Playing Games with Your PET: Extending the Partial Exploration Tool to Stochastic Games

Tobias Meggendorfer[1] and Maximilian Weininger[2(✉)]

[1] Lancaster University Leipzig, Leipzig, Germany
tobias@meggendorfer.de
[2] Institute of Science and Technology Austria, Klosterneuburg, Austria
mweining@ista.ac.at

Abstract. We present version 2.0 of the *Partial Exploration Tool* (Pet), a tool for verification of probabilistic systems. We extend the previous version by adding support for *stochastic games*, based on a recent unified framework for sound value iteration algorithms. Thereby, Pet2 is the first tool implementing a sound and efficient approach for solving stochastic games with objectives of the type reachability/safety and mean payoff. We complement this approach by developing and implementing a partial-exploration based variant for all three objectives. Our experimental evaluation shows that Pet2 offers the most efficient partial-exploration based algorithm and is the most viable tool on SGs, even outperforming unsound tools.

Keywords: Probabilistic verification · Stochastic games · Partial exploration · Model checker

1 Introduction

Stochastic games (SGs) [12] are a foundational model for sequential decision making in the presence of uncertainty and two antagonistic agents. They are practically relevant, with applications ranging from economics [1] over IT security [35] to medicine [7]; and they are theoretically fundamental, in particular because many associated classical decision problems are representative of the important complexity class NP ∩ co-NP, e.g. deciding whether the value of a reachability or mean payoff objective is greater than a given threshold [2,22] (see [10] for recent advances). See [36, Chp. 1.3] for further motivation.

However, even mature tools either do not support SGs at all (Storm [21]) or employ approaches without formal guarantees, i.e. their results can be wrong

M. Weininger has received funding from the EU's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101034413.
Data availability: We refer to the artefact with the exact code used for the submission and all logs [31], and the gitlab with the continually developed source code [30].

A. Gurfinkel and V. Ganesh (Eds.): CAV 2024, LNCS 14683, pp. 359–372, 2024.
https://doi.org/10.1007/978-3-031-65633-0_16

(Prism-games [27] and Tempest [33]), which is unacceptable in the context of safety-critical applications. This is because *value iteration* (VI), the de-facto standard approach to solving stochastic systems, lacks a sound and efficient *stopping criterion*, i.e. a "rule" to check whether the current iterates are sufficiently close to the correct value. For Markov decision processes (MDPs) (SGs with only one player) such a sound variant of VI (often called *interval iteration*) was developed a decade ago [8,17] and subsequently implemented in practically all major model checkers. However, extending the underlying reasoning to SGs proved to be surprisingly tricky, with sound variants even for special cases only developed quite recently [15]. Just a year ago, [24] presented a unified way of ensuring the soundness of VI for solving SGs with various quantitative objectives, which forms the theoretical basis for this work.

Note that the classical approaches strategy iteration and quadratic programming are sound in theory, but (i) the available implementations of [26] are prototypical and unsound [26, Sec. 5], and (ii) these approaches usually either are not practically efficient or use heuristics that actually make them unsound [18].

Contributions. We present version 2.0 of the *Partial Exploration Tool* (Pet), the first tool implementing a sound and efficient approach for solving SGs with objectives of the type reachability/safety and mean payoff (a.k.a. long-run average reward). In the following, we write Pet1 and Pet2 to refer to the previous and now presented version of Pet, respectively.

Theoretically, Pet2 is based on the results of [24]. We provide two flavours of their approach: Firstly, we implement the basic complete-exploration (CE) algorithm, enhanced with several theoretical improvements, both new and suggested in the literature. Secondly, we develop a *partial-exploration (PE)* approach, the focus of Pet2, by combining the ideas of [24] with those in [8,15,29].

Practically, Pet2 is an extension of Pet1 [29] (only applicable to MDPs). Apart from adding support for dealing with SGs and completely replacing the approach of Pet1 with the ideas of [24], we implemented many engineering improvements. Concretely, despite employing a more general algorithm, our experimental evaluation shows that Pet2's performance is on par with Pet1. Moreover, Pet2 outperforms the existing SG solvers Prism-games and Tempest, despite those not providing guarantees (and indeed returning wrong results).

2 Preliminaries

Here, we very briefly recall turn-based stochastic games as far as they are necessary to understand this paper, with more details in [32, App. A] and [24].

A *(turn-based) stochastic game (SG)* (e.g. [12]) consists of a set of states S that belong to either the *Maximizer* or *Minimizer* player; a set of available actions for every state, denoted $\mathsf{A}(s)$; and a probabilistic transition function δ that for a state-action pair gives a probability distribution over successor states. An SG where all states belong to one player is called *Markov decision process (MDP)*, see [34]; without nondeterministic choices, it is a *Markov chain (MC)*.

SGs are played in turns as follows: Starting in an *initial state* s_0, the player to whom this state belongs chooses an action $a_0 \in \mathsf{A}(s)$. Then, the play advances to the next state s_1, which is sampled according to the probability distribution given by $\delta(s, a)$. Repeating this process indefinitely yields an infinite path $\rho = s_0 a_0 s_1 a_1 \ldots$. We write $\mathsf{Paths}_\mathsf{G}$ for the set of all such infinite paths in a game G.

A *memoryless deterministic (MD) strategy* σ of Maximizer assigns an action to every Maximizer state s, i.e. $\sigma(s) \in \mathsf{A}(s)$. Minimizer strategies τ are defined analogously. By fixing a pair of strategies (σ, τ) and thereby resolving all non-deterministic choices, we obtain a Markov chain that together with an initial state $\hat{s}$ induces a unique probability distribution over the set of all infinite paths $\mathsf{Paths}_\mathsf{G}$ [6, Sec. 10.1]. For a random variable over paths $\Phi : \mathsf{Paths}_\mathsf{G} \to \mathbb{R}$ we write $\mathbb{E}^{\sigma,\tau}_{\mathsf{G},\hat{s}}[\Phi]$ for its expected value under this probability measure.

An *objective* $\Phi : \mathsf{Paths}_\mathsf{G} \to \mathbb{R}$ formalizes the "goal" of both players by assigning a value to each path. In this paper, we focus on *mean payoff* (also called long-run average reward) [16], which assign to every path the average reward that is obtained in the limit. The presented algorithms and tools can also explicitly handle reachability/safety objectives, which compute the probability of reaching a given set of states while avoiding another. Such objectives are special cases of mean payoff, see e.g. [4]. Another prominent objective is *total reward* [11], but this is practically incompatible with our approach and goals (see [32, App. B]).

Given an SG and an objective, we want to compute the *value of the game*, i.e. the optimal value the players can ensure by choosing optimal strategies. Formally, the value of state s is defined as $\mathsf{V}_{\mathsf{G},\Phi}(s) := \sup_\sigma \inf_\tau \mathbb{E}^{\sigma,\tau}_{\mathsf{G},s}[\Phi] (= \inf_\tau \sup_\sigma \mathbb{E}^{\sigma,\tau}_{\mathsf{G},s}[\Phi])$. We are interested in *approximate solutions*, i.e. given a concrete state $\hat{s}$ and precision requirement ε, our goal is to determine a number v such that $|\mathsf{V}_{\mathsf{G},\Phi}(\hat{s}) - v| < \varepsilon$.

An *end component (EC)* intuitively is a set of states in which the system *can* remain forever, given suitable strategies. Inclusion-maximal ECs are *maximal end components (MECs)*. The play of an SG eventually remains inside a single MEC with probability one [14]. In other words, MECs capture all relevant long-run behaviour of the system. The set of MECs can be identified in PTIME [13].

3 Complete-Exploration Algorithm for Solving SGs

In this section, we very briefly recall Alg. 2 of [24], a generic value-iteration based approach for SGs, which in particular is the first such algorithm that provides guarantees on the precision for mean payoff. This recapitulation is the basis for the following descriptions of both (i) the main practical improvements over [24, Alg. 2] added in our implementation (see the end of this section); and (ii) the new partial-exploration approach described in Sec.4.

Intuition. The key insight of [24] is to "split" the analysis of SGs into infinite and transient behaviour. Infinite behaviour occurs in ECs where both players want to remain under optimal strategies; this is where the mean payoff is actually "obtained". Transient behaviour occurs in states that are not part of such an

Fig. 1. Example SGs to explain deflating and inflating. States with upward and downward triangles denote Maximizer and Minimizer states, respectively.

EC, i.e. that are almost surely only visited finitely often; such states in turn achieve their value by trying to reach ECs that give them the best mean payoff.

Algorithm. Based on this intuition, we can summarize the overall structure of the algorithm. It maintains two functions L and U, which map every state to a lower and upper bound on its true value, respectively. Our aim is to improve these bounds until they are sufficiently close to each other; then we can derive the correct value up to precision ε. After initializing the bounds to safe under- and over-approximations (e.g. the minimum and maximum reward occurring in the SG), we repeatedly perform two operations, further described below: Firstly, we use so-called *Bellman updates* to back-propagate bounds through the SG, which also is the classical "value iteration step". This corresponds to the transient behaviour, intuitively computing the optimal choice of actions to reach the ECs with the best value. Secondly, we use the operations of *deflating* and *inflating*. These essentially inform states about the value obtainable by staying.

Bellman Updates. Bellman updates are at the core of all value iteration style algorithms (see e.g. [9]). Intuitively, they update the current estimates by "taking one step", i.e. computing the expectation of following the action that is optimal according to the current estimates. Formally, given a function $x \colon \mathsf{S} \to \mathbb{R}$, the Bellman update $\mathcal{B}$ computes a new estimate function as $\mathcal{B}(x)(s) := \mathsf{opt}^s_{a \in \mathsf{A}(s)} \sum_{s' \in \mathsf{S}} \delta(s,a)(s') \cdot x(s')$, where $\mathsf{opt}^s = \max$ if s is a Maximizer state and min otherwise. Importantly, if x is a correct lower or upper bound on the true value, then $\mathcal{B}(x)(s)$ is, too. However, only applying $\mathcal{B}(x)$ may not converge!

Deflating and Inflating. We briefly describe why the convergence problem arises and how deflating and inflating solve the issue, summarizing [24, Sec. V-C].

Recall the intuition that a state can get its value either from infinite behaviour (i.e. "staying" in the current area of the game) or from transient behaviour ("leaving" the current area). As a simple example, consider the SG (even MDP) in Fig. 1 (left). Assume we have $\mathsf{U} = 5$ in the grey area, i.e. by taking action b we obtain at most 5, but for s the current upper bound is the conservative over-approximation $\mathsf{U}(s) = 10$. The Bellman update prefers a over b and keeps $\mathsf{U}(s)$ at 10, even though following a forever will only yield a mean payoff of 4. This is due to a cyclic dependency: s "believes" it can (eventually) achieve 10 because it "promises" to achieve 10 by having $\mathsf{U}(s) = 10$. Deflating now identifies that Maximizer wants to "stay" in s using a, computes the actual value that is obtained by staying, i.e. 4, and compares it to the best possible exit from this region,

namely leaving with b to obtain at most 5. Maximizer has to either stay forever or eventually leave, thus we can decrease the upper bound to the maximum of these two actions, i.e. 5. Dually, inflating raises the value of Minimizer states to the minimum of leaving and staying. In other words, de-/inflating complement Bellman updates by informing players about the consequences of staying forever, forcing them to choose between this staying value and the best exit.

While this reasoning is simple for single-state, single-player cycles, it gets much more involved in the general case of stochastic games. In particular, and in contrast to MDPs, in two-player SGs the opponent can restrict which cycles and exits are reachable from certain states. For example, consider the SG in Fig. 1 (right): States p and s can form a cycle. However, if s goes to p, it depends on the choice of Minimizer in p whether the play stays in the cycle $\{p, s\}$ or leaves towards X. To tackle this issue, [24, Alg. 2] repeatedly identifies regions where players want to remain *based on their current estimates*, called *simple end component (SEC)*-candidates. It does so by fixing one player's choices (pretending that this is the strategy they "commit" to), and the regions where the other player could remain are the SEC-candidates. These are then de-/inflated. We highlight that as the player's estimates change, the SEC-candidates do, too.

Improvements. We provide several practical improvements to this approach. We briefly describe their ideas here and refer to [32, App. C] for further information.

- Instead of searching SEC-candidates in the complete SG, we once identify all MECs and then search for SEC-candidates in each MEC independently.
- SEC-candidate search is not performed every iteration, but only heuristically (improving a suggestion from [15, Sec. 6.2]).
- Instead of computing staying values precisely, which is often unnecessary, we successively approximate them (as suggested in [25, App. E-B]).
- If possible, we locally employ MDP solution approaches, *collapsing* ECs that are completely "controlled" by a single player into one state that then is handled by the normal Bellman updates (as suggested in [15, Sec. 6.2]). This transparently generalizes the existing algorithms for MDPs [4,8,17].

4 Partial-Exploration Algorithm for Solving SGs

Here, we present the novel *partial-exploration (PE)* algorithm obtained by combining the *complete-exploration (CE)* algorithm of Sect. 3 with the ideas of partial exploration [8,15,29]. Intuitively, for particular models and objectives, some states are hardly relevant, and computing their exact value is unnecessary for an ε-precise result. For example, in the `zeroconf` protocol (choosing an IP address in a nearly empty network), it is hardly interesting what we should do when we run into a collision 10 times: Since this is so unlikely to happen, the exact outcome in this case barely influences the true result. Thus, we avoid exploring what exactly is possible in this case and just assume the worst.

More generally, we want to avoid working on the complete model when executing Bellman updates or de-/inflating SEC-candidates. Instead, we use simulations to partially explore the model, finding the states that are likely to be reached under optimal strategies, and focus computations on these. If the "relevant" part of the state space (see [23]) is small in comparison to the whole model, we can save a large amount of time and memory. We refer to the [8,15,29] for a comprehensive discussion of the (dis-)advantages of this approach.

Algorithm. The algorithm follows the established structure of [8,15]: We sample a path through the model, at every state picking an action and a successor according to a guidance heuristic that prefers "relevant" states. We terminate the simulation when it has reached a state where continuing the path does not generate new information (e.g. an EC that cannot be exited). Then, we perform Bellman updates, but only on the states of the sampled path. Additionally, we repeatedly identify both collapsible areas and SEC-candidates in the partial model and de-/inflate if necessary. This final step is the main technical difference to the previous algorithms [8,15], which only employed collapsing and deflating, respectively. It also is one of the major engineering difficulties, see [32, App. D.1].

Soundness and Correctness. In [32, App. D.2], we extend the proof of correctness and termination from reachability [15, Thm. 3] to mean payoff. In the process of proving correctness, we found and fixed an error in the guidance heuristic of [15].

Practical Improvements. As before, we applied several optimizations and heuristics to this algorithm. Broadly speaking, the overall ideas are the same as for the complete exploration approach, however with several intricacies. In particular, observe that the partial model constantly changes as new states are explored. Thus, efficiently tracking and updating SEC-candidates or collapsible parts of the game is much more involved and intertwined with the rest of the algorithm.

5 Tool Description

In this section, we briefly describe relevant aspects of Pet2, focussing on new features and changes compared to its predecessor Pet1. Like Pet1, Pet2 is implemented in Java, reads models and objectives specified in the PRISM modelling language, and outputs the computed value in JSON format.

Design Choices. Firstly, Pet2 exclusively employs (sound) VI-based approaches, as opposed to, e.g., SI or LP. It offers two variants, based on complete exploration (CE) and partial exploration (PE), as presented above.

Secondly, Pet2 deliberately comes without any configuration flags. In contrast, model checkers such as Prism [27] and Storm [21] implement numerous different approaches and variants, each of which offers several hyper-parameters. While our choice eliminates the potential for fine-tuning, we experienced that even expert users often do not know how to best choose such parameters. Thus,

we tried to select internal parameters such that the tool works reasonably well out-of-the-box on all models. This comes with the additional benefit of greatly reducing the number of code paths, which in turn makes testing much easier.

Thirdly, PET2 fully commits to solving a single combination of model and objective. This allows us to exploit information about the objective already when building the model; for example, in reachability/safety objectives, we can directly re-map goal and unsafe states to dedicated absorbing states. Arguably, this might become a drawback when solving multiple queries on the same model, since the model would need to be explored several times. However, by caching the parts of the model constructed so far, we can effectively eliminate this problem. We discuss further ways to exploit this design choice in [32, App. E].

Finally, PET2 does not differentiate between Markov chains, MDPs, and SGs. The algorithms are written for SGs, and, whenever possible, apply specialized solutions locally. For example, if a part of an SG "looks like" an MDP (because only one player has meaningful choices), PET locally applies MDP reasoning where applicable. Aside from transparently gaining the performance of these specialized solutions in most cases, this also eliminates code duplication, resulting in a code base that is more understandable, maintainable, and extendable.

Differences to PET1. PET2 effectively constitutes a complete re-write of nearly all aspects of PET, with roughly 9k lines of code added and 5k deleted compared to PET1 (according to `git`); for comparison, the overall source code of PET2 has roughly 14k lines. Most importantly, PET2 now also parses SGs, fully focusses on solving one model-objective combination, and also provides efficient CE variants of the algorithms. These CE variants also come with numerous optimizations, such as graph analysis to identify states with value 0 and 1 for reachability and safety, collapsing end components, etc. Moreover, each PE variant is now specialized to the concrete objective. PET1 tried to unify as many aspects of the sampling approach as possible, which however proved to be a major design obstacle and performance penalty when incorporating all the different specialized solutions and practical optimizations for stochastic games. Additionally, we also found and fixed a bug in the mean payoff computation of PET1. In terms of data structures, we improved several smaller aspects of working with probabilistic models. For example, the "standard" internal model representation of PET2 offers dedicated support for merging / collapsing sets of states while simultaneously tracking predecessors of each state. We note that the model representation etc. is still provided by our separately available, generic purpose library, making many of these improvements available independently of PET.

Engineering Improvements. We evaluated several practical improvements, which sometimes led to quite surprising effects. We highlight some insights which we deem relevant for other developers.

1) "Unrolling" loops in hot zones of the code can lead to significant performance improvements. For example, to find the optimal action for a Bellman update, we need to use a for-loop to iterate over all available actions. This process is

performed millions of times during a normal execution. Unrolling and specializing these loops for small action sets (≤ 3 actions) led to noticeable performance improvements. Similarly, switching from for-each loops (which allocate an iterator) to index-based for loops also yielded notable improvements.

2) We trade memory for time by adding additional data structures to optimize certain access patterns. For example, maintaining the set of predecessors for each state speeds up graph algorithms such as attractor computations. Moreover, in several cases it proved beneficial to store information in multiple formats. For example, on top of a sorted array-based set, we explicitly store the set of successors of a distribution object as a (roaring) bitmap [28], offering fast "bulk" operations, such as intersection or subset checks.
3) We also investigated ahead-of-time compilation through GraalVM. While this improved start-up time, it did not result in significant speed-ups but rather even increased the runtime on some of the larger models (even with profile-guided optimization). We conjecture that this is mainly due to Java's just-in-time compiler being able to apply better fine-tuning.

6 Experimental Evaluation

We now discuss our evaluation. The first goal of our experiments is to validate that Pet2 can indeed solve SGs with reachability and mean payoff objectives in a sound way. Secondly, we assess the impact of our performance improvements and design choices, in particular having only the general algorithm for SG, by comparing Pet1 and Pet2. Finally, we investigate whether our implementation is competitive with other tools. We report some further insights in [32, App. F.3]. Our artefact, including all data, tools, scripts, logs, etc., is available at [31].

6.1 Experimental Setup

Technical Setup. We ran each experiment in a separate Docker container and, as usual, restricted it to a single CPU core (of an AMD Ryzen 5 3600) and 8 GB RAM. The timeout is 60 s (including the startup time of the Docker container). We ran every instance three times to even out potential fluctuations in execution times. While the PE approach is randomized by design, even "deterministic" algorithms may behave differently due to, e.g., non-deterministic iteration order of hash sets. We observed that the variance is negligible (the geometric standard deviation usually was ≤ 1.05). We thus only report the geometric average of the three runs in seconds. We require an absolute precision of $\varepsilon = 10^{-6}$ for all experiments.

Metrics. To summarize relative performance of Pet2 compared to tool `X`, we introduce a four-figure score, written $t[m{+}k/l]$, computed as follows: Let M the set of instances *where both tools terminated in time*. Then, t equals the geometric mean of $\mathrm{time}_{\mathtt{X}}(I)/\mathrm{time}_{\textsc{Pet2}}(I)$ over all instances $I \in M$, with $\mathrm{time}_{\mathcal{T}}$ referring to the overall runtime of tool $\mathcal{T}$ on an instance, $m = |M|$ refers to the number of

such instances, while k describes how often `X` timed out where PET2 did not and l vice versa. Note that $t > 1$ indicates that PET2 is faster on average (on M). When $t < 1$ but $k \gg l$, we see that on instances that both tools solved, PET2 was slower, but overall, PET2 solved much more models, which one may still consider advantageous.

Tools. Aside from both versions of PET, for Markov chains and MDPs we consider PRISM-GAMES[1] [27], and STORM [21]. On SGs, we compare to the unsound algorithms in PRISM-GAMES and TEMPEST [33] (an extension of STORM), as well as PRISM-EXT, an extension of PRISM-GAMES with sound algorithms described in [5,26]. Note that this selection includes all tools that participated in the SG performance comparison in QComp 2023 [3]. For all tools, we provide the exact version, ways to obtain them, and invocations in [32, App. F.1] and the artefact.

Performance Considerations. Restricting to a single CPU is commonly done to ensure that no tool accidentally exploits parallelism. However, we observed a significant decrease in performance for PET, even though all algorithms are sequential. This turned out to be due to garbage collection. Using `jhsdb`, we verified that in the single CPU case Java by default selects the Serial GC (instead of the overall default G1GC). On some instances, we consistently observed improvements of **up to 33%** (!), nearly on par with the performance without any CPU restriction, by simply changing to the parallel GC (`-XX:+UseParallelGC`), even though the parallel GC uses only one thread. Concretely, comparing CE and PE with Serial and (single-thread) parallel GC, we get scores of 1.06 and 1.04, respectively, meaning that even *on average* this change leads to a significant difference. Interestingly, for PRISM-GAMES the Serial GC performed better. We configured PET2 to always use the Parallel GC by default. In a similar manner, the hybrid engine of STORM experienced a slowdown of more than 30x due to being restricted to a single CPU, which we addressed by adding appropriate switches (see [32, App. F.1] for details). We are working with the authors of STORM to automatically detect this case.

While these differences would not invalidate our conclusions in particular, we still want to highlight these observations and emphasize the importance of both careful evaluation and choosing good default parameters.

Benchmarks. We consider benchmarks from multiple sources. Firstly, we include applicable models from the quantitative verification benchmark set (QVBS) [20], which however does not provide SGs. Secondly, we consider the SGs used in QComp 2023 [3]. Finally, we also gather several models from literature, provide variations of existing models, and create completely new models. For details on the models, we refer to [32, App. F.2]. All models are included in the artefact.

To ease the evaluation, we remove instances of QVBS that are very simple (CE- and PE-approach of STORM and PET2 taking less than one second) or

[1] Personal communication with the lead developer confirmed that on Markov chains and MDPs PRISM-GAMES uses the same approach as PRISM.

very time-consuming (all four approaches taking more than 30 s). With such a timespan, differences and trends are clearly visible, but models remain small enough for the experiments to be reproducible within reasonable time. This filtering reduces the number of executions from nearly 10000 to about 1800. Even then, the overall evaluation still takes about 24 h (with timeout of 60 s).

6.2 Results

We present central results in Fig. 2 and discuss each of our research questions.

Soundness and Scalability. We empirically validate the correctness of Pet2 by (i) comparing against the reference results in QVBS (this only affects the specialized MDP reasoning of our algorithm), (ii) ensuring that both algorithms inside Pet2 yield the same results, and (iii) comparing against manually computed values, both for existing SG benchmarks as well as handcrafted ones exhibiting various graph structures (some of which arose as test or corner cases). In all cases, Pet2's results are sound, i.e. within the allowed precision of $\varepsilon = 10^{-6}$. In contrast, throughout the whole SG benchmark set, Prism-games and Tempest return several wrong answers, see [32, App. F.3] for details. In particular, Tempest returns wrong answers in 6 out of 13 cases where we have known reference results, often by a significant margin, e.g. returning 0.0003 instead of 0.481.

Additionally, we see that Pet2 can solve models with millions of states and various difficult graph structures within a minute. Thus, we conclude that both our algorithm and implementation scale well.

Comparison to Pet1. When solving Markov chains or MDPs, Pet2 still uses algorithms that can handle SGs. This generality comes with some overhead, for example because data structures for tracking ownership of states are not necessary in MDPs. However, a score of 1.07[85+8/1] and Fig. 2 (left) show that PE in both versions of Pet performs remarkably similar, with Pet2 even slightly

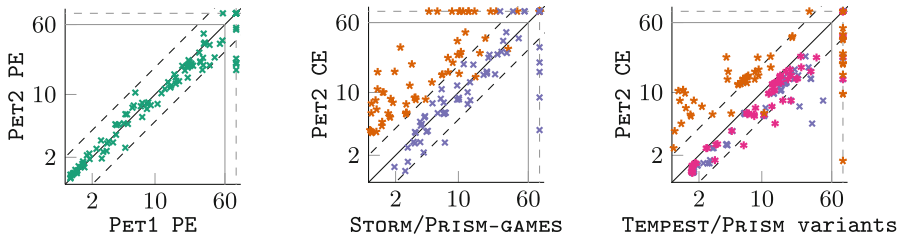

Fig. 2. Comparison of Pet2 to other tools. From left to right we compare Pet2-PE and Pet1-PE, Pet2-CE on MDP and MC with Storm (✱) and Prism-games (✖), and finally Pet2-CE on SG with Tempest (✱), Prism-games (✖), and Prism-ext (✱). A point (x, y) denotes that tool X and Pet2 needed x and y seconds, respectively. If a point is above/below the diagonal, tool X is faster/slower. Plots are on logarithmic scale, dashed diagonals indicate that one tool is twice as fast. Timeouts are pushed to the orthogonal dashed line.

faster. We conclude that the improvements in the implementation make up for the algorithmic overhead.

Comparison to other Tools. It is well known that the *structure* of a model is *the* determining factor for the relative performance of different algorithmic approaches, see e.g. [5,18,29], in particular far more than the number of states or transitions. (This is also supported by our comparison of CE and PE in [32, App. F.3], sometimes showing order of magnitude advantages in either direction.) Thus, instead of comparing tools as a whole, we compare matching algorithmic approaches (CE and PE based value/interval iteration) to assess only the impact of different implementations. For similar reasons, here we only compare the explicit engines and not symbolic or hybrid approaches (results on the latter are provided in [32, App. F.3]).

Comparing PE approaches, Pet2 outperforms the only competitor Storm with a score of 0.3[27+29/1] (Pet2 solves more than twice the number of models); see [32, App. F.3] for details. For CE algorithms, across all instances where both tools are applicable, the score of Pet2 against the Java-based tools Prism-games and Prism-ext is 1.3[104+10/4] and 1.3[53+4/0], respectively, while against the C++ tools Storm and Tempest it achieves 0.3[69+0/12] and 0.4[54+13/1], respectively. For a more detailed comparison, Fig. 2 (middle) compares the CE algorithms of Pet2 with those in Storm and Prism-games on Markov chains and MDPs. Here, as expected, Storm outperforms other tools, at least partially due to performance differences of C++ and Java. However, Pet2 performs favourably against the state-of-the-art tool Prism-games. This shows that our first, generic implementation of the CE algorithm for SG is comparable to established tools even on Markov chains and MDPs. Finally, Fig. 2 (right) compares Pet2 with the other tools on SGs, namely Prism-games, Tempest, and Prism-ext. Recall that only Prism-ext uses a sound algorithm, while the other tools use an unsound stopping criterion and thus require less work. Nonetheless, Pet2 often outperforms the other tools (even Tempest, which builds on the highly optimized Storm), making it the most viable tool on SGs not only because of soundness, but also because of performance.

Finally, to (superficially) evaluate how much objective-specific optimizations yield, we implemented viewing reachability objectives as "trivial" mean payoff objective, i.e. goal states are set to be absorbing with reward 1, and all others with reward zero. This modified query is then passed to our generic mean payoff algorithm. Notably, even then Pet2 slightly outperforms Prism-games solving the reachability objective directly (1.1[98+8/10]), and in turn the dedicated reachability approach of Pet2 "only" scores 1.2[106+8/0] against this variant.

7 Conclusion

We presented Pet2, the first tool implementing a sound and efficient approach for solving SGs with objectives of the type reachability/safety and mean payoff. Our experimental evaluation shows that (i) it is sound, while other tools

indeed return wrong answers in practice, (ii) it offers the most efficient partial-exploration based algorithm, and (iii) it is the most viable tool on SGs.

For future work, there is still a lot of room for heuristics and engineering improvements, for example adaptively choosing internal parameters, more efficient tracking and handling of SEC-candidates, using topological order of updates in VI, improved pre-computation for mean payoff, etc. Additionally, support for total reward is planned; however, as described in [32, App. B], this requires using ideas such as optimistic value iteration [5,19] in order to be reasonably efficient.

References

1. Amir, R.: Stochastic games in economics and related fields: an overview. In: Neyman, A., Sorin, S. (eds.) Stochastic Games and Applications. NATO Science Series, vol. 570, pp. 455–470. Springer, Dordrecht (2003). https://doi.org/10.1007/978-94-010-0189-2_30
2. Andersson, D., Miltersen, P.B.: The complexity of solving stochastic games on graphs. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 112–121. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10631-6_13
3. Andriushchenko, R., et al.: Tools at the frontiers of quantitative verification: QComp 2023 competition report. TOOLympics (to appear)
4. Ashok, P., Chatterjee, K., Daca, P., Křetínský, J., Meggendorfer, T.: Value iteration for long-run average reward in Markov decision processes. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 201–221. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_10
5. Azeem, M., Evangelidis, A., Kretínský, J., Slivinskiy, A., Weininger, M.: Optimistic and topological value iteration for simple stochastic games. In: Bouajjani, A., Holík, L., Wu, Z. (eds.) ATVA 2022. LNCS, vol. 13505, pp. 285–302. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-19992-9_18
6. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press, Cambridge (2008)
7. Bellomo, N., Delitala, M.: From the mathematical kinetic, and stochastic game theory to modelling mutations, onset, progression and immune competition of cancer cells. Phys. Life Rev. **5**(4), 183–206 (2008). https://doi.org/10.1016/j.plrev.2008.07.001
8. Brázdil, T., et al.: Verification of Markov decision processes using learning algorithms. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 98–114. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11936-6_8
9. Chatterjee, K., Henzinger, T.A.: Value iteration. In: Grumberg, O., Veith, H. (eds.) 25 Years of Model Checking. LNCS, vol. 5000, pp. 107–138. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69850-0_7
10. Chatterjee, K., Meggendorfer, T., Saona, R., Svoboda, J.: Faster algorithm for turn-based stochastic games with bounded treewidth. In: SODA, pp. 4590–4605. SIAM (2023). https://doi.org/10.1137/1.9781611977554.CH173
11. Chen, T., Forejt, V., Kwiatkowska, M.Z., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. Formal Methods Syst. Des. **43**(1), 61–92 (2013). https://doi.org/10.1007/s10703-013-0183-7

12. Condon, A.: On algorithms for simple stochastic games. In: Advances In Computational Complexity Theory. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 13, pp. 51–71. DIMACS/AMS (1990)
13. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J. ACM **42**(4), 857–907 (1995)
14. De Alfaro, L.: Formal verification of probabilistic systems. Ph.D. thesis, Stanford University (1997)
15. 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
16. Gillette, D.: Stochastic games with zero stop probabilities. Contrib. Theory Games **3**, 179–187 (1957)
17. Haddad, S., Monmege, B.: Interval iteration algorithm for MDPs and IMDPs. Theor. Comput. Sci. **735**, 111–131 (2018)
18. Hartmanns, A., Junges, S., Quatmann, T., Weininger, M.: A practitioner's guide to MDP model checking algorithms. In: Sankaranarayanan, S., Sharygina, N. (eds.) TACAS 2023. LNCS, vol. 13993, pp. 469–488. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30823-9_24
19. Hartmanns, A., Kaminski, B.L.: Optimistic value iteration. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12225, pp. 488–511. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53291-8_26
20. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The quantitative verification benchmark set. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 344–350. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17462-0_20
21. Hensel, C., Junges, S., Katoen, J., 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
22. Johnson, D.S.: The NP-completeness column: finding needles in haystacks. ACM Trans. Algorithms **3**(2), 24 (2007). https://doi.org/10.1145/1240233.1240247
23. Kretínský, J., Meggendorfer, T.: Of cores: a partial-exploration framework for Markov decision processes. Log. Methods Comput. Sci. **16**(4) (2020). https://lmcs.episciences.org/6833
24. Kretínský, J., Meggendorfer, T., Weininger, M.: Stopping criteria for value iteration on stochastic games with quantitative objectives. In: LICS, pp. 1–14 (2023). https://doi.org/10.1109/LICS56636.2023.10175771
25. Kretínský, J., Meggendorfer, T., Weininger, M.: Stopping criteria for value iteration on stochastic games with quantitative objectives. CoRR abs/2304.09930 (2023)
26. Kretínský, J., Ramneantu, E., Slivinskiy, A., Weininger, M.: Comparison of algorithms for simple stochastic games. Inf. Comput. **289**(Part), 104885 (2022). https://doi.org/10.1016/j.ic.2022.104885
27. Kwiatkowska, M., Norman, G., Parker, D., Santos, G.: PRISM-games 3.0: stochastic game verification with concurrency, equilibria and time. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12225, pp. 475–487. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53291-8_25
28. Lemire, D., Kai, G.S.Y., Kaser, O.: Consistently faster and smaller compressed bitmaps with roaring. SPE **46**(11), 1547–1569 (2016)
29. Meggendorfer, T.: PET - a partial exploration tool for probabilistic verification. In: Bouajjani, A., Holík, L., Wu, Z. (eds.) ATVA 2022. LNCS, vol. 13505, pp. 320–326. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-19992-9_20

30. Meggendorfer, T., Weininger, M.: Partial exploration tool gitlab. https://gitlab.lrz.de/i7/partial-exploration
31. Meggendorfer, T., Weininger, M.: Artifact for "Partial Exploration Tool 2.0" (2024). https://doi.org/10.5281/zenodo.10927672
32. Meggendorfer, T., Weininger, M.: Playing games with your pet: extending the partial exploration tool to stochastic games. CoRR abs/2405.03885 (2024). https://doi.org/10.48550/ARXIV.2405.03885
33. Pranger, S., Könighofer, B., Posch, L., Bloem, R.: TEMPEST - synthesis tool for reactive systems and shields in probabilistic environments. In: Hou, Z., Ganesh, V. (eds.) ATVA 2021. LNCS, vol. 12971, pp. 222–228. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88885-5_15
34. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Statistics, Wiley (1994). https://doi.org/10.1002/9780470316887
35. Roy, S., Ellis, C., Shiva, S.G., Dasgupta, D., Shandilya, V., Wu, Q.: A survey of game theory as applied to network security. In: HICSS, pp. 1–10. IEEE Computer Society (2010). https://doi.org/10.1109/HICSS.2010.35
36. Weininger, M.: Solving Stochastic Games Reliably. Ph.D. thesis, Technical University of Munich, Germany (2022). https://mediatum.ub.tum.de/node?id=1661588