A Practitioner's Guide to MDP Model Checking Algorithms*

Arnd Hartmanns[1], Sebastian Junges[2], Tim Quatmann[3], and Maximilian Weininger[4](✉)

[1] University of Twente, Enschede, The Netherlands a.hartmanns@utwente.nl
[2] Radboud University, Nijmegen, The Netherlands sebastian.junges@ru.nl
[3] RWTH Aachen University, Aachen, Germany tim.quatmann@cs.rwth-aachen.de
[4] Technical University of Munich, Munich, Germany maxi.weininger@tum.de

Abstract. Model checking undiscounted reachability and expected-reward properties on Markov decision processes (MDPs) is key for the verification of systems that act under uncertainty. Popular algorithms are policy iteration and variants of value iteration; in tool competitions, most participants rely on the latter. These algorithms generally need worst-case exponential time. However, the problem can equally be formulated as a linear program, solvable in polynomial time. In this paper, we give a detailed overview of today's state-of-the-art algorithms for MDP model checking with a focus on performance and correctness. We highlight their fundamental differences, and describe various optimizations and implementation variants. We experimentally compare floating-point and exact-arithmetic implementations of all algorithms on three benchmark sets using two probabilistic model checkers. Our results show that (optimistic) value iteration is a sensible default, but other algorithms are preferable in specific settings. This paper thereby provides a guide for MDP verification practitioners—tool builders and users alike.

1 Introduction

The verification of MDPs is crucial for the design and evaluation of cyber-physical systems with sensor noise, biological and chemical processes, network protocols, and many other complex systems. MDPs are the standard model for sequential decision making under uncertainty and thus at the heart of reinforcement learning. Many dependability evaluation and safety assurance approaches rely in some form on the verification of MDPs with respect to temporal logic properties. Probabilistic model checking [4,5] provides powerful tools to support this task.

The essential MDP model checking queries are for the *worst-case probability that something bad happens* (reachability) and the *expected resource consumption until task completion* (expected rewards). These are *indefinite (undiscounted)*

* This research was funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233 (MISSION), and by NWO VENI grant no. 639.021.754.

horizon queries: They ask about the probability or expectation of a random variable up until an event—which forms the horizon—but are themselves unbounded. Many more complex properties internally reduce to solving either reachability or expected rewards. For example, if the description of *something bad* is in linear temporal logic (LTL), then a product construction with a suitable automaton reduces the LTL query to reachability [6]. This paper sets out to determine the practically best algorithms to solve indefinite horizon reachability probabilities and expected rewards; our methodology is an empirical evaluation.

MDP analysis is well studied in many fields and has lead to three main types of algorithms: *value iteration* (VI), *policy iteration* (PI), and *linear programming* (LP) [55]. While indefinite horizon queries are natural in a verification context, they differ from the standard problem of e.g. operations research, planning, and reinforcement learning. In those fields, the primary concern is to *compute a policy* that (often approximately) optimizes the *discounted* expected reward over an infinite horizon where rewards accumulated in the future are weighted by a discount factor < 1 that exponentially prefers values accumulated earlier.

The lack of discounting in verification has vast implications. The *Bellman operation*, essentially describing a one-step backward update on expected rewards, is a contraction with discounting, but not a contraction without. This leads to significantly more complex termination criteria for VI-based verification approaches [34]. Indeed, VI runs in polynomial time for every fixed discount factor [49], and similar results are known for PI as well as LP solving with the simplex algorithm [60]. In contrast, VI [9] and PI [20] are known to have exponential worst-case behaviour in the undiscounted case.

So, *what is the best algorithm for model checking MDPs?* A polynomial-time algorithm exists using an LP formulation and barrier methods for its solution [12]. LP-based approaches (and their extension to MILPs) are also prominent for multi-objective model checking [21], in counterexample generation [23], and for the analysis of parametric Markov chains [16]. However, folklore tells us that iterative methods, in particular VI, are better for solving MDPs. Indeed, variations of VI are the default choice of all model checkers participating in the QComp competition [14]. This uniformity may be misleading. Indeed, for some stochastic game algorithms, using LP to solve the underlying MDPs may be preferential [3, Appendix E.4]. An application in runtime assurance preferred PI for numerical stability [45, Sect. 6]. A toy example from [34] is a famous challenge for VI-based methods. Despite the prominence of LP, the ease of encoding MDPs, and the availability of powerful off-the-shelf LP solvers, many tools did (until very recently) not include MDP model checking via LP solvers.

With this paper, we reconsider the PI and LP algorithms to investigate whether probabilistic model checking focused on the wrong family of algorithms. We report the results of an extensive empirical study with two independent implementations in the model checkers Storm [42] and mcsta [37]. We find that, in terms of performance and scalability, optimistic value iteration [40] is a solid choice on the standard benchmark collection (which goes beyond competition benchmarks) but can be beat quite considerably on challenging cases. We also emphasize the question of precision and soundness. Numerical algorithms, in particular ones that converge *in the limit*, are prone to delivering wrong results. For VI, the recognition of this problem has led to a series of improvements over the last decade [8,34,40,19,54,56]. We show that PI faces a similar problem. When using floating-point arithmetic, additional issues may arise [36,59]. Our use of various LP solvers exhibits concerning results for a variety of benchmarks. We therefore also include results for *exact* computation using rational arithmetic.

Limitations of this study. A thorough experimental study of algorithms requires a carefully scoped evaluation. We work with flat representations of MDPs that fit completely into memory (i.e. we ignore the state space exploration process and symbolic methods). We selected algorithms that are tailored to converge to *the* optimal value. We also exclude approaches that incrementally build and solve (partial or abstract) MDPs using simulation or model checking results to guide exploration: they are an orthogonal improvement and would equally profit from faster algorithms to solve the partial MDPs. Moreover, this study is on algorithms, not on their implementations. To reduce the impact of potential implementation flaws, we use two independent tools where possible. Our experiments ran on a single type of machine—we do not study the effect of different hardware.

Contributions. This paper contributes a thorough overview on how to model-check indefinite horizon properties on MDPs, making MDP model checking more accessible, but also pushing the state-of-the-art by clarifying open questions. Our study is built upon a thorough empirical evaluation using two independent code bases, sources benchmarks from the standard benchmark suite and recent publications, compares 10 LP solvers, and studies the influence of various prominent preprocessing techniques. The paper provides new insights and reviews folklore statements: Particular highlights are a new simple but challenging MDP family that leads to wrong results on all floating-point LP solvers (Section 2.3), a negative result regarding the soundness of PI with epsilon-precise policy evaluators (Section 4), and an evaluation on numerically challenging benchmarks that shows the limitations of value iteration in a practical setting (Section 5.3).

2 Background

We recall MDPs with reachability and reward objectives, describe solution algorithms and their guarantees, and address commonly used optimizations.

2.1 Markov Decision Processes

Let $\mathsf{D}_X \coloneqq \{\, \mathsf{d}\colon X \to [0,1] \mid \sum_{x\in X} \mathsf{d}(x) = 1 \,\}$ be the set of distributions over X. A Markov decision process (MDP) [55] is a tuple $\mathcal{M} = (\mathsf{S}, \mathsf{A}, \delta)$ with finite sets of states S and actions A, and a partially defined transition function $\delta\colon \mathsf{S} \times \mathsf{A} \rightharpoonup \mathsf{D}_\mathsf{S}$ such that $\mathsf{A}(s) \coloneqq \{\, a \mid (s,a) \in \mathit{domain}(\delta) \,\} \neq \emptyset$ for all $s \in \mathsf{S}$. $\mathsf{A}(s)$ is the set of enabled actions at state s. δ maps enabled state-action pairs to distributions over successor states. A Markov chain (MC) is an MDP with $|\mathsf{A}(s)| = 1$ for all s. The *semantics* of an MDP are defined in the usual way, see, e.g. [6, Chapter 10]. A (memoryless deterministic) policy—a.k.a. strategy or scheduler—is a function $\pi\colon \mathsf{S} \to \mathsf{A}$ that, intuitively, given the current state s prescribes what action $a \in \mathsf{A}(s)$ to play. Applying a policy π to an MDP induces an MC $\mathcal{M}^\pi$. A path in this MC is an infinite sequence $\rho = s_1 s_2 \dots$ with $\delta(s_i, \pi(s_i))(s_{i+1}) > 0$. Paths denotes the set of all paths and $\mathbb{P}^\pi_s$ denotes the unique probability measure of $\mathcal{M}^\pi$ over infinite paths starting in the state s.

A *reachability objective* $\mathrm{P}_{\mathsf{opt}}(\mathsf{T})$ with set of target states $\mathsf{T} \subseteq \mathsf{S}$ and $\mathsf{opt} \in \{\max, \min\}$ induces a random variable $X\colon \mathsf{Paths} \to [0, 1]$ over paths by assigning 1 to all paths that eventually reach the target and 0 to all others. $\mathrm{E}_{\mathsf{opt}}(\mathsf{rew})$ denotes an *expected reward objective*, where $\mathsf{rew}\colon \mathsf{S} \to \mathbb{Q}_{\geq 0}$ assigns a reward to each state. $\mathsf{rew}(\rho) := \sum_{i=1}^{\infty} \mathsf{rew}(s_i)$ is the accumulated reward of a path $\rho = s_1 s_2 \dots$. This yields a random variable $X\colon \mathsf{Paths} \to \mathbb{Q} \cup \{\infty\}$ that maps paths to their reward. For a given objective and its random variable X, the *value of a state* $s \in \mathsf{S}$ is the expectation of X under the probability measure $\mathbb{P}^\pi_s$ of the the MC induced by an optimal policy π from the set of all policies Π, formally $\mathsf{V}(s) := \mathsf{opt}_{\pi \in \Pi} \mathbb{E}^\pi_s[X]$.

2.2 Solution Algorithms

Value iteration (VI), e.g. [15], computes a sequence of value vectors converging to the optimum in the limit. In all variants of the algorithm, we start with a function $x\colon\ \mathsf{S} \to \mathbb{Q}$ that assigns to every state an estimate of the value. The algorithm repeatedly performs an update operation to improve the estimates. After some preprocessing, this operation has a unique fixpoint when $x = \mathsf{V}$. Thus, value iteration converges to the value in the limit. Variants of VI include interval iteration [34], sound VI [56] and optimistic VI [40]. We do not discuss these in detail, but instead refer to the respective papers.

Linear programming (LP), e.g. [6, Chapter 10], encodes the transition structure of the MDP and the objective as a linear optimization problem. For every state, the LP has a variable representing an estimate of its value. Every state-action pair is encoded as a constraint on these variables, as are the target set or rewards. The unique optimum of the LP is attained if and only if for every state its corresponding variable is set to the value of the state. We provide an in-depth discussion of theoretical and practical aspects of LP in Section 3.

Policy iteration (PI), e.g. [11, Section 4], computes a sequence of policies. Starting with an initial policy, we evaluate its induced MC, improve the policy by switching suboptimal choices and repeat the process on the new policy. As every policy improves the previous one and there are only finitely many memoryless deterministic policies (a number exponential in the number of states), eventually we obtain an optimal policy. We further discuss PI in Section 4.

2.3 Guarantees

Given the stakes in many application domains, we require guarantees about the relation between an algorithm's result $\bar{v}$ and the true value v. First, implementations are subject to floating-point errors and imprecision [59] unless they use exact (rational) arithmetic or safe rounding [36]. This can result in arbitrary

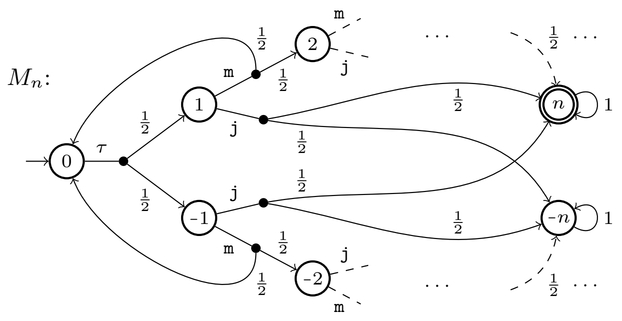


Fig. 1: A hard MDP for all algorithms

Table 1: Correct results

alg.	solver	$n \leq$
PI	–	20
LP	COPT	18
	CPLEX	18
	Glop	25
	GLPK	24
	Gurobi	18
	HiGHS	22
	lp_solve	28
	Mosek	22
	SoPlex	34

differences between $\bar{v}$ and v. Second are the algorithm's inherent properties: VI is an approximating algorithm that converges to the true value only in the limit. In theory, it is possible to obtain the exact result by rounding after exponentially many iterations [15]; in practice, this results in excessive runtime. Instead, for years, implementations used a naive stopping criterion that could return arbitrarily wrong results [33]. This problem's discovery sparked the development of sound variants of VI [8,34,40,19,54,56], including interval iteration, sound value iteration, and optimistic value iteration. A sound VI algorithm guarantees ε-precise results, i.e. $|v - \bar{v}| \leq \varepsilon$ or $|v - \bar{v}| \leq v \cdot \varepsilon$. For LP and PI, the guarantees have not yet been thoroughly investigated. Theoretically, both are exact, but implementations are often not. We discuss the problems in Sections 3 and 4.

The handcrafted MC of [33, Figure 2] highlights the lack of guarantees of VI: standard implementations return vastly incorrect results. We extended it with action choices to obtain the MDP M_n shown in Fig. 1 for $n \in \mathbb{N}$, $n \geq 2$. It has $2n+1$ states; we compute $\mathrm{P}_{\min}(\{\, n \,\})$ and $\mathrm{P}_{\max}(\{\, n \,\})$. The policy that chooses action m wherever possible induces the MC of [33, Figure 2] with $(\mathrm{P}_{\min}(\{\, n \,\}), \mathrm{P}_{\max}(\{\, n \,\})) = (\frac{1}{2}, \frac{1}{2})$. In every state s with $0 < s < n$, we added the choice of action j that jumps to n and -n. With that, the (optimal) values over all policies are $(\frac{1}{3}, \frac{2}{3})$. In VI, starting from value 0 for all states except n, initially taking j everywhere looks like the best policy for $\mathrm{P}_{\max}$. As updated values slowly propagate, state-by-state, m becomes the optimal choice in all states except $-n+1$. We thus layered a "deceptive" decision problem on top of the slow convergence of the original MC. For $n = 20$, VI with Storm and mcsta deliver the incorrect results $(0.247, 0.500)$. For Storm's PI and various LP solvers, we show in Table 1 the largest n for which they return a $\pm\,0.01$-correct result. For larger n, PI and all LP solvers claim $\approx (\frac{1}{2}, \frac{1}{2})$ as the correct solution except for Glop and GLPK which only fail for the maximum at the given n; for the minimum, they return the wrong result at $n \geq 29$ and 52, respectively. Sound VI algorithms and Storm's exact-arithmetic engine produce (ε-)correct results, though the former at excessive runtime for larger n. We used default settings for all tools and solvers.

2.4 Optimizations

VI, LP, and PI can all benefit from the following optimizations:

Graph-theoretic algorithms can be used for qualitative analysis of the MDP, i.e. finding states with value 0 or (only for reachability objectives) 1. These qualitative approaches are typically a lot faster than the numerical computations for quantitative analysis. Thus, we always apply them first and only run the numerical algorithms on the remaining states with non-trivial values.

Topological methods, e.g. [17], do not consider the whole MDP at once. Instead, they first compute a topological ordering of the strongly connected components (SCCs)[5] and then analyze each SCC individually. This can improve the runtime, as we decompose the problem into smaller subproblems. The subproblems can be solved with any of the solution methods. Note that when considering acyclic MDPs, the topological approach does not need to call the solution methods, as the resulting values can immediately be backpropagated.

Collapsing of maximal end components (MECs), e.g., [13,34], transforms the MDP into one with equivalent values but simpler structure. After collapsing MECs, the MDP is contracting, i.e. we almost surely reach a target state or a state with value zero. VI algorithms rely on this property for convergence [34,40,56]. For PI and LP, simplifying the graph structure before applying the solution method can speed up the computation.

Warm starts, e.g. [26,46], may adequately initialize an algorithm, i.e., we may provide it with some prior knowledge so that the computation has a good starting point. We implement warm starts by first running VI for a limited number of iterations and using the resulting estimate to guess bounds on the variables in an LP or a good initial policy for PI. See Sections 3 and 4 for more details.

3 Practically solving MDPs using Linear Programs

This section considers the LP-based approach to solving the optimal policy problem in MDPs. To the best of our knowledge, this is the only polynomial-time approach. We discuss various configurations. These configuration are a combination of the LP formulation, the choice of software, and their parameterization.

3.1 How to encode MDPs as LPs?

For objective $\mathrm{P}_{\max}(\mathsf{T})$ we formulate the following LP over variables x_s, $s \in \mathsf{S} \setminus \mathsf{T}$:

$$\text{minimize} \quad \sum_{s \in \mathsf{S}} x_s \quad \text{s.t. } lb(s) \leq x_s \leq ub(s) \quad \text{and}$$

$$x_s \geq \sum_{s' \in \mathsf{S} \setminus \mathsf{T}} \delta(s,a)(s') \cdot x_{s'} + \sum_{t \in \mathsf{T}} \delta(s,a)(t) \quad \text{for all } s \in \mathsf{S} \setminus \mathsf{T}, a \in \mathsf{A}$$

[5] A set $\mathsf{S}' \subseteq \mathsf{S}$ is a connected component if for all $s, s' \in \mathsf{S}'$, s can be reached from s'. We call S' strongly connected component if it is inclusion maximal.

We assume bounds $lb(s) = 0$ and $ub(s) = 1$ for $s \in \mathsf{S} \setminus \mathsf{T}$. The unique solution $\eta\colon \{\, x_s \mid s \in \mathsf{S} \setminus \mathsf{T} \,\} \to [0, 1]$ to this LP coincides with the desired objective values $\eta(x_s) = V(s)$. Objectives $\mathrm{P}_{\min}(\mathsf{T})$ and $\mathrm{E}_{\mathsf{opt}}(\mathsf{rew})$ have similar encodings: minimizing policies require maximisation in the LP and flipping the constraint relation. Rewards can be added as an additive factor on the right-hand side. For practical purposes, the LP formulation can be tweaked.

The choice of bounds. Any bounds that respect the unique solution will not change the answer. That is, any lb and ub with $0 \le lb(s) \le V(s) \le ub(s)$ yield a sound encoding. While these additional bounds are superfluous, they may significantly prune the search space. We investigate trivial bounds, e.g., knowing that all probabilities are in $[0, 1]$, bounds from a structural analysis as discussed by [8], and bounds induced by a warm start of the solver. For the latter, if we have obtained values $V' \le V$, e.g., induced by a suboptimal policy, then $V'(s)$ is a lower bound on the value x_s, which is particularly relevant as the LP minimizes.

Equality for unique actions. Markov chains, i.e., MDPs where $|\mathsf{A}| = 1$, can be solved using linear equation systems. The LP encoding uses one-sided inequalities and the objective function to incorporate nondeterministic choices. We investigate adding constraints for all states with a unique action.

$$x_s \le \sum_{s' \in S \setminus T} \delta(s,a)(s') \cdot x_{s'} + \sum_{t \in T} \delta(s,a)(t) \qquad \text{for all } s \in S \setminus T \text{ with } \mathsf{A}(s) = \{a\}$$

These additional constraints may trigger different optimizations in a solver, e.g., some solvers use Gaussian elimination for variable elimination.

A simpler objective. The standard objective assures the solution η is optimal for *every* state, whereas most invocations require only optimality in some specific states – typically the initial state s_0 or the entry states of a strongly connected component. In that case, the objective may be simplified to optimize only the value for those states. This potentially allows for multiple optimal solutions: in terms of the MDP, it is no longer necessary to optimize the value for states that are not reached under the optimal policy.

Encoding the dual formulation. Encoding a dual formulation to the LP is interesting for mixed-integer extensions to the LP, relevant for computing, e.g., policies in POMDPs [47], or when computing minimal counterexamples [58]. For LPs, due to the strong duality, the internal representation in the solvers we investigated is (almost) equivalent and all solvers support both solving the primal and the dual representation. We therefore do not further consider constructing them.

3.2 How to solve LPs with existing solvers?

We rely on the performance of state-of-the-art LP solvers. Many solvers have been developed and are still actively advanced, see [2] for a recent comparison on general benchmarks. We list the LP solvers that we consider for this work in Table 2. The columns summarize for each solver the type of license, whether it uses exact or floating-point arithmetic, whether it supports multithreading,

Table 2: Available LP solvers ("intr" = interior point)

solver	version	license	exact/fp	parallel	algorithms	mcsta	Storm
COPT [24]	5.0.5	academic	fp	yes	intr + simplex	yes	no
CPLEX [44]	22.10	academic	fp	yes	intr + simplex	yes	no
Gurobi [32]	9.5	academic	fp	yes	intr + simplex	yes	yes
GLPK [29]	4.65	GPL	fp	no	intr + simplex	no	yes
Glop [30]	9.4.1874	Apache	fp	no	simplex only	yes	no
HiGHS [35,43]	1.2.2	MIT	fp	yes	intr + simplex	yes	no
lp_solve [10]	5.5.2.11	LGPL	fp	no	simplex only	yes	no
Mosek [52]	10.0	academic	fp	yes	intr + simplex	yes	no
SoPlex [28]	6.0.1	academic	both	no	simplex only	no	yes
Z3 [53]	4.8.13	MIT	exact	no	simplex only	no	yes

and what type of algorithms it implements. We also list whether the solver is available from the two model checkers used in this study[6].

Methods. We briefly explain the available methods and refer to [12] for a thorough treatment. Broadly speaking, the LP solvers use one out of two families of methods. *Simplex*-based methods rely on highly efficient pivot operations to consider vertices of the simplex of feasible solutions. Simplex can be executed either in the *primal* or *dual* fashion, which changes the direction of progress made by the algorithm. Our LP formulation has more constraints than variables, which generally means that the dual version is preferable. *Interior methods*, often the subclass of *barrier methods*, do not need to follow the set of vertices. These methods may achieve polynomial time worst-case behaviour. It is generally claimed that simplex has superior average-case performance but is highly sensitive to perturbations, while interior-point methods have a more robust performance.

Warm starts. LP-based model checking can be done using two types of warm starts. Either by providing a (feasible) basis point as done in [26] or by presenting bounds. The former, however, comes with various remarks and limitations, such as the requirement to disable preprocessing. We therefore used warm starts only by using bounds as discussed above.

Multithreading. We generally see two types of parallelisation in LP solvers. Some solvers support a *portfolio* approach that runs different approaches and finishes with the first one that yields a result. Other solvers parallelize the interior-point and/or simplex methods themselves.

Guarantees for numerical LP solvers. All LP solvers allow tweaking of various parameters, including *tolerances* to manage whether a point is considered feasible or optimal, respectively. The experiments in Table 1 already indicate that these guarantees are *not* absolute. A limited experiment indicated that reducing these tolerances towards zero did remove some incorrect results, but not all.

[6] Support for Gurobi, GLPK, and Z3 was already available in Storm. Support for Glop was already available in mcsta. All other solver interfaces have been added.

Exact solving. SoPlex supports exact computations, with a Boost library wrapping GMP rationals [22], after a floating-point arithmetic-based startup phase [27]. While this combination is beneficial for performance in most settings, it leads to crashes for the numerically challenging models. Z3 supports only exact arithmetic (also wrapping GMP numbers with their own interface). We observe that the price of converting large rational numbers may be substantial. SMT solvers like Z3 use a simplex variation [18] tailored towards finding feasible points and in an incremental fashion, optimized for problems with a nontrivial Boolean structure. In contrast, our LP formulation is easily feasible and is a pure conjunction.

4 Sound Policy Iteration

Starting with an initial policy, PI-based algorithms iteratively improve the policy based on the values obtained for the induced MC. The algorithm for solving the induced MC crucially affects the performance and accuracy of the overall approach. This section addresses the solvers available in Storm, possible precision issues, and how to utilize a warm start, while Section 5 discusses PI performance[7].

Markov chain solvers. To solve the induced MC, Storm can employ all linear equation solvers listed in [42] and all implemented variants of VI. In our experiments, we consider (i) the generalized minimal residual method (GMRES) [57] implemented in GMM++ [25], (ii) VI [15] with a standard (relative) termination criterion, (iii) optimistic VI (OVI) [40], and (iv) the sparse LU decomposition implemented in Eigen [31] using either floating-point or exact arithmetic (LU^{X}). LU and LU^{X} provide exact results (modulo floating-point errors in LU) while OVI yields ε-precise results. VI and GMRES do not provide any guarantees.

Correctness of PI. The accuracy of PI is affected by the MC solver. Firstly, PI cannot be more precise than its underlying solver: the result of PI has the same precision as the result obtained for the final MC. Secondly, inaccuracies by the solver can hide policy improvements; this may lead to premature convergence with a sub-optimal policy. We show that PI can return arbitrarily wrong results—*even if the intermediate results are ε-precise*:

Consider the MDP in Fig. 2 with objective $\mathrm{P}_{\max}(\{G\})$. There is only one nondeterministic choice, namely in state s_0. The optimal policy is to pick b, obtaining a value of 0.5. Picking a only yields 0.1. However, when starting from the initial policy $\pi(s_0) = \mathsf{a}$, an ε-precise MC solver may return $0.1 + \varepsilon$ for both s_0 and s_1 and $\delta/2 + (1-\delta)\cdot 0.1$ for s_2. This solution is indeed ε-precise. However, when evaluating which action to pick in s_0, we can choose δ such that a seems to obtain a higher value. Concretely, we require $\delta/2 + (1-\delta)\cdot 0.1 < 0.1 + \varepsilon$. For every $\varepsilon > 0$, this can be achieved by setting $\delta < 2.5 \cdot \varepsilon$. In this case, PI would terminate with the final policy inducing a severely suboptimal value.

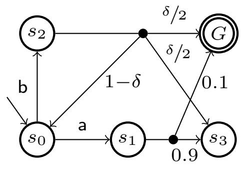


Fig. 2: Example MDP

[7] [46] addresses performance in the context of PI for stochastic games.

If every Markov chain is solved precisely, PI is correct. Indeed, it suffices to be certain that one action is better than all others. This is the essence of modified policy iteration as described in [55, Chapters 6.5 and 7.2.6]. Similarly, [46, Section 4.2] suggests to use interval iteration when solving the system induced by the current policy and stopping when the under-approximation of one action is higher than the over-approximation of all other actions.

Warm starts. PI profits from being provided a *good* initial policy. If the initial policy is already optimal, PI terminates after a single iteration. We can inform our choice of the initial policy by providing estimates for all states as computed by VI. For every state, we choose the action that is optimal according to the estimate. This is a good way to leverage VI's ability to quickly deliver good estimates [40], while at the same time providing the exactness guarantees of PI.

5 Experimental Evaluation

To understand the practical performance of the different algorithms, we performed an extensive experimental evaluation. We used three sets of benchmarks: all applicable benchmark instances[8] from the Quantitative Verification Benchmark Set (QVBS) [41] (the *qvbs* set), a subset of hard QVBS instances (the *hard* set), and numerically challenging models from a runtime monitoring application [45] (the *premise* set, named for the corresponding prototype). We consider two probabilistic model checkers, Storm [42] and the Modest Toolset's [37] mcsta. We used Intel Xeon Platinum 8160 systems running 64-bit CentOS Linux 7.9, allocating 4 CPU cores and 32 GB RAM to each experiment unless noted otherwise.

We plot algorithm runtimes in seconds in *quantile plots* as on the left and *scatter plots* as on the right of Fig. 3. The former compare multiple tools or configurations; for each, we sort the instances by runtime and plot the corresponding monotonically increasing line. Here, a point (x, y) on the a-line means that the x-th fastest instance solved by a took y seconds. The latter compare two tools or configurations. Each point (x, y) is for one benchmark instance: the x-axis tool took x while the y-axis tool took y seconds to solve it. The shape of points indicates the model type; the mapping from shapes to types is the same for all scatter plots and is only given explicitly in the first one in Fig. 3. Additional plots to support the claims in this section are provided in the appendix of the full version [39] of this paper.

The depicted runtimes are for the respective algorithm and all necessary and/or stated preprocessing, but do not include the time for constructing the MDP state spaces (which is independent of the algorithms). mcsta reports all time measurements rounded to multiples of 0.1 s. We summarize timeouts, out-of-memory, errors, and incorrect results as "n/a". Our timeout is 30 minutes for the algorithm and 45 minutes for total runtime including MDP construction. We consider a result $\bar{v}$ incorrect if $|v - \bar{v}| > v \cdot 10^{-3}$ (i.e. relative error 10^{-3}) whenever a reference result v is available. We however do not flag a result as incorrect if

[8] A *benchmark instance* is a combination of model, parameter valuation, and objective.

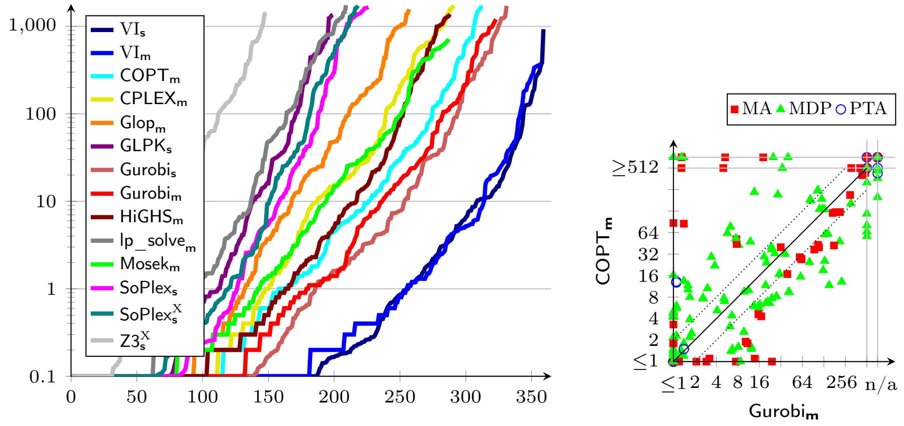


Fig. 3: Comparison of LP solver runtime on the *qvbs* set

v and $\bar{v}$ are both below 10^{-8} (relevant for the *premise* set). Nevertheless, we configure the (unsound) convergence threshold for VI as 10^{-6} relative; among the sound VI algorithms, we include OVI, with a (sound) stopping criterion of relative 10^{-6} error. To only achieve the 10^{-3} precision we actually test, OVI could thus be even faster than it appears in our plots. We make this difference to account for the fact that many algorithms, including the LP solvers, do not have a sound error criterion. We mark exact algorithms/solvers that use rational arithmetic with a superscript $^{\mathrm{X}}$. The other configurations use floating-point arithmetic (fp).

5.1 The QVBS Benchmarks

The *qvbs* set comprises all QVBS benchmark instances with an MDP, Markov automaton (MA), or probabilistic timed automaton (PTA) model[9] and a reachability or expected reward/time objective that is quantitative, i.e. not a query that yields a zero or one probability. We only consider instances where both Storm and mcsta can build the explicit representation of the MDP within 15 minutes. This yields 367 instances. We obtain reference results for 344 of them from either the QVBS database or by using one of Storm's exact methods. We found all reference results obtained via different methods to be consistent.

For LP, we have various solvers with various parameters each, cf. Section 3. For conciseness, we first compare all available LP solvers on the *qvbs* set. For the best-performing solver, we then evaluate the benefit of different solver configurations. We do the same for the choice of Markov chain solution method in PI. We then focus on these single, reasonable, setups for LP and PI each in more detail.

LP solver comparison. The left-hand plot of Fig. 3 summarizes the results of our comparison of the different LP solvers. Subscripts $_\mathsf{s}$ and $_\mathsf{m}$ indicate whether the solver is embedded in either Storm or mcsta. We apply no optimizations or

[9] MA and PTA are converted to MDP via embedding and digital clocks [48].

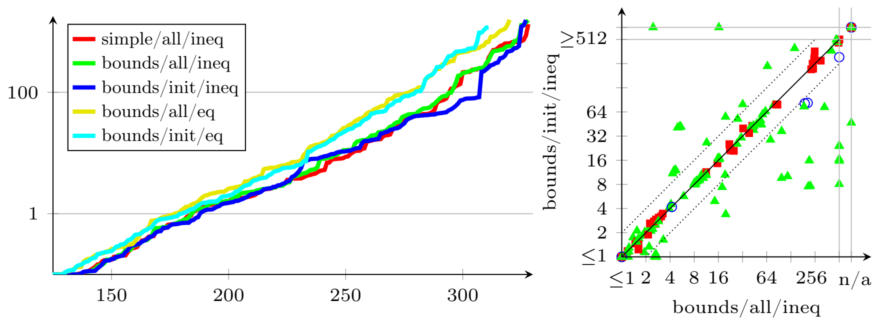


Fig. 4: Performance impact of LP problem formulation variants (using $\mathsf{Gurobi}_\mathsf{s}$)

reductions to the MDPs except for the precomputation of probability-0 states (and in Storm also of probability-1 states), and use the default settings for all solvers, with the trivial variable bounds $[0, 1]$ and $[0, \infty)$ for probabilities and expected rewards, respectively. We include VI as baseline. In Table 3, we summarize the results.

Table 3: LP summary

solver	correct	incorr.	no result
VI_s	359	8	0
VI_m	357	8	2
COPT_m	312	12	43
$\mathsf{CPLEX}_\mathsf{m}$	291	10	66
Glop_m	257	4	106
GLPK_s	199	5	163
$\mathsf{Gurobi}_\mathsf{s}$	331	4	32
$\mathsf{Gurobi}_\mathsf{m}$	323	4	40
$\mathsf{HiGHS}_\mathsf{m}$	288	10	69
$\mathsf{lp\_solve}_\mathsf{m}$	209	0	158
$\mathsf{Mosek}_\mathsf{m}$	287	15	65
$\mathsf{SoPlex}_\mathsf{s}$	226	9	132
$\mathsf{SoPlex}_\mathsf{s}^\mathrm{X}$	218	0	149
$\mathsf{Z3}_\mathsf{s}^\mathrm{X}$	148	0	219

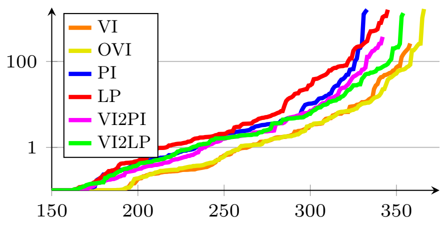
In terms of **performance** and scalability, Gurobi solves the highest number of benchmarks in any given time budget, closely followed by COPT. CPLEX, HiGHS, and Mosek make up a middle-class group. While the exact solver Z3 is very slow, SoPlex's exact mode actually competes with some fp solvers. However, the quantile plots do not tell the whole story. On the right of Fig. 3, we compare COPT and Gurobi directly: each has a large number of instances on which it is (much) better.

In terms of **reliability** of results, the exact solvers as expected produce no incorrect results; so does the slowest fp solver, lp_solve. COPT, CPLEX, HiGHS, Mosek, and fp-SoPlex perform badly in this metric, producing more errors than VI. Interestingly, these are mostly the faster solvers, the exception being Gurobi.

Overall, Gurobi achieves highest performance at decent reliability; in the remainder of this section, we thus use $\mathsf{Gurobi}_\mathsf{s}$ whenever we apply non-exact LP.

LP solver tweaking. Gurobi can be configured to use an "*auto*" portfolio approach, potentially running multiple algorithms concurrently on multiple threads, a primal or a dual simplex algorithm, or a barrier method algorithm. We compared each option with 4 threads and found no significant performance difference. Similarly, running the *auto* method with 1, 4, and 16 threads (only here, we allocate 16 threads per experiment) also failed to bring out noticeable performance differences. Using more threads results in a few more out-of-memory errors, though. We thus fix Gurobi on *auto* with 4 threads.

Fig. 4 shows the performance impact of supplying Gurobi with more precise bounds on the variables for expected reward objectives using methods from



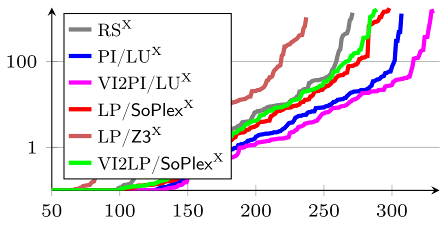
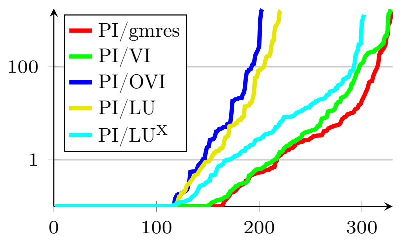


Fig. 5: Comparison of MDP model checking algorithms on the *qvbs* set

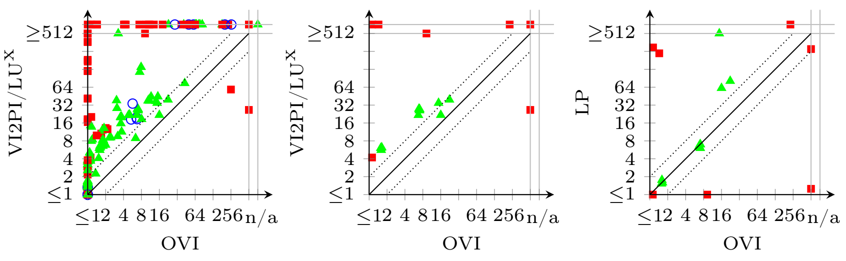
[8,51] ("bounds" instead of "simple"), of optimizing only for initial state ("init") instead of the sum over all states ("all"), and of using equality ("eq") instead of less-/greater-than-or-equal ("ineq") for unique action states. More precise bounds yield a very small improvement at essentially no cost. Optimizing for the initial state only results in a little better overall performance (in the "pocket" in the quantile plot around $x = 315$ that is also clearly visible in the scatter plot). However, it also results in 2 more incorrect results in the *qvbs* set. Using equality for unique actions noticeably decreases performance and increases the incorrect result count by 9 instances. For all experiments that follow, we thus use the more precise bounds, but do not enable the other two optimizations.



PI methods comparison. The main choice in PI is which algorithm to use to solve the induced Markov chains. On the right, we show the performance of the different algorithms available in Storm (cf. Section 4). LU^X yields a fully exact PI. This interestingly performs better than the fp version, potentially because fp errors induce spurious policy changes. The same effect likely also hinders the use of OVI, whereas VI leads to good performance. Nevertheless, gmres is best overall, and thus our choice for all following experiments with non-exact PI. VI and gmres yield 6 and 4 incorrect results, respectively. OVI and the exact methods are always correct on this benchmark set.

Best MDP algorithms for QVBS. We now compare all MDP model checking algorithms on the *qvbs* set: with floating-point numbers, LP and PI configured as described above, plus unsound VI, sound OVI, and the warm-start variants of PI and LP denoted "VI2PI" and "VI2LP", respectively. Exact results are provided by rational search (RS, essentially an exact version of VI) [50], PI with exact LU, and LP with exact solvers (SoPlex and Z3). All are implemented in Storm.

In a first experiment, we evaluated the impact of using the topological approach and of collapsing MECs (cf. Section 2.4). The results, for which we omit plots, are that the topological approach noticeably improves performance and scalability for *all* algorithms, and we therefore always use it from now on. Collapsing MECs is necessary to guarantee termination of OVI, while for the



(a) *qvbs* OVI vs. VI2PI^X (b) *hard* OVI vs. VI2PI^X (c) *hard* OVI vs. LP

Fig. 6: Additional direct performance comparisons

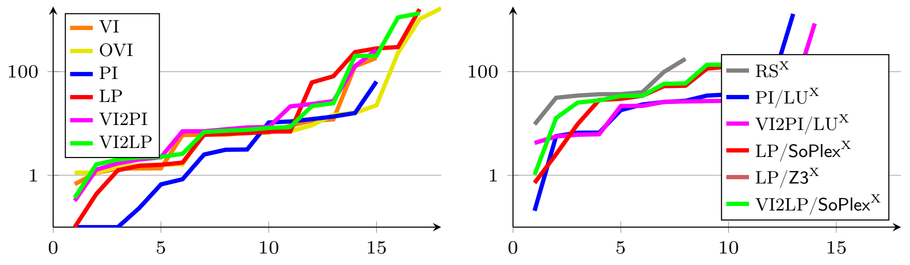


Fig. 7: Comparison of MDP model checking algorithms on the *hard* subset

other algorithms it is a potential optimization; however we found it to overall have a minimal positive performance impact only. Since it is required by OVI and does not reduce performance, we also always use it from now on.

Fig. 5 shows the complete comparison of all the methods on the *qvbs* set, for fp algorithms on the left and exact solutions on the right. Among the fp algorithms, OVI is clearly the fastest and most scalable. VI is somewhat faster but incurs several incorrect results that diminish its appearance in the quantile plot. OVI is additionally special among these algorithms in that it is sound, i.e. provides guaranteed ε-correct results—though up to fp rounding errors, which can be eliminated following the approach of [36]. On the exact side, PI with an inexact-VI warm start works best. The scatter plots in Fig. 6(a) shows the performance impact of computing an exact instead of an approximate solution.

5.2 The Hard QVBS Benchmarks

The QVBS contains many models built for tools that use VI as default algorithm. The other algorithms may actually be important to solve key challenging instances where VI/OVI perform badly. This contribution could be hidden in the sea of instances trivial for VI. We thus zoom in on a selection of QVBS instances that appear "hard" for VI: those where VI takes longer than the prior MDP state

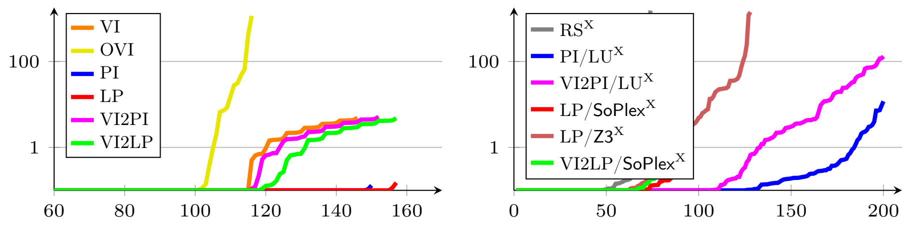


Fig. 8: Comparison of MDP model checking algorithms on the *premise* set

space construction phase in both Storm and mcsta, and additionally both phases together take at least 1 s. These are 18 of the previously considered 367 instances.

In Fig. 7, we show the behaviour of all the algorithms on this *hard* subset. OVI again works better than VI due to the incorrect results that VI returns. We see that the performance and scalability gap between the algorithms has narrowed; although OVI still "wins", LP in particular is much closer than on the full *qvbs* set. We also investigated the LP outcomes with solvers other than Gurobi: even on this set, Gurobi and COPT remain the fastest and most scalable solvers. With mcsta, in the basic configuration, they solve 16 and 17 instances, the slowest taking 835 s and 1334 s, respectively; with the topological optimization, the numbers become 17 and 15 instances with the slowest at 1373 s and 1590 s seconds. We show the detailed comparison of OVI and LP in Fig. 6(c), noting that there are a few instances where LP is much faster, and repeat the comparison between the best fp and exact algorithms (Fig. 6(b)).

5.3 The Runtime Monitoring Benchmarks

While the QVBS is intentionally diverse, our third set of benchmarks is intentionally focused: We study 200 MDPs from a runtime monitoring study [45]. The original problem is to compute the normalized risk of continuing to operate the system being monitored subject to stochastic noise, unobservable and uncontrollable nondeterminism, and partial state observations. This is a query for a conditional probability. It is answered via probabilistic model checking by unrolling an MDP model along an observed history trace of length $n \in \{ 50, \ldots, 1000 \}$ following the approach of Baier et al. [7]. The MDPs contain many transitions back to the initial state, ultimately resulting in numerically challenging instances (containing structures similar to the one of M_n in Section 2.3). We were able to compute a reference result for all instances.

Fig. 8 compares the different MDP model checking algorithms on this set. In line with the observations in [45], we see very different behaviour compared to the QVBS. Among the fp solutions on the left, LP with Gurobi terminates very quickly (under 1 s), and either produces a correct (155 instances) or a completely incorrect result (mostly 0, on 45 instances). VI behaves similarly, but is slower. OVI, in contrast, delivers no incorrect result, but instead fails to terminate on all but 116 instances. In the exact setting, warm starts using VI inherit its relative slowness and consequently do not pay off. Exact PI outperforms both exact LP solvers. In the case of exact SoPlex, out of the 112 instances it does not manage to solve, 98 are crashes likely related to a confirmed bug in its current version.

The *premise* set highlights that the best MDP model checking algorithm depends on the application. Here, in the fp case, LP appears best but produces unreliable (incorrect) results; the seemingly much worse OVI at least does not do so. Given the numeric challenge, an exact method should be chosen, and we show that these actually perform well here.

6 Conclusion

We thoroughly investigated the state of the art in MDP model checking, showing that there is no single best algorithm for this task. For benchmarks which are not numerically challenging, OVI is a sensible default, closely followed by PI and LP with a warm start—although using the latter two means losing soundness as confirmed by a number of incorrect results in our experiments. For numerically hard benchmarks, PI and LP as well as computing exact solutions are more attractive, and clearly preferable in combination. Overall, although LP has the superior (polynomial) theoretical complexity, in our practical evaluation, it almost always performs worse than the other (exponential) approaches. This is even though we use modern commercial solvers and tune both the LP encoding of the problem as well as the solvers' parameters. While we *observed* the behaviour of the different algorithms and have some intuition into what makes the *premise* set hard, an entire research question of its own is to identify and quantify the structural properties that make a model hard.

Our evaluation also raises the question of how prevalent MDPs that challenge VI are in practice. Aside from the *premise* benchmarks, we were unable to find further sets of MDPs that are hard for VI. Notably, several stochastic games (SGs) difficult for VI were found in [46]; the authors noted that using PI for the SGs was better than applying VI to the SGs. However, when we extracted the induced MDPs, we found them all easy for VI. Similarly, [3] used a random generation of SGs of at most 10,000 states, many of which were challenging for the SG algorithms. Yet the same random generation modified to produce MDPs delivered only MDPs easily solved in seconds, even with drastically increased numbers of states. In contrast, Alagöz et al. [1] report that their random generation returned models where LP beat PI. However, their setting is discounted, and their description of the random generation was too superficial for us to be able to replicate it. We note that, in several of our scatter plots, the MA instances from the QVBS (where we check the embedded MDP) appeared more challenging overall than the MDPs. We thus conclude this paper with a call for challenging MDP benchmarks—as separate benchmark sets of unique characteristics like *premise*, or for inclusion in the QVBS.

Data availability statement. The datasets generated and analysed in this study and code to regenerate them are available in the accompanying artifact [38]. For Storm, our code builds on version 1.7.0. We used mcsta version 3.1.213.

References

1. Alagöz, O., Ayvaci, M.U.S., Linderoth, J.T.: Optimally solving Markov decision processes with total expected discounted reward function: Linear programming revisited. Comput. Ind. Eng. **87**, 311–316 (2015). https://doi.org/10.1016/j.cie.2015.05.031
2. Anand, R., Aggarwal, D., Kumar, V.: A comparative analysis of optimization solvers. Journal of Statistics and Management Systems **20**(4), 623–635 (2017). https://doi.org/10.1080/09720510.2017.1395182
3. Azeem, M., Evangelidis, A., Kretínský, J., Slivinskiy, A., Weininger, M.: Optimistic and topological value iteration for simple stochastic games. CoRR **abs/2207.14417** (2022). https://doi.org/10.48550/arXiv.2207.14417
4. Baier, C., de Alfaro, L., Forejt, V., Kwiatkowska, M.: Model checking probabilistic systems. In: Handbook of Model Checking, pp. 963–999. Springer (2018)
5. Baier, C., Hermanns, H., Katoen, J.P.: The 10,000 facets of MDP model checking. In: Computing and Software Science, LNCS, vol. 10000, pp. 420–451. Springer (2019). https://doi.org/10.1007/978-3-319-91908-9_21
6. Baier, C., Katoen, J.P.: Principles of model checking. MIT Press (2008), https://mitpress.mit.edu/books/principles-model-checking
7. Baier, C., Klein, J., Klüppelholz, S., Märcker, S.: Computing conditional probabilities in Markovian models efficiently. In: TACAS. LNCS, vol. 8413, pp. 515–530. Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_43
8. Baier, C., Klein, J., Leuschner, L., Parker, D., Wunderlich, S.: Ensuring the reliability of your model checker: Interval iteration for Markov decision processes. In: CAV (1). LNCS, vol. 10426, pp. 160–180. Springer (2017). https://doi.org/10.1007/978-3-319-63387-9_8
9. Balaji, N., Kiefer, S., Novotný, P., Pérez, G.A., Shirmohammadi, M.: On the complexity of value iteration. In: ICALP. LIPIcs, vol. 132, pp. 102:1–102:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPIcs.ICALP.2019.102
10. Berkelaar, M., Eikland, K., Notebaert, P.: Introduction to lp_solve 5.5.2.11, https://lpsolve.sourceforge.net/5.5/, accessed 2023-01-25.
11. Bertsekas, D.P., Tsitsiklis, J.N.: An analysis of stochastic shortest path problems. Math. Oper. Res. **16**(3), 580–595 (1991). https://doi.org/10.1287/moor.16.3.580
12. Boyd, S.P., Vandenberghe, L.: Convex Optimization. Cambridge University Press (2014)
13. Brázdil, T., Chatterjee, K., Chmelik, M., Forejt, V., Kretínský, J., Kwiatkowska, M.Z., Parker, D., Ujma, M.: Verification of Markov decision processes using learning algorithms. In: ATVA. LNCS, vol. 8837, pp. 98–114. Springer (2014). https://doi.org/10.1007/978-3-319-11936-6_8
14. Budde, C.E., Hartmanns, A., Klauck, M., Kretínský, J., Parker, D., Quatmann, T., Turrini, A., Zhang, Z.: On correctness, precision, and performance in quantitative verification – QComp 2020 competition report. In: ISoLA (4). LNCS, vol. 12479, pp. 216–241. Springer (2020). https://doi.org/10.1007/978-3-030-83723-5_15
15. Chatterjee, K., Henzinger, T.A.: Value iteration. In: 25 Years of Model Checking. LNCS, vol. 5000, pp. 107–138. Springer (2008). https://doi.org/10.1007/978-3-540-69850-0_7
16. Cubuktepe, M., Jansen, N., Junges, S., Katoen, J.P., Topcu, U.: Convex optimization for parameter synthesis in MDPs. IEEE Trans. Autom. Control. (2022). https://doi.org/10.1109/TAC.2021.3133265

17. Dai, P., Mausam, Weld, D.S., Goldsmith, J.: Topological value iteration algorithms. J. Artif. Intell. Res. **42**, 181–209 (2011), https://www.jair.org/index.php/jair/article/view/10725
18. Dutertre, B., de Moura, L.M.: A fast linear-arithmetic solver for DPLL(T). In: CAV. LNCS, vol. 4144, pp. 81–94. Springer (2006)
19. Eisentraut, J., Kelmendi, E., Kretínský, J., Weininger, M.: Value iteration for simple stochastic games: Stopping criterion and learning algorithm. Inf. Comput. **285**(Part), 104886 (2022). https://doi.org/10.1016/j.ic.2022.104886
20. Fearnley, J.: Exponential lower bounds for policy iteration. In: ICALP (2). LNCS, vol. 6199, pp. 551–562. Springer (2010). https://doi.org/10.1007/978-3-642-14162-1_46
21. Forejt, V., Kwiatkowska, M.Z., Parker, D.: Pareto curves for probabilistic model checking. In: ATVA. LNCS, vol. 7561, pp. 317–332. Springer (2012). https://doi.org/10.1007/978-3-642-33386-6_25
22. Free Software Foundation: The GNU Multiple Precision Arithmetic Library, https://gmplib.org/, accessed 2023-01-25.
23. Funke, F., Jantsch, S., Baier, C.: Farkas certificates and minimal witnesses for probabilistic reachability constraints. In: TACAS (1). LNCS, vol. 12078, pp. 324–345. Springer (2020)
24. Ge, D., Huangfu, Q., Wang, Z., Wu, J., Ye, Y.: Cardinal Optimizer (COPT) user guide (2022), https://guide.coap.online/copt/en-doc
25. GetFEM project: Gmm++ Library, https://getfem.org/gmm/, accessed 2023-01-25.
26. Giro, S.: Optimal schedulers vs optimal bases: An approach for efficient exact solving of Markov decision processes. Theor. Comput. Sci. **538**, 70–83 (2014). https://doi.org/10.1016/j.tcs.2013.08.020
27. Gleixner, A.M., Steffy, D.E., Wolter, K.: Improving the accuracy of linear programming solvers with iterative refinement. In: ISSAC. pp. 187–194. ACM (2012)
28. Gleixner, A.M., Steffy, D.E., Wolter, K.: Iterative refinement for linear programming. Tech. Rep. 3, ZIB, Takustr. 7, 14195 Berlin (2016). https://doi.org/10.1287/ijoc.2016.0692
29. GNU Project: GLPK (GNU Linear Programming Kit), http://www.gnu.org/software/glpk/glpk.html
30. Google: Glop – linear optimization, https://developers.google.com/optimization/lp, accessed 2023-01-25.
31. Guennebaud, G., Jacob, B., et al.: Eigen v3 (2010), http://eigen.tuxfamily.org
32. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2022), https://www.gurobi.com
33. Haddad, S., Monmege, B.: Reachability in MDPs: Refining convergence of value iteration. In: RP. LNCS, vol. 8762, pp. 125–137. Springer (2014)
34. Haddad, S., Monmege, B.: Interval iteration algorithm for MDPs and IMDPs. Theor. Comput. Sci. **735**, 111–131 (2018). https://doi.org/10.1016/j.tcs.2016.12.003
35. Hall, J., Galabova, I., Gottwald, L., Feldmeier, M.: HiGHS – high performance software for linear optimization, https://www.maths.ed.ac.uk/hall/HiGHS/, accessed 2023-01-25.
36. Hartmanns, A.: Correct probabilistic model checking with floating-point arithmetic. In: TACAS (2). LNCS, vol. 13244, pp. 41–59. Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_3
37. Hartmanns, A., Hermanns, H.: The Modest Toolset: An integrated environment for quantitative modelling and verification. In: TACAS. LNCS, vol. 8413, pp. 593–598. Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_51

38. Hartmanns, A., Junges, S., Quatmann, T., Weininger, M.: A Practitioner's Guide to MDP Model Checking Algorithms (Artefact) (2023). https://doi.org/10.5281/zenodo.7509474
39. Hartmanns, A., Junges, S., Quatmann, T., Weininger, M.: A practitioner's guide to MDP model checking algorithms (2023). https://doi.org/10.48550/ARXIV.2301.10197
40. Hartmanns, A., Kaminski, B.L.: Optimistic value iteration. In: CAV (2). LNCS, vol. 12225, pp. 488–511. Springer (2020). https://doi.org/10.1007/978-3-030-53291-8_26
41. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The quantitative verification benchmark set. In: TACAS. LNCS, vol. 11427, pp. 344–350. Springer (2019). https://doi.org/10.1007/978-3-030-17462-0_20
42. Hensel, C., Junges, S., Katoen, J.P., Quatmann, T., Volk, M.: The probabilistic model checker Storm. Int. J. Softw. Tools Technol. Transf. **24**(4), 589–610 (2022). https://doi.org/10.1007/s10009-021-00633-z
43. Huangfu, Q., Hall, J.A.J.: Parallelizing the dual revised simplex method. Math. Program. Comput. **10**(1), 119–142 (2018). https://doi.org/10.1007/s12532-017-0130-5
44. IBM: IBM ILOG CPLEX Optimizer, https://www.ibm.com/analytics/cplex-optimizer, accessed 2023-01-25.
45. Junges, S., Torfah, H., Seshia, S.A.: Runtime monitors for Markov decision processes. In: CAV (2). LNCS, vol. 12760, pp. 553–576. Springer (2021)
46. Kretinsky, J., Ramneantu, E., Slivinskiy, A., Weininger, M.: Comparison of algorithms for simple stochastic games. Inf. Comput. (2022). https://doi.org/10.1016/j.ic.2022.104885
47. Kumar, A., Zilberstein, S.: History-based controller design and optimization for partially observable MDPs. In: ICAPS. vol. 25, pp. 156–164 (2015)
48. Kwiatkowska, M.Z., Norman, G., Parker, D., Sproston, J.: Performance analysis of probabilistic timed automata using digital clocks. Formal Methods Syst. Des. **29**(1), 33–78 (2006). https://doi.org/10.1007/s10703-006-0005-2
49. Littman, M.L., Dean, T.L., Kaelbling, L.P.: On the complexity of solving Markov decision problems. In: UAI. pp. 394–402. Morgan Kaufmann (1995)
50. Mathur, U., Bauer, M.S., Chadha, R., Sistla, A.P., Viswanathan, M.: Exact quantitative probabilistic model checking through rational search. Formal Methods Syst. Des. **56**(1), 90–126 (2020)
51. McMahan, H.B., Likhachev, M., Gordon, G.J.: Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In: ICML. ACM International Conference Proceeding Series, vol. 119, pp. 569–576. ACM (2005). https://doi.org/10.1145/1102351.1102423
52. MOSEK ApS: The MOSEK Optimization Suite 10.0.34, https://docs.mosek.com/latest/intro/index.html, accessed 2023-01-25.
53. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: TACAS. LNCS, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24
54. Phalakarn, K., Takisaka, T., Haas, T., Hasuo, I.: Widest paths and global propagation in bounded value iteration for stochastic games. In: CAV (2). LNCS, vol. 12225, pp. 349–371. Springer (2020), https://doi.org/10.1007/978-3-030-53291-8_19
55. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Statistics, Wiley (1994). https://doi.org/10.1002/9780470316887
56. Quatmann, T., Katoen, J.P.: Sound value iteration. In: CAV (1). LNCS, vol. 10981, pp. 643–661. Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_37

57. Saad, Y., Schultz, M.H.: Gmres: a generalized minimal residual algorithm for solving nonsymmetric linear systems. Siam Journal on Scientific and Statistical Computing **7**, 856–869 (1986), https://epubs.siam.org/doi/10.1137/0907058
58. Wimmer, R., Jansen, N., Vorpahl, A., Ábrahám, E., Katoen, J.P., Becker, B.: High-level counterexamples for probabilistic automata. Log. Methods Comput. Sci. **11**(1) (2015)
59. Wimmer, R., Kortus, A., Herbstritt, M., Becker, B.: Probabilistic model checking and reliability of results. In: DDECS. pp. 207–212. IEEE Computer Society (2008). https://doi.org/10.1109/DDECS.2008.4538787
60. Ye, Y.: The simplex and policy-iteration methods are strongly polynomial for the Markov decision problem with a fixed discount rate. Mathematics of Operations Research **36**(4), 593–603 (2011)