Of Cores: A Partial-Exploration Framework for Markov Decision Processes

Jan Křetínský
Technical University of Munich, Germany
jan.kretinsky@in.tum.de

Tobias Meggendorfer
Technical University of Munich, Germany
tobias.meggendorfer@in.tum.de

Abstract

We introduce a framework for approximate analysis of Markov decision processes (MDP) with bounded-, unbounded-, and infinite-horizon properties. The main idea is to identify a "core" of an MDP, i.e., a subsystem where we provably remain with high probability, and to avoid computation on the less relevant rest of the state space. Although we identify the core using simulations and statistical techniques, it allows for rigorous error bounds in the analysis. Consequently, we obtain efficient analysis algorithms based on partial exploration for various settings, including the challenging case of strongly connected systems.

2012 ACM Subject Classification Theory of computation → Verification by model checking; Theory of computation → Random walks and Markov chains

Keywords and phrases Markov Decision Processes, Reachability, Approximation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.5

Related Version A full version of the paper is available at `https://arxiv.org/abs/1906.06931`.

Funding This work has been partially supported by the Czech Science Foundation grant No. 18-11193S and the German Research Foundation (DFG) project KR 4890/2 "Statistical Unbounded Verification".

1 Introduction

Markov decision processes (MDP) are a well established formalism for modelling, analysis and optimization of probabilistic systems with non-determinism, with a large range of application domains [18]. Classical objectives such as reachability of a given state or the long-run average reward (mean payoff) can be solved by a variety of approaches. In theory, the most suitable approach is linear programming as it provides exact answers (rational numbers with no representation imprecision) in polynomial time. However, in practice for systems with more than a few thousand states, linear programming is not very usable, see, e.g., [2]. As an alternative, one can apply dynamic programming, typically value iteration (VI) [4], the default method in the probabilistic model checkers PRISM [13] and Storm [9].

Despite better practical scalability of VI, systems with more than a few million states still remain out of reach of the analysis not only because of time-outs, but now also memory-outs, see, e.g., [6]. There are various heuristics designed to deal with so large state spaces, including abstractions, e.g., [8, 11], or a dual approach based on restricting the analysis to a part of the state space. Examples of the latter approach are asynchronous VI in probabilistic planning, e.g., [17], or projections in approximate dynamic programming, e.g., [5]. In both, only a certain subset of states is considered for analysis, leading to speed ups in orders of magnitude. These are best-effort solutions, which can only guarantee convergence to the true result in the limit, with no error bounds at any finite time. Surprisingly, this was the case with the

standard VI as well until recently, when an error bound (and thus a stopping criterion) was given independently in [10, 6]. The error bound follows from the under- and (newly obtained) over-approximations converging to the true value. This resulted not only in error bounds on VI, but opened the door to error bounds for other techniques, including those where even convergence is not guaranteed. For instance, while VI iteratively approximates the value of all states, the above-mentioned *asynchronous VI* evaluates states at different paces. Thus convergence is often unclear and even the rate of convergence is unknown and very hard to analyze. However, it is not too hard to extend the error bound technique for VI to asynchronous VI. A prime example is the modification of BRTDP [17] to reachability [6] with error bounds. These ideas are further developed for, e.g., settings with long-run average reward [2] or continuous time [1].

While these solutions are efficient, they are ad-hoc, sharing the idea of *simulation / learning-based partial exploration* of the system, but not the correctness proof. In this paper, we build the foundations for designing such frameworks and provide a new perspective on these approaches, leading to algorithms for settings where previous ideas cannot apply.

The previous algorithms use (i) simulations to explore the state space and (ii) planning heuristics and machine learning to analyze the experience and to bias further simulations to areas that seem more relevant for the analysis of the given property (e.g., reaching a state s_{42}), where (iii) the exact VI computation takes place and yields results with a guaranteed error bound. In contrast, this paper identifies a general concept of a "*core*" of the MDP, independently of the particular objective (which states to reach) and, to a certain extent, even of the type of property (reachability, mean payoff, linear temporal logic formulae, etc.). This core intuitively consists of states that are important for the analysis of the MDP, whereas the remaining parts of the state space affect the result only negligibly. To this end, the defining property of a core is that the system stays within the core with high probability.

There are several advantages of cores, compared to the tailored techniques. Since the core is agnostic of any particular property, it can be *re-used* for multiple queries. Thus, the repetitive effort spent by the simulations and heuristics to explore the relevant parts of the state space by the previous algorithms can be saved. Furthermore, identifying the core can serve to better *understand* the typical behaviour of the system. Indeed, the core is typically a lot smaller than the system (and thus more amenable to understand) and only contains the more likely behaviours, even for real-world models as shown in the experimental evaluation. Finally, the general concept of core provides a *unified* argument for the correctness of the previous algorithms since – implicitly – they gradually construct a core. This abstract view thus allows for easier development of further partial-exploration techniques within this framework.

More importantly, making the notion of core explicit naturally leads us to identify a new standpoint and approach for the more complicated case of strongly connected systems, where the previous algorithms as well as cores cannot help. In technical terms, minimal cores are closed under end components. Consequently, the minimal core for a strongly connected system is the whole system. And indeed, it is impossible to give guarantees on infinite-horizon behaviour whenever a single state is ignored. In order to provide *some* feasible analysis for this case, we introduce the n-step core. It is defined by the system staying there with high probability for *some* time. However, instead of n-step analysis, we suggest to observe the "stability" of the core, i.e. the tendency of the probability to leave this core if longer and longer runs are considered. We shall argue that this yields (i) rigorous bounds for N-step analysis for $N \gg n$ more efficiently than a direct N-step analysis, and (ii) finer information on the "long run" behaviour (for different lengths) than the summary for the

infinite run, which, n.b., never occurs in reality. This opens the door towards a rigorous analysis of "typical" behaviour of the system, with many possible applications in the design and interpretation of complex systems.

Our contribution can be summarized as follows:

- We introduce the notion of core, study its basic properties, in its light re-interpret previous results in a unified way, and discuss its advantages.
- We stipulate a new view on long-run properties as rather corresponding to long runs than an infinite one. Then a modified version of cores allows for an efficient analysis of strongly connected systems, where other partial-exploration techniques necessarily fail.
- We show how these modified cores can aid in design and interpretation of systems.
- We provide efficient algorithms for computing both types of cores and evaluate them on several examples.

2 Preliminaries

In this section, we recall basics of probabilistic systems and set up the notation. We assume familiarity with the central ideas of measure theory. As usual, $\mathbb{N}$ and $\mathbb{R}$ refers to the (positive) natural numbers and real numbers, respectively. For any set S, we use $\overline{S}$ to denote its complement. A *probability distribution* on a finite set X is a mapping $p : X \to [0,1]$, such that $\sum_{x\in X} p(x) = 1$. Its *support* is denoted by $\mathrm{supp}(p) = \{x \in X \mid p(x) > 0\}$. $\mathcal{D}(X)$ denotes the set of all probability distributions on X. An event happens *almost surely* (a.s.) if it happens with probability 1.

▶ **Definition 1.** *A* Markov chain (MC) *is a tuple* $\mathsf{M} = (S, s_0, \delta)$*, where* S *is a countable set of* states*,* $s_0 \in S$ *is the* initial *state, and* $\delta : S \to \mathcal{D}(S)$ *is a* transition function *that for each state* s *yields a probability distribution over successor states.*

▶ **Definition 2.** *A* Markov decision process (MDP) *is a tuple of the form* $\mathcal{M} = (S, s_0, A, \mathsf{Av}, \Delta)$*, where* S *is a finite set of* states*,* $s_0 \in S$ *is the* initial *state,* A *is a finite set of* actions*,* $\mathsf{Av} : S \to 2^A \setminus \{\emptyset\}$ *assigns to every state a non-empty set of* available actions*, and* $\Delta : S \times A \to \mathcal{D}(S)$ *is a* transition function *that for each state* s *and (available) action* $a \in \mathsf{Av}(s)$ *yields a probability distribution over successor states. Furthermore, we assume w.l.o.g. that actions are unique for each state, i.e.* $\mathsf{Av}(s) \cap \mathsf{Av}(s') = \emptyset$ *for* $s \neq s'$ *(which can be achieved by replacing* A *with* $S \times A$ *and adapting* Av *and* Δ *appropriately).*

For ease of notation, we overload functions mapping to distributions $f : Y \to \mathcal{D}(X)$ by $f : Y \times X \to [0,1]$, where $f(y,x) := f(y)(x)$. For example, instead of $\delta(s)(s')$ and $\Delta(s,a)(s')$ we write $\delta(s,s')$ and $\Delta(s,a,s')$, respectively.

An *infinite path* ρ in a Markov chain is an infinite sequence $\rho = s_0 s_1 \ldots \in S^\omega$, such that for every $i \in \mathbb{N}$ we have that $\delta(s_i, s_{i+1}) > 0$. A *finite path* (or *history*) $\varrho = s_0 s_1 \ldots s_n \in S^*$ is a finite prefix of an infinite path. Similarly, an *infinite path* in an MDP is some infinite sequence $\rho = s_0 a_0 s_1 a_1 \ldots \in (S \times A)^\omega$, such that for every $i \in \mathbb{N}$, $a_i \in \mathsf{Av}(s_i)$ and $\Delta(s_i, a_i, s_{i+1}) > 0$. *Finite paths* ϱ are defined analogously as elements of $(S \times A)^* \times S$. We use ρ_i and ϱ_i to refer to the i-th state in the given (in)finite path.

A *strategy* on an MDP is a function $\pi : (S \times A)^* \times S \to \mathcal{D}(A)$, which given a finite path $\varrho = s_0 a_0 s_1 a_1 \ldots s_n$ yields a probability distribution $\pi(\varrho) \in \mathcal{D}(\mathsf{Av}(s_n))$ on the actions to be taken next. We denote the set of all strategies of an MDP by Π. Fixing any strategy π induces a Markov chain $\mathcal{M}^\pi = (S^\pi, s_0^\pi, \delta^\pi)$, where the states are given by $S^\pi = (S \times A)^* \times S$ and, for some state $\varrho = s_0 a_0 \ldots s_n \in S^\pi$, the successor distribution is defined as $\delta^\pi(\varrho, \varrho a_{n+1} s_{n+1}) = \pi(\varrho, a_{n+1}) \cdot \Delta(s_n, a_{n+1}, s_{n+1})$.

Any Markov chain M induces a unique measure $\mathbb{P}^{\mathsf{M}}$ over infinite paths [3, p. 758]. Assuming we fixed some MDP $\mathcal{M}$, we use $\mathbb{P}^{\pi}_{s}$ to refer to the probability measure induced by the Markov chain $\mathcal{M}^{\pi}$ with initial state s. Whenever π or s are clear from the context, we may omit them, in particular, $\mathbb{P}^{\pi}$ refers to $\mathbb{P}^{\pi}_{s_0}$. See [18, Sec. 2.1.6] for further details. For a given MDP $\mathcal{M}$ and measurable event E, we use the shorthand $\mathbb{P}^{\max}[E] := \sup_{\pi\in\Pi} \mathbb{P}^{\pi}[E]$ and $\mathbb{P}^{\max}_{s}[E] := \sup_{\pi\in\Pi} \mathbb{P}^{\pi}_{s}[E]$ to refer to the maximal probability of E over all strategies (starting in s). Analogously, $\mathbb{P}^{\min}[E]$ and $\mathbb{P}^{\min}_{s}[E]$ refer to the respective minimal probabilities.

A pair (T, B), where $\emptyset \neq T \subseteq S$ and $\emptyset \neq B \subseteq \bigcup_{s\in T} \mathsf{Av}(s)$, is an *end component* of an MDP $\mathcal{M}$ if (i) for all $s \in T, a \in B \cap \mathsf{Av}(s)$ we have $\mathrm{supp}(\Delta(s,a)) \subseteq T$, and (ii) for all $s, s' \in T$ there is a finite path $\varrho = sa_0 \dots a_n s' \in (T \times B)^* \times T$, i.e. the path stays inside T and only uses actions in B. Intuitively, an end component describes a set of states for which a particular strategy exists such that all possible paths remain inside these states. By abuse of notation, we identify an end component with the respective set of states, e.g., $s \in E = (T, B)$ means $s \in T$. An end component (T, B) is a *maximal end component (MEC)* if there is no other end component (T', B') such that $T \subseteq T'$ and $B \subseteq B'$. The set of MECs of an MDP $\mathcal{M}$ is denoted by $\mathsf{MEC}(\mathcal{M})$ and can be obtained in polynomial time [7].

In the following, we will primarily deal with unbounded and bounded variants of *reachability* queries. Essentially, for a given MDP and set of states, the task is to determine the maximal probability of reaching them, potentially within a certain number of steps. Technically, we are interested in determining $\mathbb{P}^{\max}[\Diamond T]$ and $\mathbb{P}^{\max}[\Diamond^{\leq n} T]$, where T is the set of target states and $\Diamond T$ ($\Diamond^{\leq n} T$) refers to the measurable set of runs that visit T at least once (in the first n steps). The dual operators $\Box T$ and $\Box^{\leq n} T$ refer to the set of runs which remain inside T forever or for the first n steps, respectively. See [3, Sec. 10.1.1] for further details. Our techniques are easily extendable to other related objectives like *long run average reward* (*mean payoff*) [18], *LTL formulae* or ω-regular objectives [3]. We briefly comment on this in Section 3.3.

We are interested in finding approximate solutions efficiently, i.e. trading precision for speed of computation. In our case, "approximate" means ε-optimal for some given precision $\varepsilon > 0$, i.e. the value we determine has an absolute error of less than ε. For example, given a reachability query $\mathbb{P}^{\max}[\Diamond T]$ and precision ε, we are interested in finding a value v with $|\mathbb{P}^{\max}[\Diamond T] - v| < \varepsilon$.

3 The Core Idea

In this section, we present the novel concept of *cores*, inspired by the approach of [6], where a specific reachability query was answered approximately through heuristic based methods. Omitted proofs can be found in [12, App. A.1]. We first establish a running example to motivate our work and explain the difference to previous approaches.

Consider a flight of an air plane. The controller, e.g. the pilot and the flight computer, can take many decisions to control the plane. The system as a whole can be in many different states. One may be interested in the maximal probability of arriving safely. This intuitively describes how likely it is to arrive, assuming that the pilot acts optimally and the computer is bug-free. Of course, this probability may be less than 100%, since some components may fail even under optimal conditions. See Figure 1 for a simplified MDP modelling this example.

The key observation in [6] is that some extreme situations may be very unlikely and we can simply assume the worst case for them without losing too much precision. This allows us to completely ignore these situations. For example, consider the unlikely event of hazardous bit flips during the flight due to cosmic radiation. This event might eventually lead to a

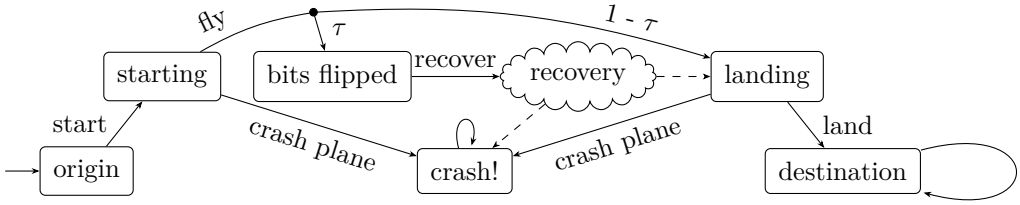


Figure 1 A simplified model of a flight, where $\tau = 10^{-10}$ is the probability of potentially hazardous bit flips occurring during the flight. The "recovery" node represents a complex recovery procedure, comprising many states.

crash or it might have no influence on the evolution of the system at all. Since this event is so unlikely to occur, we can simply assume that it always leads to a crash and still get a very precise result. Consequently, we do not need to explore the corresponding part of the state space (the "recovery" part), saving resources. As shown in the experimental evaluation in Section 5, many real-world models indeed exhibit a similar structure.

In [6], the state space was explored relative to a particular reachability objective, storing upper and lower bounds on each state for the objective in consideration. We make use of the same principle idea, but approach it from a different perspective, agnostic of any objective. We are interested in finding *all* relevant states of the system, i.e. all states which are reasonably likely to be reached. Such a set of states is an *intrinsic property* of the system, and we show that this set is both sufficient and necessary to answer *any* non-trivial reachability query ε-precisely. In particular, once computed, this set can be reused for multiple queries.

3.1 Infinite-Horizon Cores

First, we define the notion of an *ε-core*. Intuitively, an ε-core is a set of states which can only be exited with probability less than ε.

▶ **Definition 3** (Core). *Let $\mathcal{M}$ be an MDP and $\varepsilon > 0$. A set $S_\varepsilon \subseteq S$ is an* ε-core *if $\mathbb{P}^{\max}[\Diamond \overline{S_\varepsilon}] < \varepsilon$, i.e. the probability of ever exiting S_ε is smaller than ε.*

When ε is clear from the context, we may refer to an ε-core by "core". Observe that the core condition is equivalent to $\mathbb{P}^{\min}[\Box S_\varepsilon] \geq 1 - \varepsilon$.

The set of all states S trivially is a core for any ε. Naturally, we are interested in finding a core which is as small as possible, which we call a *minimal core*.

▶ **Definition 4** (Minimal Core). *Let $\mathcal{M}$ be an MDP and $\varepsilon > 0$. $S^*_\varepsilon \subseteq S$ is a* minimal ε-core *if it is inclusion minimal, i.e. S^*_ε is an ε-core and there exists no ε-core $S'_\varepsilon \subsetneq S^*_\varepsilon$.*

When ε is clear from the context, we may refer to a minimal ε-core by "minimal core". In the running example, a minimal core for $\varepsilon = 10^{-6}$ would contain all states except the "bit flipped" state and the "recovery" subsystem, since they are reached only with probability $\tau < \varepsilon$.

▶ Remark 5. We note that this idea may seem similar to the one of [19], but is subtly different. In that work, the authors consider a classical reachability problem using value iteration. They approximate the exit probability of a *fixed* set $S_?$ to bound the error on the computed reachability.

In the following, we derive basic properties of cores, show how to efficiently construct them, and relate them to the approaches of [2, 6].

First, we observe that finding a core of a given size (for a non-trivial ε) is NP-complete.

▶ **Theorem 6.** *For $0 < \varepsilon < \frac{1}{4}$, $\{(\mathcal{M}, k) \mid \mathcal{M}$ has an ε-core of size $k\}$ is NP-complete.*

Observe that this result only implies that finding minimal cores is hard. In the following section, we introduce a learning-based approach which quickly identifies reasonably sized cores.

▶ **Theorem 7.** *Let $\mathcal{M}$ be an MDP and $\varepsilon > 0$. A set $S_\varepsilon \subseteq S$ is an ε-core of $\mathcal{M}$ if and only if for every subset of states $R \subseteq S$ we have that $0 \leq \mathbb{P}^{\max}[\Diamond R] - \mathbb{P}^{\max}[\Diamond(R \cap S_\varepsilon) \cap \Box S_\varepsilon] < \varepsilon$.*

This theorem shows that for any reachability objective R, we can determine $\mathbb{P}^{\max}[\Diamond R]$ up to ε precision by determining the reachability of R on the sub-model induced by any ε-core, i.e. by only considering runs which remain inside S_ε. This also shows that, in general, cores are necessary to determine reachability up to precision ε.

We emphasize that this does *not* imply that identifying a core is necessary for all queries. For example, we have that $\mathbb{P}^{\max}_{s_0}[\Diamond\{s_0\}] = 1$ even without constructing any core. Nevertheless, for any non-trivial property, i.e. a reachability query with value less than $1-\varepsilon$, a computation restricted to a subset which does not satisfy the core property cannot give ε-guarantees on its results – only lower bounds can be proven. Thus, a set of states satisfying the core property has to be considered for non-trivial properties. In particular, the approach of [6] implicitly builds a core for such properties.

Of course, one could simply construct the whole state set S for the computation, which trivially satisfies the core condition. But, using the methods presented in the following section, we can efficiently identify a considerably smaller core. In particular, we observe in Section 5 that for some models we are able to identify a very small core orders of magnitude faster than the construction of the state set S, speeding up subsequent computations drastically.

3.2 Learning a Core

We introduce a sampling based algorithm which builds a core. In the interest of space, we only briefly describe the algorithm. Further discussion can be found in [12, App. A.2] and [6]. The algorithm is structurally very similar to the one presented in [6]. Nevertheless, we present it explicitly here since (i) it is significantly simpler and (ii) we introduce modifications later on.

We assume that the model is described by an initial state and a successor function, yielding all possible actions and the resulting distribution over successor states. This allows us to only construct a small fraction of the state space and achieve sub-linear runtime for some models.

During the execution of the algorithm, the system is traversed by following the successor function, starting from the initial state. Each state encountered is stored in a set of *explored* states, all other, not yet visited states are *unexplored*. Unexplored successors of explored states are called *partially explored*. Furthermore, the algorithm stores for each (explored) state s an upper bound $U(s)$ on the probability of reaching some unexplored state starting from s. The algorithm gradually grows the set of explored states and simultaneously updates these upper bounds, until the desired threshold is achieved in the initial state, i.e. $U(s_0) < \varepsilon$, and thus the set of explored states provably satisfies the core property. In particular, the upper bound is updated by sampling a path according to SamplePath and back-propagating the values along that path using Bellman backups.

SamplePath samples paths following some heuristic. In particular, it does not have to follow the transition probabilities given by the successor function. For example, a successor might be sampled with probability proportional to its upper bound times the transition

Algorithm 1 LearnCore.

Input: MDP $\mathcal{M}$, precision $\varepsilon > 0$, upper bounds U, state set S_ε with $s_0 \in S_\varepsilon$
Output: S_ε s.t. S_ε is an ε-core

```
while U(s_0) ≥ ε do
    ϱ ← SamplePath(s_0, U)                          ▷ Generate path
    S_ε ← S_ε ∪ ϱ                                   ▷ Expand core
    UpdateECs(S_ε, U)
    for s in ϱ in reverse order do                  ▷ Back-propagate values
        U(s) ← max_{a∈A(s)} Σ_{s'∈S} Δ(s,a,s') · U(s')
return S_ε
```

probability. The intuition behind this approach is that all states which are unlikely to be reached are not relevant and hence do not need to be included in the core. By trying to reach unexplored states the algorithm likely only reaches states which indeed are important.

UpdateECs identifies MECs of the currently explored sub-system and "collapses" them into a single representative state. This is necessary to ensure convergence of the upper bounds to the correct value – technically this process removes spurious fixed points of U.

For (a.s.) termination, we only require that the sampling heuristic is "(almost surely) fair". This means that (i) any partially explored state is reached eventually (a.s.), in order to explore a sufficient part of the state space, and (ii) any explored state with $U(s) > 0$ is visited infinitely often (a.s.), in order to back-propagate values accordingly. Further, we require that the initial upper bounds are consistent with the given state set, i.e. $U(s) \geq \mathbb{P}_s^{\max}[\Diamond \overline{S_\varepsilon}]$. This is trivially satisfied by $U(\cdot) = 1$. Note that in contrast to [6], the set whose reachability we approximate dynamically changes and, further, only upper bounds are computed.

▶ **Theorem 8.** *Algorithm 1 is correct and terminates (a.s.) if* SamplePath *is (a.s.) fair and the given upper bounds U are consistent with the given set S_ε.*

Proof Sketch. *Correctness*: By assumption $U(s)$ initially is a correct upper bound for the "escape" probability, i.e. $U(s) \geq \mathbb{P}_s^{\max}[\Diamond \overline{S_\varepsilon}]$. Each update (a Bellman backup) preserves correctness, independent of the sampled path. Hence, if $U(s_0) < \varepsilon$, we have $\mathbb{P}_s^{\max}[\Diamond \overline{S_\varepsilon}] < \varepsilon$.

Termination: As we assumed that SamplePath is (a.s.) fair, eventually (a.s.) the whole model will be explored, and all MECs will be collapsed by UpdateECs. Then, all states are visited infinitely often (a.s.), and thus all upper bounds will eventually converge to 0. ◀

As Algorithm 1 is correct and terminates for any faithful upper bounds and initial state set, we can restart the algorithm and interleave it with other approaches refining the upper bounds. For example, one could periodically update the upper bounds using, e.g., strategy iteration. Further, we can reuse the computed upper bounds and state set to compute a core for a tighter precision.

3.3 Using Cores for Verification

We explain how a core can be used for verification and how our approach differs from existing ones. Clearly, we can compute reachability or safety objectives on a given core ε-precisely. In this case, our approach is not too different from the one in [6]. Yet, we argue that our approach yields a stronger result. Due to cores being an intrinsic object, we are able to reuse and adapt this idea easily to many other objectives. Observe that a dedicated adaption may still yield slightly better performance, but requires significantly more work. For example, see [2] for an adaption to mean payoff.

To see how we can connect our idea to mean payoff, we briefly explain this objective and then recall an observation of [2]. First, rational rewards are assigned to each state, which are obtained on each visit. Then, the mean payoff of a particular run is the limit average reward obtained from the visited states. The mean payoff under a particular strategy then is obtained by integrating over the set of all runs. As mentioned by [2], a mean payoff objective can be decomposed into a separate analysis of each (explored) MEC and a (weighted) reachability query

$$\text{optimal mean payoff} = \sup_{\pi \in \Pi} \sum\nolimits_{M \in \mathsf{MEC}(\mathcal{M})} \text{mean payoff of } \pi \text{ in } M \cdot \mathbb{P}^{\pi}\left[\Diamond \Box M\right].$$

Since we can bound the reachability on unexplored MECs by the core property, we can easily bound the error on the computed mean payoff (assuming we know an a-priori lower and upper bound on the reward function). Consequently, we can approximate the optimal mean payoff by only analysing the corresponding core.

Similarly, LTL queries and parity objectives can be answered by a decomposition into analysis of MECs and their reachability. Intuitively, given a MEC one can decide whether the MEC is "winning" or "losing" for these objectives. The overall probability of satisfying the objective then equals the probability of reaching a winning MEC [3]. Again, we can bound the reachability of unexplored MECs and thus the error we incur when only analysing the core.

In general, many verification tasks can be decomposed into a reachability query and analysis of specific parts of the system. Since our framework is agnostic of the verification task in question, it can be transparently plugged in to obtain significant speed-ups.

We highlight that our approach is directly applicable to models with infinite state space, since finite cores still may exist for these models.

4 Beyond Infinite Horizon

In the previous section, we have seen that MECs play an essential role for many objectives. Hence, we study the interplay between cores and MECs.

▶ **Proposition 9.** *Let $M \in \mathsf{MEC}(\mathcal{M})$ be a MEC. If there is a state $s \in M$ with $\mathbb{P}^{\max}[\Diamond\{s\}] \geq \varepsilon$ then $M \subseteq S_\varepsilon$ for every ε-core S_ε.*

Proof. Recall that for $s, s' \in M$, we have $\mathbb{P}^{\max}_s[\Diamond\{s'\}] = 1$, thus $\mathbb{P}^{\max}[\Diamond\{s\}] = \mathbb{P}^{\max}[\Diamond\{s'\}] \geq \varepsilon$ and thus $s' \in S_\varepsilon$. ◀

This implies that sufficiently reachable MECs always need to be contained in a core *entirely*. Many models comprise only a few or even a single MEC, e.g., restarting protocols like mutual exclusion or biochemical models of reversible reactions. Together with the result of Theorem 7, i.e. constructing a core is necessary for ε-precise answers, this shows that in general we cannot hope for any reduction in state space, even when only requiring ε-optimal solutions for *any* of the discussed properties. In particular, the approach of [6] necessarily has to explore the full model. Yet, real-world models often exhibit a particular structure, with many states only being visited infrequently. Since we necessarily have to give up on something to obtain further savings, we propose an extension of our idea, motivated by a modification of our running example.

Instead of a one-way trip, consider the plane going back and forth between the origin and the destination, as shown in Figure 2. Clearly, the plane eventually will suffer from a bit flip, *independently* of the strategy. Furthermore, assuming that there is a non-zero probability of not being able to recover from the error, the plane will eventually crash.

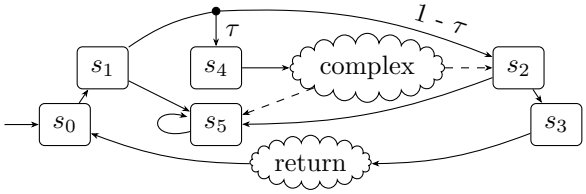


Figure 2 An adaptation of the model from Figure 1, with an added return trip, represented by the "return" node. State and action labels have been omitted in the interest of space.

We make two observations. First, any core needs to contain at least parts of the recovery sub-system, since it is reached with probability 1. Thus, this (complex) sub-system has to be constructed. Second, the witness strategy is meaningless, since any strategy is optimal – the crash cannot be avoided in the long run. In particular, deliberately crashing the plane has the same long run performance as flying it "optimally". In practice, we often actually are interested in the performance of such a model for a long, but not necessarily infinite horizon.

To this end, one could compute the step bounded variants of the objectives, but this incurs several problems: (i) choosing a sensible step bound n, (ii) computational overhead (a precise computation has a worst-case complexity of $|\Delta| \cdot n$ even for reachability), and (iii) all states reachable within n steps have to be constructed (which equals the whole state space for practically all models and reasonable choices of n). In the following, we present a different approach to this problem, again based on the idea of cores.

4.1 Finite-Horizon Cores

We introduce *finite-horizon cores*, which are completely analogous to (infinite-horizon) cores, only with a step bound attached to them.

▶ **Definition 10** (Finite-Horizon Core). *Let $\mathcal{M}$ be an MDP, $\varepsilon > 0$, and $n \in \mathbb{N}$. A set $S_{\varepsilon,n} \subseteq S$ is an* n-step ε-core *if $\mathbb{P}^{\max}[\Diamond^{\leq n}\overline{S_{\varepsilon,n}}] < \varepsilon$ and it is a* minimal n-step ε-core *if it is additionally inclusion minimal.*

As before, whenever n or ε are clear from the context, we may drop the corresponding part of the name. Again, similar properties hold and we omit the completely analogous proof.

▶ **Theorem 11.** *Let $\mathcal{M}$ be an MDP, $\varepsilon > 0$, and $n \in \mathbb{N}$. Then $S_{\varepsilon,n} \subseteq S$ is an n-step ε-core if and only if for all $R \subseteq S$ we have $0 \leq \mathbb{P}^{\max}[\Diamond^{\leq n} R] - \mathbb{P}^{\max}[\Diamond^{\leq n}(R \cap S_{\varepsilon,n}) \cap \Box^{\leq n} S_{\varepsilon,n}] < \varepsilon$.*

These finite-horizon cores are much smaller than their "infinite" counterparts on some models, even for large n. For instance, in our modified running example of Figure 2, omitting the "complex" states gives an n-step core even for very large n (depending on τ). On the other hand, finding such finite cores seems to be harder in practice. Naively, one could apply the core learning approach of Algorithm 1 to a modified model where the number of steps is encoded into the state space, i.e. $S' = S \times \{0, \dots, n\}$. Unfortunately, this yields abysmal performance, since we store and back-propagate $|S| \cdot n$ values instead of only $|S|$. Nevertheless, we can efficiently approximate them by enhancing our previous approach with further observations.

4.2 Learning a Finite Core

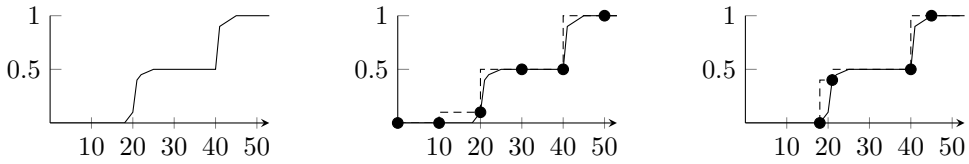
In Algorithm 2, we present our learning variant for the finite-horizon case. This algorithm is structurally very similar to the previous Algorithm 1. The fundamental difference is in Line 6, where the bounds are updated. One key observation is that the probability of

Algorithm 2 LearnFiniteCore.

Input: MDP $\mathcal{M}$, precision $\varepsilon > 0$, step bound n, upper bounds GetBound / UpdateBound, state set $S_{\varepsilon,n}$ with $s_0 \in S_{\varepsilon,n}$

Output: $S_{\varepsilon,n}$ s.t. $S_{\varepsilon,n}$ is an n-step ε-core

1: **while** GetBound$(s_0, n) \geq \varepsilon$ **do**
2: $\quad \varrho \leftarrow$ SamplePath$(s_0, n,$ GetBound$)$ ▷ Generate path
3: $\quad S_{\varepsilon,n} \leftarrow S_{\varepsilon,n} \cup \varrho$ ▷ Update Core
4: $\quad$ **for** $i \in [n-1, n-2, \ldots, 0]$ **do** ▷ Back-propagate values
5: $\quad\quad s \leftarrow \varrho_i$, $r \leftarrow n - i$
6: $\quad\quad$ UpdateBound $\left(s, r, \max_{a \in A(s)} \sum_{s' \in S} \Delta(s,a,s') \cdot \text{GetBound}(s', r-1)\right)$
7: **return** $S_{\varepsilon,n}$



(a) True bounds. **(b)** Simple approx. **(c)** Adaptive approx.

Figure 3 An example for the different approximation approaches. The graphs depict the probability of exiting the core on the y axis within a given amount of steps on the x axis by a solid line and the corresponding approximation returned by GetBounds by a dashed line. From left to right, we have example bounds, which agree with the dense representation, followed by our sparse approach, which over-approximates the bounds, but requires less memory, and finally an adaptive approach, which closely resembles the precise bounds while consuming less memory.

reaching some set R within k steps is at least as high as reaching it within $k-1$ steps, i.e. $\mathbb{P}_s^{\max}[\Diamond^{\leq k} R] < \varepsilon$ is non-decreasing in k for any s and $R \subseteq S$. Therefore, we can use function over-approximations to store upper bounds sparsely and avoid storing n values for each state. To allow for multiple implementations, we thus delegate the storage of upper bounds to an abstract function approximation, namely GetBound and UpdateBound. This approximation scheme is supposed to store and retrieve the upper bound of reaching unexplored states for each state and number of *remaining* steps. We only require it to give a *consistent* upper bound, i.e. whenever we call UpdateBound(s, r, p), GetBound(s, r') will return at least p for all $r' \geq r$. Moreover, we require the trivial result GetBound$(s, 0) = 0$ for all states s. In the following Section 4.3, we list several possible instantiations.

▶ **Theorem 12.** *Algorithm 2 is correct if* UpdateBound–GetBound *are consistent and correct w.r.t. the given state set $S_{\varepsilon,n}$. Further, if* UpdateBound *stores all values precisely and* SamplePath *samples any state reachable within n steps infinitely often (a.s.), the algorithm terminates (a.s.).*

Sketch. *Correctness*: As before, the upper bound function is only updated through Bellman backups, which preserve correctness.

Termination: Given that the upper bound function stores all values precisely, the algorithm is an instance of asynchronous value iteration, which is guaranteed to converge [18]. ◀

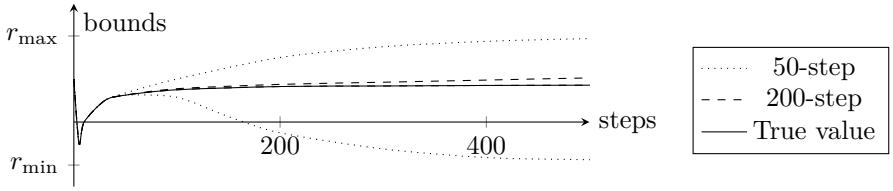


Figure 4 A schematic plot for an average reward extrapolation analysis on step bounded cores. The solid line represents the true value, while the dotted and dashed lines are the respective upper and lower bounds computed for a 50 and 200-step core. Note that the second dashed line (lower bound on the 200 core) coincides with the solid line (true value).

4.3 Implementing the function approximation

Several instances of the UpdateBound–GetBound approach are outlined in Figure 3. The first, trivial implementation is dense storage, i.e. explicitly storing a table mapping $S \times \{0, \dots, n-1\} \to [0,1]$. This table representation consumes an unnecessary amount of memory, since we do not need exact values in order to just guide the exploration. Hence, in our implementation, we use a simple sparse approach where we only store the value every K steps, where K manually chosen. This is depicted in Figure 3b for $K = 10$ – every black dot represents a stored value, the dashed lines represent the value returned by GetBounds. The adaptive approach of Figure 3c adaptively chooses which values to store and is left for future work.

4.4 Stability and its applications

In this section, we explain the idea of a core's *stability*. Given an n-step core $S_{\varepsilon,n}$, we can easily compute the probability $\mathbb{P}^{\max}[\Diamond^{\leq N}\overline{S_{\varepsilon,n}}]$ of exiting the core within $N > n$ steps using, e.g., value iteration. The rate of increase of this exit probability intuitively gives us a measure of quality for a particular core. Should it rapidly approach 1 for increasing N, we know that the system's behaviour may change drastically within a few more steps. If instead this probability remains small even for large N, we can compute properties with a large step bound on this core with tight guarantees. We define stability as the whole function mapping the step bound N to the exit probability, since this gives a more holistic view on the system's behaviour than a singular value. In the following, we give an overview of how finite cores and the idea of stability can be used for analysis and interpretation, helping to design and understand complex systems.

As we have argued above, infinite-horizon properties may be deceiving, since (unrecoverable) errors often are bound to happen eventually. Consequently, one might be interested in a "very large"-horizon analysis instead. Unfortunately, such an analysis scales linearly both with the number of transitions and the horizon. Considering that many systems have millions of states, an analysis with a horizon of only 10,000 steps is already out of reach for existing tools. We first show how stable cores can be used for efficient extrapolation to such large horizons.

For simplicity, we consider reachability and argue how to transfer this idea to other objectives. We apply the ideas of interval iteration as used in, e.g., [10, 6], as follows. Intuitively, since we have no knowledge of the partially explored states, we simply assume the worst / best case for them, i.e. assign a lower bound of 0 and upper bound of 1. Furthermore, any explored target state is assigned a lower and upper bound of 1. By applying interval iteration, we can obtain bounds on the N-step and even unbounded reachability. Through

the core property, the bounds for $N \leq n$ necessarily are smaller than ε. But, for larger N, there are no formal guarantees given by the core property. It might be the case that the core is left with probability 1 in $n+1$ steps. Nevertheless, in practice this allows us to get good approximations even for much larger bounds. Often the computation of an n-step core and subsequent approximation of the desired property is even faster than directly computing the N-step property, as shown in the evaluation.

For LTL and parity objectives, we can simply preprocess the obtained n-core by identifying the winning MECs and then applying the reachability idea, to obtain bounds on the satisfaction probability on the core. In the case of mean-payoff, we again require lower and upper bounds on the rewards $r_{\min}$ and $r_{\max}$ of the system in order to properly initialize the unknown values. Then, with the same approach, we can compute bounds on the n-step average reward by simply assign the lower and upper bounds $r_{\min}$ and $r_{\max}$ to all unexplored states instead of 0 and 1. See Figure 4 for a schematic plot of this analysis. Here, the 50-step core is too coarse for any reasonable analysis, it is unstable and can be exited with high probability. On the other hand, the 200-step core is very stable and accurately describes the system's behaviour for a longer period of time. Noticeably, it also contains a MEC guaranteeing a lower bound on the average reward, hence the lower bound actually agrees with the true value. Since the system may be significantly larger than the bounded cores or even infinitely large, this analysis potentially is much more efficient than analysis of the whole system, as shown in the experimental evaluation.

Note that we cannot use this method to obtain arbitrarily precise results. Given some n-step core and some (step bounded) property, there is a maximal precision we can achieve, depending on the property and the structure of the model. Hence, this method primarily is useful to quickly obtain an overview of a system's behaviour instead of verifying a particular property. As we have argued, one cannot avoid constructing a particular part of the state space in order to obtain an ε-precise result. Nevertheless, this may provide valuable insights in a system, quickly giving a good overview of its behaviour or potential design flaws.

We highlight that the presented algorithm can incrementally refine cores. For example, if a 100-step core does not yield a sufficiently precise extrapolation, the algorithm can reuse the computed core in order to construct a 200-step core. By applying this idea in an interactive loop, one can extract a condensed representation of the systems behaviour automatically, with the possibility for further refinements until the desired level of detail has been obtained.

5 Experimental Evaluation

In this section we give practical results for our algorithms on some examples, both the hand-crafted plane model and hand-picked models from case studies.

5.1 Implementation Details

We implemented our approach in Java, using PRISM [13] as a library for parsing its modelling language and basic computations. The implementation supports Markov chains, continuous-time Markov chains (CTMC, via embedding or uniformization [18, Ch. 11.5]) and Markov decision processes. Further, we implemented our own version of some utility classes, e.g., explicit MDP representation and MEC decomposition.

Inspired by the results of [6], we considered the following sampling heuristics. Given a state s, each heuristic first selects an action a which maximizes the expected upper bound. If there are multiple such actions, one of them is randomly selected. Then, a successor is chosen as follows: The RN (Random) heuristics samples a successors according

Table 1 Summary of our experimental results on several models and configurations for the infinite horizon core learning. The "PRISM" column shows the total number of states and construction time when explored with the explicit engine. The following columns show the size and total construction time of a 10^{-6}-core for each of the sampling heuristics.

Model	Param.	PRISM		RN		GD		MX	
`zeroconf`	100; 5; 0.1	496,291	8.4s	17,805	2.3s	1,072	0.4s	1,450	0.5s
(`N`; `K`; `loss`)	100; 10; 0.1	$3.0 \cdot 10^6$	55s	11,900	1.7s	1,006	0.3s	1,730	0.5s
	100; 15; 0.1	$4.7 \cdot 10^6$	159s	16,997	2.4s	1,067	0.4s	2,002	0.7s
`airplane`	100; `ff`	10,208	0.4s	6	0.1s	6	0.1s	6	0.1s
(`size`; `return`)	10000; `ff`	MEMOUT		6	0.1s	6	0.1s	6	0.1s
`brp`	20; 10	2,933	0.2s	2,352	0.3s	2,568	0.4s	2,675	0.5s
(`N`; `MAX`)	20; 100	26,423	0.6s	7,060	0.6s	3,421	0.4s	3,679	0.5s
	20; 1000	261,323	0.6s	7,624	0.6s	5,118	0.4s	3,912	0.5s
`wlan`	—	345,000	4.5s	344,835	52s	344,996	50s	344,997	52s

to $\Delta(s,a)$. The GD (Guided) approach samples a successor weighted by the respective upper bound, i.e. randomly select a state with probability proportional to $U(s') \cdot \Delta(s,a,s')$ or $\textsc{GetBound}(s',r) \cdot \Delta(s,a,s')$, respectively. Finally, MX (Max) samples a successor s' with probability proportional only to its upper bound $U(s')$ or $\textsc{GetBound}(s',r)$, respectively.

Recall that our algorithms can be restarted with faithful upper bounds and thus we can interleave it with other computations. In our implementation we alternate between the guided exploration of Algorithm 2 and precise computation on the currently explored set of states, guaranteeing convergence in the finite setting.

5.2 Models

In our evaluation, we considered the following models. All except the `airplane` model are taken from the PRISM case studies [16]. `airplane` is our running example from Figure 1 and Figure 2, respectively. The parameter `return` controls whether a return trip is possible, `size` quadratically influences the size of the "recovery" region. `zeroconf` [14] describes the IPv4 Zeroconf Protocol with `N` hosts, the number `K` of probes to send, and a probability of a message `loss`. `wlan` [15] is a model of two WLAN stations in a fixed network topology sending messages on the shared medium. `brp` is a DTMC modelling a file transfer of `N` chunks with bounded number `MAX` of retries per chunk. Finally, `cyclin` is a CTMC modelling the cell cycle control in eukaryotes with `N` molecules. We analyse this model using uniformization.

5.3 Results

We evaluated our implementation on an i7-4700MQ 4x2.40 GHz CPU with 16 GB RAM. We used a default precision of 10^{-6} for all experiments. The results for the infinite and finite construction are summarized in Table 1 and Table 2, respectively. We briefly discuss them in the following sections. Note that the results may vary due to the involved randomization.

5.3.1 Infinite Cores

As already explained in [6], the `zeroconf` model is very well suited for this type of analysis, since a lot of the state space is hardly reachable. In particular, most states are a result of collisions and several message losses, which is very unlikely. Consequently, a very small part

Table 2 Summary of our experimental results on several models and configurations for the finite-horizon 10^{-6}-core learning with 100 step bound; using the notation from Table 1.

Model	Param.	PRISM		RN		GD		MX	
`airplane`	100; `tt`	20,413	0.6s	11	0.3s	11	0.3s	554	0.6s
(`size`; `return`)	10000; `tt`	MEMOUT		11	0.3s	11	0.3s	603	0.6s
`wlan`	—	345,000	4.4s	36,718	15s	37,284	11s	36,825	12s
`cyclin`	4	431,101	12s	9,122	1,649s	4,380	3.9s	99,325	93s
(`N`)	5	$2.3 \cdot 10^6$	78s	26,925	8,840s	11,419	13s	817,058	1,462s

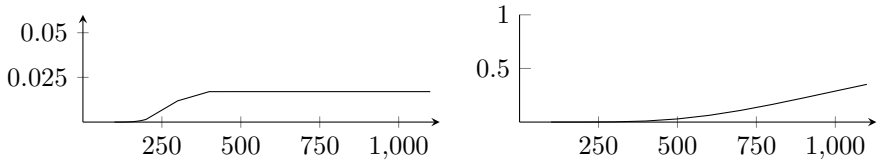


Figure 5 Stability analysis of the `wlan` (left) and `cyclin`($\mathtt{N} = 4$) (right) 100-step core built with the GD heuristic. The graphs show the probability of exiting the respective core within the given amount of steps. Note that the y axis of the `wlan` graph is scaled for readability.

of the model already satisfies the core property. In particular, the size of the core remains practically constant when increasing the parameter K, as only unimportant states are added to the system. Observe that the order of magnitude of explored states is very similar to the experiments from [6]. The same holds true for the `airplane` model, where a significant number of states is dedicated to recovering from an unlikely error. Hence, a small core exists independently of the total size of the model. The `brp` model shows applicability of the approach to Markov chains. In line with the other results, when scaling up the maximal number of allowed errors, the size of the core changes sub-linearly, since repeated errors are increasingly unlikely. In case of the `wlan` model, we observe that all our methods essentially construct the full model.

Comparison to [6]. We also executed the tool presented in [6] where applicable (only MDP are supported). We tested the tool both with a bogus `false` property, i.e. approximating the probability of reaching the empty set, which corresponds to constructing a core, and an actual property. We used the `MAX_DIFF` heuristic of [6], which is similar to GD. Especially on the `false` property, our tool consistently outperformed the previous one in terms of time and memory by up to several orders of magnitude. We suspect that this is mostly due to a more efficient implementation. The number of explored states was similar, as to be expected in light of Theorem 7 and its discussed consequences.

5.3.2 Finite Cores

As expected, the finite core construction yields good results on the `airplane` model, constructing only a small fraction of the state space. Interestingly, the MX heuristic explores significantly more states, which is due to this heuristic ignoring probabilities when selecting a successor and thus sampling a few paths in the recovery region. Also on the real-world models `wlan` and `cyclin`, the constructed 100-step core is significantly smaller than the whole model. For `wlan`, the construction of the respective cores unfortunately takes longer than building the whole model. We conjecture that a more fine-tuned implementation can

Method	Time		States
	model	bounds	
50 steps	0.4s	1.0s	2,137
100 steps	0.9s	3.5s	6,151
200 steps	4.0s	15.3s	22,989
Complete	8.7s	242.4s	431,101

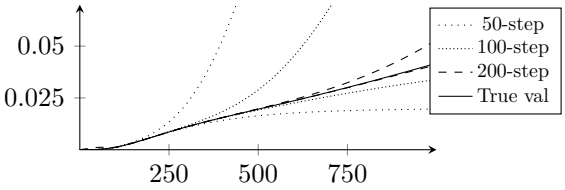


Figure 6 Overview of an extrapolation analysis for `cyclin`($N = 4$). We computed several step-bounded cores with precision 10^{-3}. On these, we computed bounds of a reachability query with increasing step bound. The table on the left lists the time for model construction + computation of the bounds for 1000 steps and the size of the constructed model. The plot on the right shows the upper and lower bounds computed for each core together with the true value. Observe that for growing step-size of the core, the approximation naturally gets more precise.

overcome this issue. In any case, model checking on the explored sub-system supposedly terminates significantly faster since only a much smaller state space is investigated, and the core can be re-used for more queries.

Finally, we applied the idea of stability from Section 4.2 on the `wlan` and `cyclin` models, with results outlined in Figure 5. Interestingly, for the `wlan` model, the escape probability stabilizes at roughly 0.017 and we obtain the exact same probability for *all* heuristics, even for $N = 10{,}000$. This suggests that by building the 100-step core we identified a very stable sub-system of the whole model. Additionally, we observe that at 200-400 steps, the behaviour of the system significantly changes. For the `cyclin` model, we instead observe a continuous rise of the exit probability. Nevertheless, even with 500 additional steps, the core still is only exited with a probability of roughly 6% and thus closely describes the system's behaviour.

On the `cyclin` model, we also applied our idea of extrapolation. The results are summarized in Figure 6. To show how performant this approach is, we reduced the precision of the core computation to 10^{-3}. Despite this coarse accuracy, we are able to compute accurate bounds on a 1000-step reachability query over 10 times faster by only building the 200-step core instead of constructing the full model. These results suggest that our idea of using the cores for extrapolation in order to quickly gain understanding of a model has a vast potential.

5.3.3 Heuristics

Overall, we see that the unguided, random sampling heuristic RN often is severely outperformed by the guided approaches GD and MX, both in terms of runtime and constructed states. Surprisingly, the differences between GD and MX often are small, considering that MX is significantly more "greedy" by completely ignoring the actual transition probabilities. We conjecture that this greediness is the reason for the abysmal performance of MX on the `cyclin` model, where GD seems to strike the right balance between exploration and exploitation. Altogether, the results show that a sophisticated heuristic increases performance by orders of magnitude and further research towards optimizing these heuristic may prove beneficial.

6 Conclusion

We have presented a new framework for approximate verification of probabilistic systems via partial exploration and applied it to both Markov chains and Markov decision processes. Our evaluation shows that, depending on the structure of the model, this approach can yield

significant state space savings and thus reduction in model checking times. Our central idea – finding relevant sub-parts of the state space – can easily be extended to further models, e.g., stochastic games, and objectives, e.g., mean payoff. We have also shown how this idea can be transferred to the step-bounded setting and derived the notion of stability. This in turn allows for an efficient analysis of long-run properties and strongly connected systems.

Future work includes implementing a more sophisticated function approximation for the step-bounded case, e.g., as depicted in Figure 3c. Here, an adaptive method could yield further insight in the model by deriving points of interest, i.e. an interval of remaining steps where the exit probability significantly changes. These breakpoints might indicate a significant change in the systems behaviour, e.g., the probability of some error occurring not being negligible any more, yielding interesting insights into the structure of a particular model. For example, in the bounds of Figure 3, the regions around 20 and 40 steps, respectively, seems to be of significance.

Moreover, a more sophisticated sampling heuristic to be used in SamplePath could be of interest. For example, one could apply an advanced machine learning technique here, which also considers state labels or previous decisions and their outcomes.

In the spirit of [6], our approach also could be extended to a PAC algorithm for black-box systems. Extensions to stochastic games and continuous time systems are also possible.

Further interesting variations are cores for discounted objectives [20] or cost-bounded cores, a set of states which is left with probability smaller than ε given that at most k cost is incurred. This generalizes both the infinite (all edges have cost 0) and the step bounded cores (all edges have cost 1) and allows for a wider range of analysis.

References

1 Pranav Ashok, Yuliya Butkova, Holger Hermanns, and Jan Křetínskỳ. Continuous-Time Markov Decisions Based on Partial Exploration. In *ATVA*, pages 317–334. Springer, 2018.

2 Pranav Ashok, Krishnendu Chatterjee, Przemyslaw Daca, Jan Kretínský, and Tobias Meggendorfer. Value Iteration for Long-Run Average Reward in Markov Decision Processes. In *CAV*, pages 201–221, 2017. doi:10.1007/978-3-319-63387-9_10.

3 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

4 Richard Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

5 Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II: Approximate Dynamic Programming*. Athena Scientific, 2012.

6 Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Křetínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of Markov Decision Processes Using Learning Algorithms. In *ATVA*, pages 98–114. Springer, 2014. doi:10.1007/978-3-319-11936-6_8.

7 Costas Courcoubetis and Mihalis Yannakakis. The Complexity of Probabilistic Verification. *J. ACM*, 42(4):857–907, 1995. doi:10.1145/210332.210339.

8 Pedro R. D'Argenio, Bertrand Jeannet, Henrik Ejersbo Jensen, and Kim Guldstrand Larsen. Reduction and Refinement Strategies for Probabilistic Analysis. In *PAPM-PROBMIV*, pages 57–76. Springer, 2002.

9 Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *CAV*, pages 592–600. Springer, 2017.

10 Serge Haddad and Benjamin Monmege. Reachability in MDPs: Refining convergence of value iteration. In *International Workshop on Reachability Problems*, pages 125–137. Springer, 2014.

11 Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. PASS: abstraction refinement for infinite probabilistic models. In *TACAS*, pages 353–357. Springer, 2010.

12 Jan Křetínský and Tobias Meggendorfer. Of Cores: A Partial-Exploration Framework for Markov Decision Processes. *arXiv e-prints*, June 2019. `arXiv:1906.06931`.

13 M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *TOOLS*, pages 200–204, 2002.

14 Marta Kwiatkowska, Gethin Norman, David Parker, and Jeremy Sproston. Performance analysis of probabilistic timed automata using digital clocks. *FMSD*, 29(1):33–78, 2006.

15 Marta Kwiatkowska, Gethin Norman, and Jeremy Sproston. Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In *Process Algebra and Probabilistic Methods: Performance Modeling and Verification*, pages 169–187. Springer, 2002.

16 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. The PRISM Benchmark Suite. In *QEST*, pages 203–204. IEEE Computer Society, 2012. The models are accessible at `http://www.prismmodelchecker.org/casestudies/`.

17 H Brendan McMahan, Maxim Likhachev, and Geoffrey J Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML*, pages 569–576. ACM, 2005.

18 M.L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley and Sons, 1994.

19 Tim Quatmann and Joost-Pieter Katoen. Sound Value Iteration. In *CAV (1)*, volume 10981 of *LNCS*, pages 643–661. Springer, 2018.

20 Aaron Sidford, Mengdi Wang, Xian Wu, and Yinyu Ye. Variance Reduced Value Iteration and Faster Algorithms for Solving Markov Decision Processes. In *SODA*, pages 770–787. SIAM, 2018.