0.02	3	0.04	
	4 2	1517		$[T_2]_{\max}$	2288	0.67	3	0.03	3	0.05	
	5 2	3169			4812	0.94	3	0.05	3	0.06	
<i>consensus</i> (3 proc.s) [R K]	3 2	17455	2	$[T_1]_{\max}$	40386	9.85	3	0.22	3	0.27	
	4 2	61017		$[T_2]_{\max}$	140676	144.06	3	0.87	3	1.06	
	5 2	181129			<i>mem-out</i>			3	2.83	3	3.44
<i>zeroconf</i> [K]	4	5449	2	$[T_1]_{\max}$	12916	1.25	2	0.13	4	0.60	
	6	10543		$[T_2]_{\max}$	24639	7.07	4	0.46	4	0.79	
	8	17221			40833	19.6	4	0.76	4	1.13	
<i>zeroconf-tb</i> [K T]	2 14	29572	2	$[T_1]_{\max}$	61816	5.25	3	1.69	2	0.85	
	4 10	19670		$[T_2]_{\max}$	46659	5.01	2	0.32	3	0.84	
	4 14	42968			103964	11.01	2	0.63	3	1.77	
<i>team-form.</i> [N]	3	12475	2	$[T_1]_{\max}$	14935	1.37	4	0.21	7	0.24	
	4	96665		$[\rho_2]_{\max}$	115289	11.57	4	1.08	7	1.72	
	5	907993			<i>mem-out</i>			2	5.66	6	12.66
<i>team-form.</i> [N]	3	12475	3	$[T_1]_{\max}$	14935	1.37	3	0.18	57	1.39	
	4	96665		$[T_2]_{\max}$	115289	10.55	5	1.77	61	14.55	
	5	907993		$[\rho_2]_{\max}$	<i>mem-out</i>			2	9.49	57	141.76
<i>scheduler</i> [K]	5	31965	2	$[\rho_1]_{\min}$	57954	59.15	8	6.10	10	8.08	
	25	633735		$[\rho_2]_{\min}$	<i>mem-out</i>			8	526.56	11	776.44
	50	2457510			<i>mem-out</i>			8	3938.94	10	5361.86
<i>dpm</i> [k]	100	636	2	$[\rho_1]_{\min}^{\leq k}$	n/a	n/a	3	4.50	6	0.12	
	200	636		$[\rho_2]_{\min}^{\leq k}$	n/a	n/a	3	4.30	11	0.32	
	300	636			n/a	n/a	3	4.59	9	0.36	

Table 1. Experimental results for our implementation and a comparison with [11].

significant improvements in scalability: the biggest models solved with value iteration are about 20 times bigger than those for LP. One factor in the low run-times for our technique is that the algorithms generally require a fairly small number of steps, even when generating the Pareto curve.

Pareto curves. Finally, we show in Fig. 3 the Pareto curves generated for some of our examples. Plot (a) shows, for a task-graph scheduling problem, how different schedulers vary in terms of completion time and energy usage. Plot (b) is from an instance of assume-guarantee verification; the plot shows how it is possible to bound the probability of an error in the overall system (y-axis) for various different reliability levels of one of the components (x-axis). Plot (c) shows a 3-objective Pareto curve evaluating strategies in a team-formation protocol (see Fig. 3 caption for objectives). In each case, the plots give a clear, visual illustration of the trade-off between competing objectives. The curves could also be used to quickly answer any additional achievability or numerical queries for those objectives, without running any further model checking.

6 Conclusions

We have presented novel techniques for multi-objective model checking of MDPs, using a value iteration-based computation to build successive approximations of the Pareto curve. Compared to existing approaches, this gives significant gains in efficiency and scalability, and enables verification of time-bounded properties. Furthermore, we showed the benefits of visualising the Pareto curve for several probabilistic model checking case studies. Future directions include extending our techniques to timed probabilistic models such as CTMDPs and PTAs.

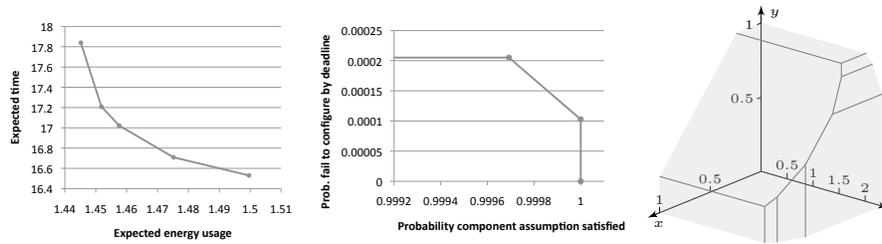


Fig. 3. Pareto curves from: (a) task-graph scheduler, $K=2$; (b) Zeroconf protocol, $K=2, T=10$; (c) team formation protocol, $N=3$ (axes $x/y/z$ = Probability of completing task 1/probability of completing task 2/expected size of successful team)

Acknowledgments. The authors are part supported by ERC Advanced Grant VERIWARE and EPSRC grant EP/F001096/1. Vojtěch Forejt is also supported by a Royal Society Newton Fellowship.

References

1. Altman, E.: Constrained Markov Decision Processes. Chapman & Hall/CRC (1999)
2. Bleuler, S., Laumanns, M., Thiele, L., Zitzler, E.: PISA-A Platform and Programming Language Independent Interface for Search Algorithms. In: EMO'03 (2003)
3. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge Univ. Press (2004)
4. Brázdil, T., Brožek, V., Chatterjee, K., Forejt, V., Kučera, A.: Two views on multiple mean-payoff objectives in Markov decision processes. In: LICS'11 (2011)
5. Chatterjee, K., Majumdar, R., Henzinger, T.: Markov decision processes with multiple objectives. In: Proc. STACS'06. pp. 325–336. Springer (2006)
6. Clímaco, J. (ed.): Multicriteria Analysis. Springer (1997)
7. Coello, C., Lamont, G., van Veldhuizen, D.: Evolutionary Algorithms for Solving Multi-Objective Problems. Springer (2007)
8. Diakonikolas, I., Yannakakis, M.: Succinct approximate convex Pareto curves. In: Proc. SODA'08. pp. 74–83. SIAM (2008)
9. Etesami, K., Kwiatkowska, M., Vardi, M., Yannakakis, M.: Multi-objective model checking of Markov decision processes. LMCS 4(4), 1–21 (2008)
10. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: SFM'11. LNCS, vol. 6659. Springer (2011)
11. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Quantitative multi-objective verification for probabilistic systems. In: Proc. TACAS'11 (2011)
12. Forejt, V., Kwiatkowska, M., Parker, D.: <http://arxiv.org/abs/1206.6295>
13. Kemeny, J., Snell, J., Knapp, A.: Denumerable Markov Chains. Springer (1976)
14. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Proc. CAV'11. LNCS, vol. 6806, pp. 585–591. Springer (2011)
15. Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Assume-guarantee verification for probabilistic systems. In: Proc. TACAS'10. pp. 23–37. Springer (2010)
16. Legriel, J., Cotton, S., Maler, O.: On universal search strategies for multi-criteria optimization using weighted sums. In: Proc. CEC'11. pp. 2351–2358 (2011)
17. Legriel, J., Guernic, C.L., Cotton, S., Maler, O.: Approximating the Pareto front of multi-criteria optimization problems. In: Proc. TACAS'10. pp. 69–83 (2010)
18. Papadimitriou, C., Yannakakis, M.: On the approximability of trade-offs and optimal access of web sources. In: Proc. FOCS'00. pp. 86–92 (2000)
19. Puterman, M.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley and Sons (1994)