OF CORES: A PARTIAL-EXPLORATION FRAMEWORK FOR MARKOV DECISION PROCESSES

JAN KŘETÍNSKÝ AND TOBIAS MEGGENDORFER

Technical University of Munich, Germany
e-mail address: jan.kretinsky@in.tum.de

Technical University of Munich, Germany
e-mail address: 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. We obtain efficient analysis algorithms based on partial exploration for various settings, including the challenging case of strongly connected systems.

1. Introduction

Markov decision processes (MDP) are an established formalism for modelling, analysis, and optimization of probabilistic systems with non-determinism, with a large range of application domains [Put94, Whi93, Whi88]. 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., [ACD+17]. As an alternative, one can apply dynamic programming, typically value iteration (VI) [Bel57], the default method in the probabilistic model checkers PRISM [KNP02] and Storm [DJKV17].

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., [BCC+14]. Surprisingly, the standard VI also suffered from a fundamental

correctness issue, where convergence was only guaranteed in the limit, without a proper *stopping criterion*. Only recently, an error bound (and thus a stopping criterion) was given independently in [HM14, BCC$^+$14]. The error bound was derived from the under- and (newly obtained) over-approximations converging to the true value. This resulted not only in error bounds on VI, but opened the door to error bounds for other techniques, including those where even convergence is not guaranteed. For instance, while VI iteratively approximates the value of all states, the above-mentioned *asynchronous VI* evaluates states at different paces. Thus convergence is often unclear and even the rate of convergence is unknown and very hard to analyze. However, augmenting asynchronous VI with this error bound immediately provides a correct algorithm. A prime example is the modification of BRTDP [MLG05] to reachability [BCC$^+$14] with error bounds. These ideas are further developed for, e.g., settings with long-run average reward [ACD$^+$17], continuous time [ABHK18], or stochastic games [KKKW18].

While these solutions are efficient, they are ad-hoc, implicitly sharing the idea of *simulation / learning-based partial exploration* of the system. 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.

In essence, the previous algorithms use (i) simulations to explore the state space and (ii) heuristics to analyze their 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, which is independent of the particular objective (which state 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 can 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. Moreover, the general concept of cores provides a *unified* understanding of the previous algorithms and allows for easier development of further partial-exploration techniques within this framework. Additionally, identifying the core can serve to better *understand* the typical behaviour of the system. The core potentially is a lot smaller than the whole 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. In other words, the core comprises only *important* states of the system. This underlying idea of cores is not bound to MDP in any way and can be extended naturally to a broad variety of probabilistic formalisms, such as probabilistic programs, evolutionary games, and many more, providing a unified notion of importance across all of these areas. Altogether, this motivates us to investigate this notion *eo ipso*.

Moreover, in the case of MDP, making the notion of core explicit 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 so called *end components* (parts of the state space in which the system may remain forever). Consequently, the minimal core for a system which consists of a single end component 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 any kind of feasible analysis for this case, we introduce the n-step core. It is defined by the system staying there with high probability for n steps. This n-step core can naturally be used for analysis of bounded, n-step horizon properties. However, by explicitly viewing the core as a set of states we are able to derive the notion of "stability" of a core. This stability essentially describes 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 classical, direct N-step analysis on appropriately shaped models, 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 explain how this notion can be transferred to other properties and models.
- We provide efficient, learning-based algorithms for computing both types of cores and evaluate them on several examples.

1.1. **Related Work.** Since the notion of core is fundamentally novel, we list works related to two areas of our contributions, namely (i) to speed up (reachability) analysis of MDP and (ii) algorithms to efficiently find small cores in practice. Note that the former point is not a primary goal of our work, only an immediate and useful consequence.

In order to improve value iteration, several approaches are considered. [HK19] employs the idea of *optimistic* value iteration, essentially guessing and verifying upper bounds, saving computational effort. In [QK18], the authors approximate the exit probability of a particular set to bound the error on the computed reachability, potentially allowing for early termination. Note that this may seem similar to our idea, however the authors consider a fixed, pre-computed set, relative to a particular reachability query, while our approach seeks to find a set of states only dependent on the model itself.

Another natural idea is to apply state space reduction heuristics. This includes abstraction approaches, e.g., [DJJL02, HHWZ10], or a dual approach based on restricting the analysis to a part of the state space. Examples of the latter are asynchronous VI in probabilistic planning, e.g., [MLG05], or projections in approximate dynamic programming, e.g., [Ber12]. In both, only a certain subset of states is considered for analysis, leading to speed ups in orders of magnitude. However, these are best-effort solutions, which can only guarantee convergence to the true result in the limit, with no error bounds at any time.

Based on [MLG05], [BCC$^+$14] additionally provides an error bounds while only exploring a subset of states. The approach of [BCC$^+$14] inspired our work and thus naturally is closely related. In particular, our experimental evaluation shows that the approach of [BCC$^+$14] explores a core for almost all practical examples. However, our fundamental goal is different: While [BCC$^+$14] aims to answer a given query, we instead provide an analysis of a *system*.

Algorithmically, our approach to find small cores is related to the ideas of [HM14, BCC$^+$14]. Both works maintain bounds for each state, iterating an operator similar to classical value iteration, and collapse end components to ensure convergence. However, [HM14] constructs the whole system, in contrast to our sampling-based approach. Our algorithms are structurally close to the BRTDP algorithm of [BCC$^+$14]. We also use similar ideas to focus computation on promising areas by the means of a guided sampling approach. However, we again emphasize that the overall goal is fundamentally different.

2. Preliminaries

In this section, we recall basics of probabilistic systems and set up the notation. We assume familiarity with 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 2.1. 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 2.2. A *Markov decision process (MDP)* is a tuple $\mathcal{M} = (S, s_0, A, \mathsf{Av}, \Delta)$, 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.

We assume w.l.o.g. that actions are unique for each state, i.e. $\mathsf{Av}(s) \cap \mathsf{Av}(s') = \emptyset$ for $s \neq s'$. This 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.

Remark 2.3. In some works, Markov chains and decision processes are defined without an initial state, which instead is given as part of the query. While natural for some problems, our notion of cores is fundamentally dependent on the initial state.

2.1. **Paths.** An *infinite path* ρ in a Markov chain is an infinite sequence $\rho = s_0 s_1 \cdots \in S^\omega$, 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 $\rho = s_0 a_0 s_1 a_1 \cdots \in (S \times A)^\omega$, 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. In the following, we

slightly abuse notation by identifying $(S \times A)^\omega$ and $(S \times A)^* \times S$ with the set of infinite and finite paths, respectively.

2.2. **Strategies.** 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 call a strategy *memoryless randomized* (or *stationary*) if it is of the form $\pi : S \to \mathcal{D}(A)$, and *memoryless deterministic* (or *positional*) if it is of the form $\pi : S \to A$. We denote the set of all strategies of an MDP by Π, and the set of all memoryless deterministic strategies as Π^{MD}. Note that Π^{MD} is finite, since at each of the finitely many states there exist only finitely many actions to choose from.

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 $\delta^\pi(\varrho, \varrho a_{n+1} s_{n+1}) = \pi(\varrho, a_{n+1}) \cdot \Delta(s_n, a_{n+1}, s_{n+1})$.

2.3. **Measures.** Any Markov chain M induces a unique measure $\mathbb{P}^{\mathsf{M}}$ over infinite paths [BK08, p. 758]. Assuming we fixed some MDP $\mathcal{M}$, we use $\mathbb{P}_s^\pi$ to refer to the probability measure induced by the Markov chain $\mathcal{M}^\pi$ with initial state s. See [Put94, Sec. 2.1.6] for further details. Whenever π or s are clear from the context, we may omit them, in particular, $\mathbb{P}^\pi$ refers to $\mathbb{P}_{s_0}^\pi$. For a given MDP $\mathcal{M}$ and measurable event A, we use the shorthand $\mathbb{P}^{\max}[A] := \sup_{\pi \in \Pi} \mathbb{P}^\pi[A]$ and $\mathbb{P}_s^{\max}[A] := \sup_{\pi \in \Pi} \mathbb{P}_s^\pi[A]$ to refer to the maximal probability of A over all strategies (starting in s). Analogously, $\mathbb{P}^{\min}[A]$ and $\mathbb{P}_s^{\min}[A]$ refer to the respective minimal probabilities.

Note that in general the supremum or infimum may not be obtained, depending on the structure of A. However, we only consider events where an optimal strategy always exists, hence we use the superscripts max and min for emphasis.

2.4. **End components.** A non-empty set of states $C \subseteq S$ in a Markov chain is *strongly connected* if for every pair $s, s' \in C$ there is a non-trivial path from s to s'. Such a set C is a *strongly connected component* (SCC) if it is maximal w.r.t. set inclusion, i.e. there exists no strongly connected C' with $C \subsetneq C'$. The set of SCCs in an MC M is denoted by $\mathsf{SCC}(\mathsf{M})$.

The concept of SCCs is generalized to MDPs by so called *(maximal) end components*. 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 $\varrho = s a_0 \ldots a_n s' \in (T \times B)^* \times T$, 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 $\mathsf{MEC}(\mathcal{M})$.

Remark 2.4. For a Markov chain M, the computation of $\mathsf{SCC}(\mathsf{M})$ and a topological ordering of the SCCs can be achieved in linear time w.r.t. the number of states and transitions by, e.g., Tarjan's algorithm [Tar72]. Similarly, the MEC decomposition $\mathsf{MEC}(\mathcal{M})$ of an MDP can be computed in polynomial time [CY95]. See [CH11, CH12, CH14] for improved algorithms on general MDP and various special cases.
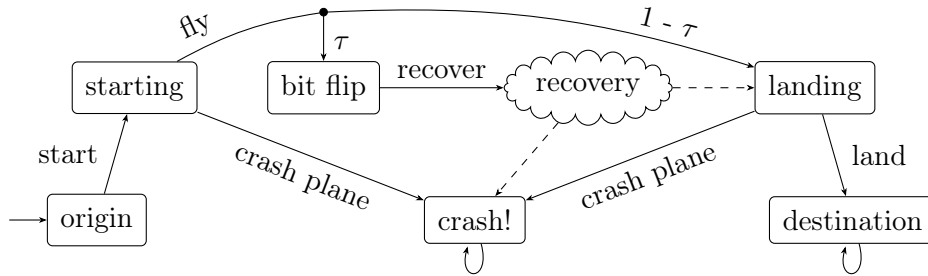
FIGURE 1. A simplified model of a flight, where $\tau = 10^{-10}$ is the probability of potentially hazardous bit flips occurring during the flight. The "recovery" node represents a complex recovery procedure, comprising many states.

2.5. **Objectives.** In the following, we 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 $\mathbb{P}^{\max}[\lozenge T]$ and $\mathbb{P}^{\max}[\lozenge^{\leq n} T]$, where T is the set of target states and $\lozenge T$ ($\lozenge^{\leq n} T$) refers to the measurable set of runs that visit T at least once (in the first n steps). The dual operators $\square T$ and $\square^{\leq n} T$ refer to the set of runs which remain inside T forever or for the first n steps, respectively. See [BK08, Sec. 10.1.1] for further details, e.g., proofs of measurability.

Our techniques are easily extendable to other related objectives like *long run average reward* (*mean payoff*) [Put94], *LTL formulae*, and ω-regular objectives [BK08], or more general systems like *stochastic games*. We comment on these extensions in Section 3.3. Some of these require further knowledge about the model, which we also explain there.

2.6. **Approximate Solutions.** We are interested in finding approximate solutions efficiently, or, in other words, 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 a (guaranteed) absolute error of less than ε. For example, given a reachability query $\mathbb{P}^{\max}[\lozenge T]$ and precision ε, we are interested in finding a value v with $|\mathbb{P}^{\max}[\lozenge T] - v| < \varepsilon$.

3. THE CORE IDEA

In this section, we present the novel concept of *cores*, inspired by the approach of [BCC$^+$14], where a specific reachability query was answered approximately through heuristic based methods. We first establish a running example to motivate our work and explain the difference to previous approaches.

Consider the flight of an airplane. The plane is controlled by the pilot and the flight computer. Together, they can take many decisions to control the plane depending on the current state. Naturally, one may be interested in the maximal probability of arriving at the destination. This intuitively describes how likely it is to arrive safely, assuming that the pilot acts optimally and the computer is bug-free. This probability may be less than 100%. For example, some components may fail even under optimal conditions. See Figure 1 for a simplified MDP modelling this example.

One key observation in [BCC+14] is that some extreme situations may be very unlikely and we can simply assume the worst or best 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 crash or it might have no influence on the evolution of the system at all due to redundancy. 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.

In [BCC+14], the state space is 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 fundamental 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 (in a particular sense) necessary to answer reachability queries 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 3.1 (Core). Let $\mathcal{M}$ be an MDP and $\varepsilon > 0$. A set $S_\varepsilon \subseteq S$ is an ε-*core* if $\mathbb{P}^{\max}[\lozenge \overline{S_\varepsilon}] < \varepsilon$, 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 $\mathbb{P}^{\min}[\Box S_\varepsilon] \geq 1 - \varepsilon$, i.e. the probability to remain inside the core forever is large. We highlight that the set of reachable states is a "0-core". As such, cores intuitively extend the straightforward idea of considering reachable states for analysis to considering only *likely* reachable states.

In the following, we derive basic properties of cores, show how to efficiently construct them, and relate them to the approaches of [ACD+17, BCC+14]. First, we prove the key statement motivating our interest in cores, namely that they are both sufficient and, in a sense, required to compute ε-precise reachability queries.

Theorem 3.2. *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 $0 \leq \mathbb{P}^{\max}[\lozenge R] - \mathbb{P}^{\max}[\lozenge(R \cap S_\varepsilon) \cap \Box S_\varepsilon] < \varepsilon$.*

Proof. We prove both directions of the equivalence separately.

First, let S_ε be a core of $\mathcal{M}$ and $R \subseteq S$ states in $\mathcal{M}$. Clearly,

$$\mathbb{P}^{\max}[\lozenge R] \leq \mathbb{P}^{\max}[\lozenge R \cap \lozenge \overline{S_\varepsilon}] + \mathbb{P}^{\max}[\lozenge R \cap \Box S_\varepsilon]$$

by simple case distinction. Furthermore, we have that

$$\mathbb{P}^{\max}[\lozenge R \cap \Box S_\varepsilon] = \mathbb{P}^{\max}[\lozenge(R \cap S_\varepsilon) \cap \Box S_\varepsilon] \leq \mathbb{P}^{\max}[\lozenge R]$$

and

$$0 \leq \mathbb{P}^{\max}[\lozenge R \cap \lozenge \overline{S_\varepsilon}] \leq \mathbb{P}^{\max}[\lozenge \overline{S_\varepsilon}] < \varepsilon.$$

Together, we obtain

$$0 \leq \mathbb{P}^{\max}[\lozenge R] - \mathbb{P}^{\max}[\lozenge(R \cap S_\varepsilon) \cap \Box S_\varepsilon] \leq$$
$$\mathbb{P}^{\max}[\lozenge R \cap \lozenge \overline{S_\varepsilon}] + \mathbb{P}^{\max}[\lozenge R \cap \Box S_\varepsilon] - \mathbb{P}^{\max}[\lozenge(R \cap S_\varepsilon) \cap \Box S_\varepsilon] < \varepsilon.$$
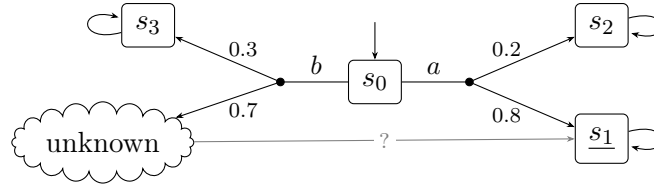
Figure 2. An MDP showing that cores are not always required to answer reachability queries ε-precisely.

For the other direction, assume that $S' \subsetneq S$ is not a core. Now, pick $R = \overline{S'}$, $R \neq \emptyset$ by assumption. Clearly, $R \cap S' = \emptyset$, hence we only need to prove that $\mathbb{P}^{\max}[\Diamond R] > \varepsilon$. By definition, since S' is not a core, we have that $\mathbb{P}^{\max}[\Diamond \overline{S'}] > \varepsilon$. □

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. Conversely, the theorem also shows that if we would consider a set of states not satisfying the core property then there is at least one reachability property that we cannot answer with epsilon-precision guarantees.

Remark 3.3. In the conference paper [KM19] we incorrectly reported a stronger statement, claiming that cores are required to compute *any* property (except some corner cases). This mistake was discovered independently by the authors and reviewers of this work. We show a counterexample to this claim in Figure 2. Here, we can already see that the maximal probability of reaching state s_1 is 0.8 by choosing action a in the initial state s_0, independent of the system's behaviour in the "unknown" area, as we explain below. However, in order to obtain an ε-core for any $\varepsilon < 0.7$, we would need to explore further.

Now, we explain the example of Figure 2 in more detail. Action b immediately leads us to state s_3 with 0.3 probability, a MEC with neither outgoing edges nor a target state. The probability of reaching the target s_1 after choosing action b thus is at most 0.7, independent of the probability of reaching s_1 from the unknown region, indicated by the grey arrow. In other words, one can derive the upper bound 0.7 on the probability of reaching the target after taking action b without investigating the unknown area. Dually, following action a yields a lower bound of 0.8, hence it is clear that the probability of reaching the target is at least 0.8. Moreover, since the remaining probability of 0.2 after taking action a also leads to such an "absorbing" MEC, we can conclude that the maximal probability of reaching s_1 is 0.8. Note that if instead of s_2 there would be another unknown region, we would need to explore it, since it may also eventually lead to the target set.

We emphasize that the counterexample relies on this particular structure of the MDP relative to the reachability objective, i.e. that there is a "shortcut" to the goal as offered by action a together with an immediate dead-end associated with all other actions (see action b in this example). In our experiments, we only rarely observed such a structure.

Theorem 3.2 motivates us to find cores. Of course, one could simply construct the whole state set S, since it is a core for any ε. Note that, in a sense, this is what traditional explicit methods are doing. However, this clearly does not yield any computational advantages. Thus, we naturally are interested in finding a core which is as small as possible, which we call a *minimal core*.
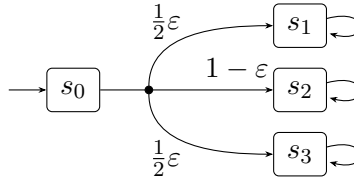
FIGURE 3. A simple MDP showing that minimal cores are not unique.

Definition 3.4 (Minimal Core). Let $\mathcal{M}$ be an MDP and $\varepsilon > 0$. $S_\varepsilon^* \subseteq S$ is a *minimal ε-core* if it is minimal w.r.t. set inclusion, 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, as they are reached with probability $\tau \ll \varepsilon$.

Unfortunately, finding small cores for $\varepsilon > 0$ is computationally quite expensive, as we show in the following. We first prove that there may be several minimal cores for one system. While this statement is rather obvious, we include it due to the instructiveness of its proof, hinting at the underlying combinatorics we use in the following proof.

Proposition 3.5 (Non-uniqueness). *There is an MDP with minimal cores $S_\varepsilon^*, S'^*_\varepsilon$ satisfying $S_\varepsilon^* \neq S'^*_\varepsilon$ for any $0 < \varepsilon < \frac{1}{2}$.*

Proof. Consider the MDP shown in Figure 3. Any ε-core contains the states s_0 and s_2. But $\{s_0, s_2\}$ is not a valid core, since $\mathbb{P}^{\max}[\lozenge \overline{\{s_0, s_2\}}] = \mathbb{P}^{\max}[\lozenge \{s_1, s_3\}] = \varepsilon$. Hence, at least one of s_1 and s_3 has to be part of a core. It is easy to verify that both $\{s_0, s_1, s_2\}$ and $\{s_0, s_2, s_3\}$ are (minimal) cores. $\square$

By extending the above example we can show that there indeed might be exponentially many minimal cores. More importantly, we observe that finding a core of a given size (for a non-trivial ε) is NP-complete.

Theorem 3.6 (NP-completeness). *The problem $\{(\mathcal{M}, k) \mid \mathcal{M} \text{ has an } \varepsilon\text{-core of size } k\}$ is NP-complete for any $0 < \varepsilon < 1$.*

Proof. Containment: The problem is in NP, since the reachability problem of a given set of states in MDP is in P. Thus, a core serves as its own, linearly sized certificate.

Hardness: For hardness, we show a reduction from the `VERTEX-COVER` problem. We briefly recall this problem: One is given an (undirected) graph (V, E) and a threshold $k \in \mathbb{N}$. The question is whether it is possible to find a subset of vertices $V' \subseteq V$ with size at most k, i.e. $|V'| \leq k$, such that each edge of the graph is incident to at least one vertex in V', i.e. for every $\{v, w\} \in E$ either $v \in V'$ or $w \in V'$ (or both).

Let thus a graph (V, E) and number $k \in \mathbb{N}$ be an instance of `VERTEX-COVER`. We construct the MDP $\mathcal{M}$ as depicted in Figure 4. We define the set of states $S = \{s_0, s_-\} \cup V \cup E$, i.e. there is a state for each edge and each vertex of the graph together with two special states s_0 and s_-. In s_0, there is an action for each edge, leading to the respective edge's state with probability 1. Each edge state has a single available action, leading to the two vertices incident to this edge with probability $\frac{\varepsilon}{2}$ each. The remaining probability of $1 - \varepsilon$ directly leads to the sink state s_-. Each vertex state directly moves to the sink state with probability 1. Essentially, in the initial state, we can pick any edge of the graph and probabilistically
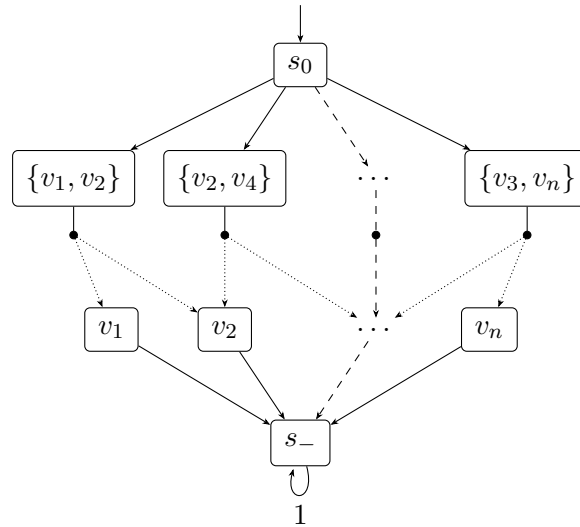
FIGURE 4. The MDP used in the reduction from Vertex cover to cores. All dotted transitions (from "edge" state actions to "vertex" states) have a transition probability of $\frac{\varepsilon}{2}$. For readability, edges from "edge" state actions to the sink state, carrying the remaining $1 - \varepsilon$ probability, are omitted.

move to its two vertices. Clearly, the size of the MDP is linear in the size of the graph (note that ε is fixed and thus not part of the input).

We show that this MDP admits a core of size at most $2 + |E| + k$ iff the input graph has a vertex cover of size at most k. First, assume that a vertex cover V' exists. We show that $S_\varepsilon = \{s_0, s_-\} \cup E \cup V'$ is a core by contradiction. Assume that the probability of reaching $\overline{S_\varepsilon} = V \setminus V'$ is at least ε and let π be a deterministic witness strategy. Let $e = \{v, w\} \in E$ the edge which is selected by π in the initial state, i.e. the unique state with $\Delta(s_0, \pi(s_0), e) = 1$. Clearly, neither v nor w are in S_ε, since otherwise π would not be a witness. This contradicts the assumption that V' is a vertex cover, since e is not covered by it.

Now, assume that a core S_ε of size $2 + |E| + k$ exists. Clearly, s_0, s_- and all edge states are part of any core, since all of them can be reached with probability one (and $\varepsilon < 1$). Hence, let $V' = V \cap S_\varepsilon$ be the vertex states in the core. Clearly, $|V'| \leq k$. We show that V' is a vertex cover. Let $e = \{v, w\} \in E$ be any edge. By analogous argumentation as above we necessarily have either $v \in V'$ or $w \in V'$. $\qquad\square$

Observe that this result only implies that finding the smallest minimal cores is hard. By virtue of Theorem 3.2, we still are interested in finding small, not necessarily minimal cores— any reduction in number of states directly translates to a speed-up in subsequent computations. Thus we introduce a learning-based approach which quickly identifies reasonably sized cores in the following section.

Remark 3.7. We highlight that the above proof shows NP-completeness even for the restricted class of acyclic MDP. However, an NP-completeness proof for Markov chains remains open. Instead of using actions to choose edge states, we could introduce a uniform distribution over them. However, then the transition probabilities from edge states to vertex

---

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
 1: **while** $U(s_0) \geq \varepsilon$ **do**
 2:    $\varrho \leftarrow$ SAMPLEPATH(s_0, U) ▷ Generate path
 3:    $S_\varepsilon \leftarrow S_\varepsilon \cup \varrho$ ▷ Expand core
 4:    UPDATEECS(S_ε, U)
 5:    **for** s in ϱ **do**   ▷ Back-propagate values
 6:        $U(s) \leftarrow \min\{U(s), \max_{a \in A(s)} \sum_{s' \in S} \Delta(s, a, s') \cdot U(s')\}$
 7: **return** S_ε

---

states would need to be re-weighted to $|E|\frac{\varepsilon}{2}$. Consequently, ε has to be smaller than $\frac{1}{|E|}$ and thus this does not prove the NP-hardness for arbitrary ε.

An interesting variant is the computation of a "pre-core": states which necessarily are contained in any core for some given ε, as, for example, s_0, E and s_- in the above hardness proof. We suspect that a greedy algorithm may be able to identify such states in PTIME. Such an analysis may yield an interesting preprocessing step to quickly identify very important states, thus speeding up computation of cores. This particularly may be helpful for systems which are strongly connected, which we explain later.

3.2. **Learning a Core.** As we have shown in the previous section, finding a minimal core is NP-complete, hence we aim for a best-effort, learning-based algorithm, often identifying a small core. To this end, we introduce a guided, sampling-based algorithm in Algorithm 1. Our method is structurally very similar to the algorithm introduced in [BCC+14]. 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, instead of an explicit list of transitions. This allows us to only construct a small fraction of the state space and achieve sub-linear runtime (in the number of states and transitions) for many models. In particular, we observe in Section 5 that for some models we are able to identify a small core orders of magnitude faster than the construction of the state set S, speeding up subsequent computations drastically.

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*: The algorithm is aware of their existence but has no other information about these states. Furthermore, the algorithm stores for each (explored) state s an upper bound $U(s)$ on the probability of reaching unexplored states starting from s. The algorithm gradually grows the set of explored states and simultaneously updates their upper bounds safely until the desired threshold is achieved in the initial state, i.e. $U(s_0) < \varepsilon$. Then, 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 updates (also called Bellman backups).

SAMPLEPATH samples paths following some heuristic. These paths do not have to be rooted in the initial state s_0, follow the transition probabilities given by the successor

function, resolve non-determinism in a particular way, or be of a particular length. For example, a successor might be sampled with probability proportional to its upper bound times the transition probability or in a round-robin scheme[1]. In our implementation, we follow the former idea. The intuition is as follows. We want to explore states which indeed are likely to be reached, since those have to be included in a core anyway. But we do not want to waste computational effort on states which have a small probability of reaching new unexplored states. The product of transition probability and upper bound is only large if that successor is both likely to be reached and has a (presumably) large chance of reaching a new unexplored state. Otherwise, the successor probably is hardly reachable in general or we already have gathered enough information and hence do not need to explore further in that direction. As we show later in the experimental evaluation in Section 5, using the currently stored upper bounds as guidance often yields significant speed-ups in practice. SAMPLEPATH can also incorporate machine learning techniques and domain knowledge about the system, yielding even better suggestions about important states.

UPDATEECS identifies MECs of the currently explored sub-system and "collapses" them into a single representative state. Alternatively, this can be viewed as linking the bounds of all states in each end component together. In particular, each state's bound is set to the maximum bound of all actions leaving the end component, omitting all "internal" actions. This is necessary to ensure convergence of the upper bounds to the correct value. Technically this process removes spurious fixed points of U. We briefly explain this issue in the following, it is more thoroughly explained in, e.g., [BCC+14, ACD+17].

Recall that from each state within an EC we can reach every other state of the EC with probability one. Remaining inside the (explored) EC will not lead to an unexplored state. If, for example, some state s can reach unexplored states with probability 0.5, so can every state s' in the EC by first moving to s and then following the actions necessary to achieve the 0.5 probability. Setting the upper bound of all states in an EC to the maximum upper bound of all "outgoing" actions thus intuitively is correct—but it is also necessary for convergence: Observe that by definition, the upper bound of each state initially is set to 1. Now, consider, for example, a MEC consisting of a single state s with a self loop under action a. Since s can reach a state with upper bound 1 under action a (namely itself), the update in Line 6 of the algorithm will always keep $U(s)$ at 1. By identifying $(\{s\}, \{a\})$ as a MEC and removing the internal action, we ensure convergence. Furthermore, MECs without outgoing edges are the only parts of the system which "create" 0 upper bounds—only there do we know for sure that no unexplored state can be reached. We omit a precise definition of the underlying MEC-quotienting procedure [DA97], since it entails a lot of technical subtleties, distracting from our main result. For the sake of understanding the algorithm, it is safe to assume that $\mathcal{M}$ does not contain any MECs except trivial sinks, which we can easily identify and immediately assign an upper bound of 0.

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. Observe that we do not require that ϱ always starts in s_0, only that this happens again and again. Further, we require that the initial upper bounds are consistent with the given state set, i.e. $U(s) \geq \mathbb{P}_s^{\max}[\lozenge \overline{S_\varepsilon}]$. This is trivially satisfied by $U(\cdot) = 1$. Note that in contrast to [BCC+14],

---

[1]For example, by numbering the successors of some action arbitrarily, we simply select a successor in ascending fashion whenever we choose that particular action.

the set whose reachability we approximate dynamically changes and, further, only upper bounds are computed.

Theorem 3.8. *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. Correctness: By assumption $U(s)$ initially is a correct upper bound for the "escape" probability, i.e. $U(s) \geq \mathbb{P}_s^{\max}[\lozenge \overline{S_\varepsilon}]$. Each update in Line 6 preserves correctness, independent of the sampled path. Moreover, we set $U(s) \leftarrow 0$ if UpdateECs identifies an EC without outgoing actions, which trivially is correct, too. Hence, if $U(s_0) < \varepsilon$, we have $\mathbb{P}_s^{\max}[\lozenge \overline{S_\varepsilon}] < \varepsilon$.

Termination: Recall that we assumed that SamplePath is (a.s.) fair. This implies that eventually the whole model will be explored, i.e. $S_\varepsilon = S$ (otherwise there would exist a partially explored state which is never visited, contradicting the fairness condition). Consequently, all MECs will be collapsed by UpdateECs. In particular, all MECs without outgoing actions have their upper bound U set to 0. Moreover, since U is monotonically decreasing by definition, the upper bounds of any state s converge to a value $U^*(s)$. Now, assume that there exists a state s where $U^*(s) > 0$, i.e. its upper bound does not converge to zero. Let w.l.o.g. s be a state with maximal U^* among all states. Recall that every state is visited infinitely often by our fairness assumption, in particular s. By definition of U, it is easy to see that all successors s' of s under any action necessarily have the same value $U^*(s') = U^*(s)$, since otherwise the value of s would eventually be decreased by an update. Now, this implies that the set of states with maximal values, i.e. $\{s' \mid U^*(s') = U^*(s)\}$ is closed under the transition dynamics of the system and contains at least one end component, contradicting the fact all ECs are collapsed by UpdateECs. Consequently, we have $U^*(s) = 0$ for any state s, in particular we have that eventually $U(s_0) < \varepsilon$, proving termination. $\square$

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, which can speed up convergence drastically for particular models. Further, we can reuse the computed upper bounds and state set to compute a core for a tighter precision.

3.3. **Extending Cores to other Properties and Models.** We explain how a core can be used for verification and how our approach differs from existing ones. Of course, we can compute reachability or safety objectives on a given core ε-precisely. In this case, our approach conceptually is not too different from the one in [BCC⁺14]. 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 [ACD⁺17] 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 [ACD⁺17]. First, rational rewards are assigned to each state, which are obtained upon each visit to that state. 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 [ACD⁺17], a mean payoff objective can be decomposed into a separate analysis of each

(explored) MEC and a (weighted) reachability query

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

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 [BK08]. Again, we can bound the reachability of unexplored MECs and thus the error we incur when only analysing the core. Note that the statement of Theorem 3.2 directly carries over to these settings. Moreover, it also transfers to *minimal reachability / satisfaction* queries. Intuitively, the minimal probability of reaching a given set is obtained by maximizing the probability of reaching states from which the given set can be avoided forever, i.e. ECs not intersecting the given set. More formally, we have $\mathbb{P}^{\min}_{\mathcal{M},s}[\Diamond R] = 1 - \mathbb{P}^{\max}_{\mathcal{M},s}[\Box \overline{R}]$. Observe that in this sense minimal reachability can be phrased as maximizing satisfaction of an LTL query. More directly, we can again decompose minimal reachability into a graph analysis to identify all "safe" ECs and then apply maximal reachability. Hence, given a core, we can approximate the minimal probability up to a precision of ε and a variant of Theorem 3.2 is applicable.

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 at a *controllable* loss of precision.

We highlight that our approach moreover is directly applicable to models with infinite state space, since finite cores still may exist for these models. Moreover, we can also apply our core idea directly to *stochastic games*, i.e. MDP where an additional, antagonistic player is introduced. Here, we can compute a core by interpreting the game as an MDP where both players cooperate. In other words, a core of a stochastic game is a set of states where neither player can ever escape from with significant probability. It is not difficult to see that the essence of Theorem 3.2 also carries over to this setting.

Even more generally, the essential idea of cores, namely to classify states as "important" based on the probability of them occurring along a path can be transferred to many other probabilistic formalisms, immediately providing an intuitive, unified notion of importance to these areas. For example, the concept of *stochastic invariants* [CNZ17] of probabilistic programs is equivalent to a core of the underlying Markov chain.

4. Beyond Infinity

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 4.1. *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 $s' \in S_{\varepsilon}$. $\qquad\square$
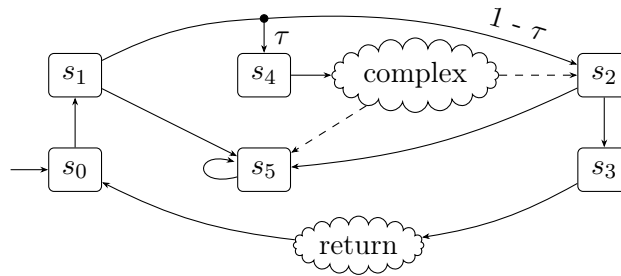
FIGURE 5. An adaptation of the model from Figure 1, with an added return trip, represented by the "return" node. State and action labels have been omitted for readability.

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 3.2, 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. In our experimental evaluation, strongly connected components prove to be a challenge for our approach, since it spends a lot of time computing unnecessary information until eventually the whole model is explored. Here, a "pre-core" approach as mentioned in Remark 3.7 may help to improve performance.

However, real-world models often exhibit a particular structure, with many states only being visited infrequently. For example, a biological process may reach some ratio of specimen very rarely. 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, as shown in Figure 5. Now, the plane eventually will suffer from a bit flip. Additionally, assuming that there is a non-zero probability of not being able to recover from the error, the plane will eventually crash with probability 1, *independent* of the strategy.

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". Note that this is quite different from computing the optimal strategy for a single trip and applying it repeatedly. In practice, we are, in fact, often 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 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.

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:    $\varrho \leftarrow$ SamplePath$(s_0, n,$ GetBound$)$ ▷ Generate path of length n
 3:    $S_{\varepsilon,n} \leftarrow S_{\varepsilon,n} \cup \varrho$ ▷ Update Core
 4:    **for** $i \in [n-1, n-2, \dots, 0]$ **do**   ▷ Back-propagate values
 5:        $s \leftarrow \varrho_i$, $r \leftarrow n - i$
 6:        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** $S_{\varepsilon,n}$

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 4.2 (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 $\mathbb{P}^{\max}[\lozenge^{\leq n} \overline{S_{\varepsilon,n}}] < \varepsilon$ and it is a *minimal n-step ε-core* if it additionally is minimal w.r.t. set inclusion.

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

Theorem 4.3. *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 $0 \leq \mathbb{P}^{\max}[\lozenge^{\leq n} R] - \mathbb{P}^{\max}[\lozenge^{\leq n}(R \cap S_{\varepsilon,n}) \cap \square^{\leq n} S_{\varepsilon,n}] < \varepsilon$.*

Finite-horizon cores are much smaller than their "infinite" counterparts on some models, even for large step bounds n. For instance, in our modified running example of Figure 5, 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. $S' = S \times \{0, \dots, n\}$. However, this comes with a huge increase in space complexity, 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 reaching some set R within k steps is at least as high as reaching it within $k-1$ steps, i.e. $\mathbb{P}^{\max}_s[\lozenge^{\leq k} R] < \varepsilon$ 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 Section 4.3, we list several possible instantiations.

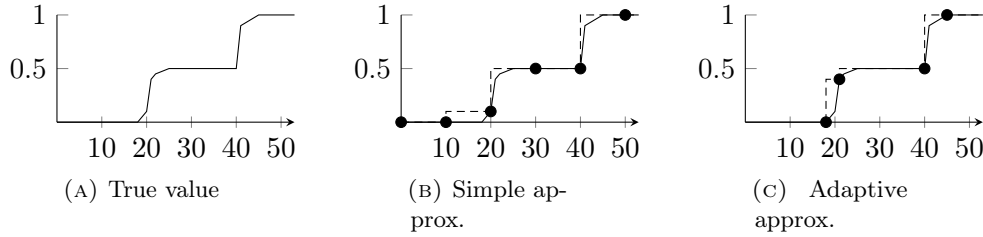(A) True value   (B) Simple approx.   (C) Adaptive approx.

FIGURE 6. An example for different function approximation schemes which could be used to implement UPDATEBOUND and GETBOUND. The graphs depict an arbitrarily chosen, monotonous function by a solid line and the corresponding approximation returned by the approximation scheme 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. Dots represent the values stored by the sparse and adaptive approach. For readability, we do not depict the accumulating errors of the approximative methods in this figure.

Theorem 4.4. *Algorithm 2 is correct if* UPDATEBOUND *and* GETBOUND *are consistent and correct w.r.t. the given state set* $S_{\varepsilon,n}$. *Further, if* UPDATEBOUND *stores all values precisely and* SAMPLEPATH *yields any path of length n infinitely often (a.s.), the algorithm terminates (a.s.).*

Proof. 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 [Put94]. More formally, observe that in this case we can essentially characterize UPDATEBOUND and GETBOUND by a value vector $v : S \times \{0, \ldots, n\} \to [0, 1]$. Then, the update in Line 6 corresponds to setting $v(s, r) = \max_{a \in \mathsf{Av}(s)} \sum_{s' \in S} \Delta(s, a, s') \cdot v(s', r - 1)$ and $v(s, 0) = 0$. Assuming that every possible path of length n is sampled infinitely often, we update every possible state-step pair infinitely often. Thus, assume for contradiction that there is a state s and step r where $v(s, r) > 0$ but there exists a path of length n containing state s at position $n - r$. W.l.o.g. assume that r is minimal among all such state-step pairs with $v(s, r) > 0$. Clearly, $r > 0$ by definition of v. But, since r is minimal, we have that $v(s', r - 1) = 0$ for all reachable s'. Since there exists a path reaching s, all of its successors are reachable and hence have a value of 0. Consequently, the algorithm eventually updates $v(s, r) = 0$, contradicting the assumption. $\square$

4.3. **Implementing the function approximation.** To illustrate the flexibility of our approach, we sketch several ideas for the implementation of UPDATEBOUND and GETBOUND in Figure 6. A concrete discussion and implementation of the more complex approach is left for future work.
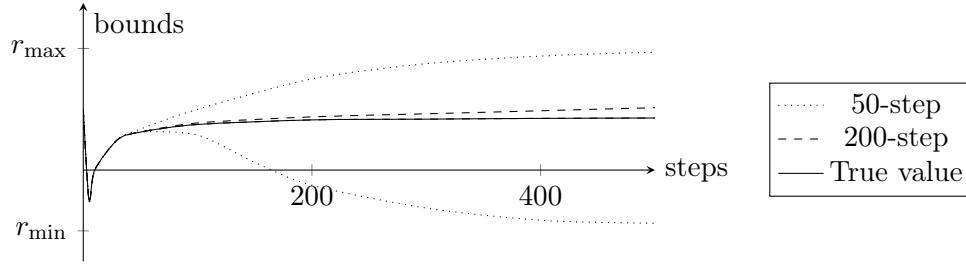
Figure 7. A schematic plot for an average reward extrapolation analysis on step bounded cores. The solid line represents the true value, while the dotted and dashed lines are the respective upper and lower bounds computed for a 50 and 200-step core. Note that the second dashed line (lower bound on the 200 core) coincides with the solid line (true value).

The first, trivial implementation is dense storage, i.e. explicitly storing a table mapping $S \times \{0, \ldots, n-1\} \to [0, 1]$. 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 is manually chosen. Note that if we choose $K = 1$ we again obtain the "dense" approach. This approach is depicted in Figure 6b for $K = 10$. Every black dot represents a stored value, the dashed lines represent the value returned by GetBound. We highlight that this approximation introduces accumulating errors. This may actually prevent convergence, since these accumulated errors alone could be larger than ε and thus the stopping criterion of the algorithm in Line 1 never is satisfied. To overcome this issue, our implementation repeatedly computes the exact values for all explored states with a simple n-step value iteration, updating the sparsely stored value appropriately. Note that such an update can be achieved with linear memory.

A more advanced idea is sketched in Figure 6c. Using more sophisticated function approximation methods, we could adaptively choose which values are stored. For example, there might be regions where the value of the function changes drastically and we should store more details there. In the figure, this happens around 20 and 40 steps, respectively.

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 $\mathbb{P}^{\max}[\lozenge^{\leq N} \overline{S_{\varepsilon,n}}]$ 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. This advantage becomes more apparent in the experimental evaluation. 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 in reality (unrecoverable) errors 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 far beyond the reach of existing tools. We first explain how stable cores can be used for efficient extrapolation to such large horizons.

For simplicity, we consider reachability and later argue how to transfer this idea to other objectives. We apply the ideas of interval iteration as used in, e.g., [HM14, BCC+14], as follows. Intuitively, since we have no knowledge of the partially explored states, we simply assume the worst and best case for them, i.e. assign a lower bound of 0 and an upper bound of 1. Furthermore, any explored target state is assigned a lower and upper bound of 1, as we know for sure that we reach our goal there. By applying interval iteration we can obtain bounds on the N-step and even unbounded reachability. Because of the core property, the bounds for $N \leq n$ necessarily are smaller than ε. But, for larger N, i.e. $N > n$, 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 approach allows us to get good approximations even for much larger bounds. We even observe that the computation of an n-step core and subsequent approximation of the desired property often is faster than directly computing the N-step property, as shown in our experimental evaluation.

For LTL and parity objectives, we can preprocess the obtained n-core by identifying the winning MECs and then applying the reachability idea. This yields bounds on the probability of satisfying the given objective 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 assigning the lower and upper bounds $r_{\min}$ and $r_{\max}$ to all unexplored states instead of 0 and 1. See Figure 7 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.

Note that we cannot use this method to obtain arbitrarily precise results. Given some n-step core and a particular (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 considering a particular part of the state space in order to obtain an ε-precise result. Nevertheless, the smaller n-step core may provide valuable insights in a system, quickly giving a good overview of its behaviour or potential design flaws. For example, an engineer could repeatedly run this analysis while formalizing or designing a system to quickly detect mistakes in the formalization or get a brief summarization of the systems performance. Moreover, the fact that the system drastically changes its behaviour after n steps may also provide valuable insights.

We recall 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 several examples, both the hand-crafted plane model and models from case studies. In the interest of space and readability, we hand-picked some noteworthy results from our overall evaluation. Further details together with evaluation results on the complete PRISM benchmark suite [KNP12], can be found in Section A.

5.1. **Implementation Details.** We implemented our approach in Java, using PRISM 4.5 [KNP02] as a library for parsing its modelling language and basic computations, verifying the correctness of our results. Our implementation supports Markov chains, continuous-time Markov chains (CTMC, via embedding or uniformization [Put94, 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. We point out that fine-tuning some parameters of the implementation (e.g., how often UpdateECs computes a full MEC decomposition) significantly impacts performance on some models. This suggests that by investing additional effort into choosing these parameters heuristically the runtime could be improved further.

In [BCC+14], the authors presented several potential sampling heuristics, i.e. implementations of GetPath. We evaluated some of the presented heuristics together with additional ones. As reported in [BCC+14], it turns out that first selecting an action maximizing the expected upper bound and then selecting a successor in a weighted, guided fashion yields the best overall performance. In particular, we sample a successor weighted by the respective upper bound, i.e. after selecting an action a we randomly select a successor state s' with probability proportional to $U(s') \cdot \Delta(s, a, s')$ or GetBound$(s', r) \cdot \Delta(s, a, s')$, respectively. A detailed explanation and comparison between different sampling heuristics is presented in Section A.1. In the following, we only consider the guided approach, since it consistently yielded the best performance.

5.2. **Models.** In our evaluation, we consider the following models. `airplane` is our running example from Figure 1 and Figure 5, respectively. All other models are taken from the PRISM benchmark suite [KNP12][2]. We briefly describe the models and how the associated parameters change them.

In `airplane`, the parameter `return` controls whether a return trip is possible and `size` quadratically influences the size of the "recovery" region. `zeroconf` [KNPS06] describes the IPv4 Zeroconf Protocol with `N` hosts, the number `K` of probes to send, and a probability `loss` of losing a message. `wlan` [KNS02] is a model of two WLAN stations in a fixed network topology sending messages on the shared medium, potentially leading to collisions. `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, converting it to a DTMC.

---

[2]Also available online at `http://www.prismmodelchecker.org/casestudies/`.

TABLE 1. Summary of our experimental results on several models and configurations for the unbounded and step-bounded 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 and a 100-step 10^{-6}-core, respectively. While building the step-bounded core for `brp`, we used the "dense" storage approach, since the simple approximation yielded unreliable performance.

Model	Param.	PRISM		Core		100-Core	
`zeroconf`	$100; 5; 0.1$	496,291	13 s	820	1 s	1,087	1 s
(`N`; `K`; `loss`)	$100; 10; 0.1$	$3.0 \cdot 10^6$	77 s	706	1 s	1,036	1 s
	$100; 15; 0.1$	$4.7 \cdot 10^6$	120 s	766	1 s	1,192	1 s
`airplane`	$100;$ `ff`	10,208	1 s	6	0 s	6	0 s
(`size`; `return`)	$100;$ `tt`	20,413	1 s	TIMEOUT		33	0 s
	$10000;$ `ff`	MEMOUT		6	0 s	6	0 s
	$10000;$ `tt`	MEMOUT		TIMEOUT		32	0 s
`brp`	$20; 10$	2,933	0 s	1,437	1 s	1,359	0 s
(`N`; `MAX`)	$20; 100$	26,423	1 s	1,442	1 s	1,324	0 s
	$20; 1000$	261,323	8 s	1,437	1 s	1,336	0 s
`wlan`	—	345,000	8.2 s	345,000	74 s	35,998	46 s
`cyclin`	4	431,101	18 s	TIMEOUT		11,465	4 s
(`N`)	5	$2.3 \cdot 10^6$	118 s	TIMEOUT		36,613	13 s

5.3. **Results.** We evaluated our implementation on an Intel Xeon `E5-2630` 2.20 GHz CPU, allocating one core (using `taskset`) and 8 GB of RAM to the Java process (using `-Xmx8G`). We used a default precision of 10^{-6} and a timeout of 15 minutes for all experiments. The evaluation is performed with the help of GNU `parallel` [Tan11]. The results for the infinite and finite construction are summarized in Table 1. We 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 [BCC+14], 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 several consecutive collisions and message losses, which is very unlikely. Consequently, a very small part of the model already satisfies the core property. The size of the core remains practically constant when increasing the parameter `K`, as only unimportant states are added to the system. We note that the order of magnitude of explored states is very similar to the experiments from [BCC+14]. Similarly, in the `airplane` model a significant number of states is dedicated to recovering from an unlikely error. Hence, a small core exists independent 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.
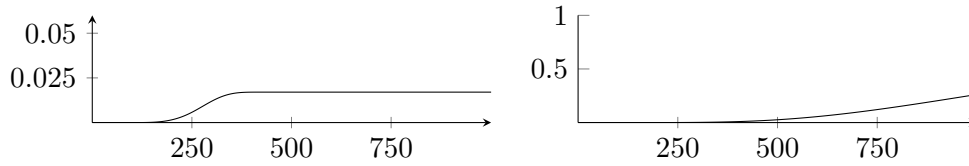
FIGURE 8. Stability analysis of the `wlan` (left) and `cyclin(N = 4)` (right) 100-step core. The graphs show the probability of exiting the respective core within the given amount of steps. The y axis of the `wlan` graph is scaled.

The `airplane` model with return trip (and `cyclin` to a lesser extent) actually shows a structural weakness of our purely sampling-/VI-based approach: Recall that the "non-recovery" region, i.e. the round-trip path before an error occurs, is not a MEC, however the probability of exiting is very low, namely $\tau \ll \varepsilon$ per round-trip. This leads to two problems. Firstly, any sampling based approach which is influenced by the transition probabilities (including our weighted approach) only rarely explores the eventually important recovery region. Secondly, even if a path is sampled in that region the update-computation only propagates a miniscule fraction of the obtained information back to the round trip states. Here, a hybrid approach combined with strategy iteration might be useful. We emphasize that this not an inherent issue of the "core" idea, but rather an inherent issue of value iteration—computing a reachability property on this model using value iteration takes very long due to the latter reason, too.

Comparison to [BCC+14]**:** We also executed the tool presented in [BCC+14] where applicable (only MDP are supported). We tested the tool both with an unsatisfiable property ($\Diamond\emptyset$), 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 [BCC+14], since it was suggested to be the best-performing setting. Especially on the $\Diamond\emptyset$ 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, especially since the number of explored states was similar.

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. 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 when using the "simple" approximation. Nevertheless, 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.

During our experiments, we used the "simple" approximation approach ($K = 5$) introduced in Section 4.3. Interestingly, this approach actually yielded significant speed-ups and a smaller core on the `cyclin` model compared to using the "dense" approximation. On the other hand, "dense" terminated much faster with a comparable core size on both `wlan` and `brp`, with a speed-up of nearly an order of magnitude. We conjecture that this difference potentially is related to `cyclin` being a uniformized CTMC, resulting in a particular structure.

Finally, we applied the idea of stability from Section 4.2 on the `wlan` and `cyclin` models, with results outlined in Figure 8. Interestingly, for the `wlan` model, the escape probability stabilizes at roughly 0.017 and we obtain the exact same probability for *all* heuristics (see
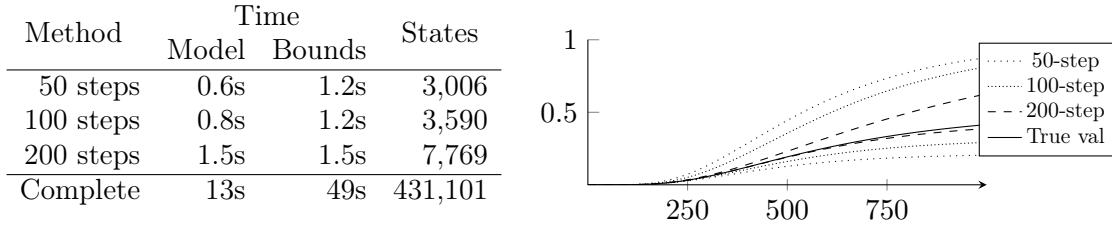
Method	Time		States
	Model	Bounds	
50 steps	0.6s	1.2s	3,006
100 steps	0.8s	1.2s	3,590
200 steps	1.5s	1.5s	7,769
Complete	13s	49s	431,101

FIGURE 9. 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.

Section A.1), 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. Recall that the `wlan` model has roughly $3.5 \cdot 10^5$ states in total, while the identified subsystem comprises only 10% of these states. This means that most of the long term behaviour is described by only a fraction of the states. Additionally, we observe that the crucial actions leading to the other 90% of the state space happen at around 200–400 steps and the system is stable afterwards. This information is only visible since we considered the stability function as a whole instead of a single number.

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 5.5% 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 9. 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[3] over 10 times faster by only building the 200-step core instead of constructing the full model. In particular, we obtain the result significantly faster than the construction of the whole 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. *PRISM benchmark suite.* For readability, we briefly summarize our findings from Section A.3 here. Firstly, we observe that our methods are sensitive to the structure of the model and the particular guidance heuristic used. Weighted guidance, incorporating both computed bounds and transition probabilities, shows the best potential out of the investigated heuristics. Moreover, the results suggest that small cores are not too common in the models of the PRISM benchmark suite. However, we conjecture that this is mostly due to the specific structure of these models. In particular, most of them describe abstract protocols, where probabilistic branching is only used in a few critical locations and is of a special structure; a significant part of the system size stems from non-deterministic interleaving

---

[3]We used the (arbitrarily chosen) query `dim > CYCLIN / 4`.

of parallel processes. For example, in the `firewire` protocol, randomness is only used to select either a `fast` or `slow` mode to eventually resolve ties. This often results in rather large probabilities and thus hardly any "unimportant" states. For such models, our method naturally is not applicable. However, given real-world constraints, many low-probability events are introduced to the model, e.g., hardware failures, sensor noise, or transmission errors due to environmental influences. These low probability errors allow for non-trivial cores, as is the case with, for example, the `zeroconf` or `brp` model.

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 6c. Note that this 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 6, the regions around 20 and 40 steps, respectively, seems to be of significance.

Moreover, a 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 terms of performance, one could consider parallelizing the sampling procedures and applying rare-event detection mechanisms to reduce the number of samples needed to find a core. Another point for performance improvements is the use of faster MEC detection algorithms. In particular, [CH11] presents an incremental MEC decomposition algorithm which is able to maintain the set of MECs of a dynamically changing MDP. Currently, our implementation recomputes the ECs from scratch every time.

In the spirit of [BCC$^+$14], our approach also could be extended to a PAC algorithm for black-box systems. Extensions to continuous time systems are also possible. Practically, application of our methods to models derived from biological systems or chemical reactions could yield more satisfactory results, since such systems are much more "probabilistic" than the abstract protocols considered in our evaluation.

Further interesting variations are cores for discounted objectives [SWWY18] 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 much wider range of analysis.

References

[ABHK18]   Pranav Ashok, Yuliya Butkova, Holger Hermanns, and Jan Křetínskỳ. Continuous-time Markov decisions based on partial exploration. In *ATVA*, pages 317–334. Springer, 2018.

[ACD+17]   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.

[BCC+14]   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.

[Bel57]   Richard Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

[Ber12]   Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II: Approximate Dynamic Programming*. Athena Scientific, 2012.

[BK08]   Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[CH11]   Krishnendu Chatterjee and Monika Henzinger. Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification. In *SODA*, pages 1318–1336, 2011.

[CH12]   Krishnendu Chatterjee and Monika Henzinger. An $O(n^2)$ time algorithm for alternating büchi games. In *SODA*, pages 1386–1399. SIAM, 2012.

[CH14]   Krishnendu Chatterjee and Monika Henzinger. Efficient and dynamic algorithms for alternating büchi games and maximal end-component decomposition. *J. ACM*, 61(3):15:1–15:40, 2014.

[CNZ17]   Krishnendu Chatterjee, Petr Novotný, and Dorde Zikelic. Stochastic invariants for probabilistic termination. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 145–160. ACM, 2017.

[CY95]   Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995.

[DA97]   Luca De Alfaro. *Formal verification of probabilistic systems*. Number 1601. Citeseer, 1997.

[DJJL02]   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.

[DJKV17]   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.

[HHWZ10]   Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. PASS: abstraction refinement for infinite probabilistic models. In *TACAS*, pages 353–357. Springer, 2010.

[HK19]   Arnd Hartmanns and Benjamin Lucien Kaminski. Optimistic value iteration. *CoRR*, abs/1910.01100, 2019.

[HM14]   Serge Haddad and Benjamin Monmege. Reachability in MDPs: Refining convergence of value iteration. In *International Workshop on Reachability Problems*, pages 125–137. Springer, 2014.

[KKKW18]   Edon Kelmendi, Julia Krämer, Jan Kretínský, and Maximilian Weininger. Value iteration for simple stochastic games: Stopping criterion and learning algorithm. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, volume 10981 of *Lecture Notes in Computer Science*, pages 623–642. Springer, 2018.

[KM19]   Jan Kretínský and Tobias Meggendorfer. Of cores: A partial-exploration framework for markov decision processes. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 5:1–5:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[KNP02]   M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *TOOLS*, pages 200–204, 2002.

[KNP12]   Marta Z. Kwiatkowska, Gethin Norman, and David Parker. The PRISM benchmark suite. In *QEST*, pages 203–204. IEEE Computer Society, 2012.

[KNPS06]   Marta Kwiatkowska, Gethin Norman, David Parker, and Jeremy Sproston. Performance analysis of probabilistic timed automata using digital clocks. *FMSD*, 29(1):33–78, 2006.

[KNS02]   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.
[MLG05]   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, 2005.
[Put94]   M.L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John
           Wiley and Sons, 1994.
[QK18]   Tim Quatmann and Joost-Pieter Katoen. Sound value iteration. In *CAV (1)*, volume 10981 of
           LNCS, pages 643–661. Springer, 2018.
[SWWY18]   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.
[Tan11]   O. Tange. Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–
           47, Feb 2011.
[Tar72]   Robert Tarjan. Depth-first search and linear graph algorithms. *SICOMP*, 1(2):146–160, 1972.
[Whi88]   Douglas J White. Further real applications of markov decision processes. *Interfaces*, 18(5):55–61,
           1988.
[Whi93]   Douglas J White. A survey of applications of markov decision processes. *Journal of the operational
           research society*, 44(11):1073–1096, 1993.

Appendix A. Evaluation Details

In this section, we provide further evaluation data and implementation details.

A.1. **Sampling heuristics.** All of our implementations of GetPath sample paths originating from the initial state. We consider two classes of sampling heuristics namely *action-based* and *graph-based*.

For action-based heuristics, we first select an action maximizing the expected upper bound, i.e. randomly choose an action from $\arg\max_{a \in \mathsf{Av}(s)} \sum_{s' \in S} \Delta(s, a, s') \cdot U(s')$. Once such an action a is selected, we obtain a successor based on a scoring function $f(s, a, s')$, i.e. a state s' is selected with probability proportional to $f(s, a, s')$. The graph-based approach however essentially ignores the available actions and chooses successors directly based on a scoring function $f(s, s')$.

We considered three action-based heuristics, namely

- `PROB` (P): $f(s, a, s') = \Delta(s, a, s')$,
- `WEIGHTED` (W): $f(s, a, s') = U(s') \cdot \Delta(s, a, s')$, and
- `DIFFERENCE` (D): $f(s, a, s') = U(s')$.

Moreover, we considered the following graph-based heuristics

- `GRAPH-WEIGHTED` (GW): $f(s, s') = U(s') \cdot \max_{a \in \mathsf{Av}(s)} \Delta(s, a, s')$, and
- `GRAPH-DIFFERENCE` (GD): $f(s, s') = U(s')$.

In the finite-horizon case, the heuristics use GetBound(s', r) instead of $U(s')$. The `DIFFERENCE` heuristics corresponds to the `M-D` mode of [BCC+14], originally proposed in [MLG05].

A.2. **Setup.** We evaluated our methods on the complete PRISM benchmark set where applicable, i.e. all DTMC, MDP, and CTMC. We treated all CTMCs via unifomization, determining the uniformization constant using PRISM's `getDefaultUniformisationRate()` method (which equals 1.02 times the maximum exit rate). For each considered model / parameter combination (215 in total), we (i) built the complete model using PRISM's methods (ii) built an unbounded core and (iii) built step bounded cores for $n \in \{10, 100, 200, 500\}$, using each of our 5 heuristics, resulting in 26 executions per model and 5590 runs in total. All cores are built with a precision requirement of $\varepsilon = 10^{-6}$. Each execution was run with a time limit of 15 minutes and memory limit of 8 GB, bound to a single CPU core to avoid influences of potential parallelism. Since a significant part of these models are quite large, exceeding the imposed limits by a huge margin, we encountered several time- and mem-outs. For each evaluation, we thus describe in detail how we treated such failures when aggregating the values.

We note that the models of the PRISM benchmark suite are not too well suited for our methods overall. We explain this issue in the following section. Nevertheless, we evaluated our methods on this dataset, since it is one of the largest established datasets for this purpose.

A.3. **Evaluation Results.** In the first evaluation, we compare the overall performance of our methods and compare the different heuristics. To this end, we compute for each method the average number of states, fraction of states compared to the original model, the number of failures, and the number of times the method succeeded where the complete model construction did not succeed. The results are presented in Table 2. Since a large number of these models comprise a single connected component (SCC or MEC), we separately consider all models where (i) the complete model construction finished and (ii) several components are found (or one not too large one), of which there are 117 in total. The results are shown in Table 3.

Overall, we see that the unguided, random sampling heuristic P often is significantly outperformed by the guided approaches W and D in terms of runtime. Note that both the "Time" and "States" column only shows averages over all instances where the particular method succeeded, hence the averages size of P-cores seem lower than the others'. When comparing the instances where all heuristics succeed, we see that the average sizes are very similar. We also observe that the size and construction time (and failure rates) of step-bounded does not significantly increase when going from 200 to 500 steps. This suggests that most of the considered models where the construction succeeds are quite "stable" after 200 steps—otherwise, either the average size of the computed 500-step cores or the number of timeouts would be much larger. Finally, we see that the size of the identified cores are mostly independent of the used method (accounting for the difference in the averages due to timeouts). When repeating the experiments, we further learned that the size of the identified cores usually deviate only by a few percent. This suggests that the performance of our approach is quite stable, despite the large amount of involved randomization.

Since the benchmark set contains many vastly different models, both in terms of structure and size, we further report results grouped by the different model types for the unbounded core construction in Table 4. As already expected, we see that our methods are highly dependent on the model structure. Moreover, we see that on all models except `zeroconf`, the weighted approaches W and GW obtain the smallest core (or one of equal size). In particular, on the "embedded" model, W and GW obtain a 50% reduction while the other approaches do not identify a non-trivial core. This suggests that out of the considered heuristics, the weighted approach W / GW is the most efficient one on average (in terms of size). These specific differences also suggest that our methods are sensitive to the particular heuristic, and developing further, more sophisticated methods can significantly improve performance.

While these results suggest that cores seem to be relatively rare in practice, we point out that this is mostly due to the specific structure of the benchmarks in the PRISM benchmark suite. Some models are mostly non-deterministic, containing only a few probabilistic branches with "large" probabilities. For example, "csma" describes a shared bus where the only source of probabilistic branching is choosing a random back-off delay after a collision—the smallest involved probability is in the order of 1/10th. Most models in the middle part of the table are of this structure. On other classes, the parameters of the benchmark suite simply are not chosen large enough in order to obtain "unimportant" states. This is the case with the "brp" model, where scaling up the maximal number of retransmissions can drastically improve the savings of our core approach, as we have shown in Section 5. Finally, many models are infinitely repeating protocols and thus are strongly connected. In light of Proposition 4.1, we cannot expect any improvement there. All models of this kind are located in the bottom group of the table.

Table 2. Comparison of our heuristics on the complete PRISM benchmark suite (215 instances in total). The "Time" and "States" column are (arithmetic) averages over all instances solved by the particular method. "Fraction" describes the average fraction of states compared to the whole model, i.e. $|S_\varepsilon|/|S|$, considering only those instances where both the particular heuristic and the complete construction succeeded. "Failures" and "Success" are the percentage of models where the complete construction succeeded but the heuristic failed and vice versa. For example, the third line ("Unbounded W") means that the W heuristic for unbounded cores took an average of 32 seconds to converge, resulting in 342306 states, with only 75% of the states of the complete model built. Finally, on 42% of the models this method failed but the complete construction finished in time, dually on 7% of the models the heuristic succeeded but the complete construction did not.

	Time	States	Fraction	Failures	Success
Complete	7 s	268880	—	21 %	—
Unbounded					
W	32 s	342306	75 %	42 %	7 %
P	44 s	149538	73 %	61 %	0 %
D	28 s	344932	81 %	39 %	6 %
GW	34 s	339944	75 %	41 %	7 %
GD	33 s	362027	81 %	42 %	6 %
Bounded 10					
W	5 s	5579	15 %	3 %	21 %
P	11 s	349	13 %	12 %	19 %
D	4 s	14738	18 %	5 %	21 %
GW	6 s	5763	15 %	3 %	21 %
GD	5 s	14741	18 %	5 %	21 %
Bounded 100					
W	39 s	29387	52 %	29 %	6 %
P	127 s	5047	50 %	49 %	5 %
D	31 s	24935	65 %	33 %	5 %
GW	34 s	28812	51 %	29 %	6 %
GD	22 s	24405	64 %	33 %	5 %
Bounded 200					
W	41 s	22415	61 %	35 %	3 %
P	127 s	7424	64 %	60 %	1 %
D	35 s	24796	74 %	45 %	1 %
GW	40 s	22665	60 %	33 %	3 %
GD	32 s	23981	73 %	44 %	1 %
Bounded 500					
W	37 s	14926	66 %	38 %	3 %
P	96 s	9334	74 %	67 %	0 %
D	39 s	14723	77 %	46 %	1 %
GW	37 s	14084	65 %	37 %	3 %
GD	34 s	13983	76 %	44 %	1 %

Table 3. Comparison of our heuristic on models of the PRISM benchmark
suite which are not strongly connected (117 in total). We use the same metrics
as in Table 2. We determine whether a model is strongly connected based
on the results of the complete construction, hence only models where this
construction succeeds are included. Consequently, the "Success" column is
not relevant.

	Time	States	Fraction	Failures
Complete	4 s	217159	—	—
Unbounded				
W	14 s	121213	74 %	9 %
P	40 s	72114	72 %	32 %
D	12 s	122865	79 %	9 %
GW	16 s	121752	74 %	11 %
GD	15 s	123926	79 %	11 %
Bounded 10				
W	0 s	273	11 %	6 %
P	2 s	244	10 %	6 %
D	0 s	395	13 %	6 %
GW	0 s	273	11 %	6 %
GD	0 s	400	13 %	6 %
Bounded 100				
W	33 s	20343	54 %	24 %
P	84 s	7388	55 %	40 %
D	25 s	19975	61 %	19 %
GW	27 s	19479	53 %	23 %
GD	15 s	19228	59 %	19 %
Bounded 200				
W	46 s	25642	65 %	25 %
P	96 s	10169	69 %	48 %
D	38 s	29370	73 %	34 %
GW	46 s	25870	63 %	21 %
GD	39 s	29614	72 %	31 %
Bounded 500				
W	36 s	15823	69 %	28 %
P	92 s	10739	71 %	48 %
D	42 s	16431	77 %	34 %
GW	42 s	15888	69 %	25 %
GD	36 s	15368	76 %	31 %

In general, most of the models in the benchmark suite describe abstract protocols. For
these, probabilistic branching is only present in few critical locations and is of a particular
structure—mostly, randomness is used to, e.g., resolve ties or similar, resulting in rather large
probabilities. Here, our method is, by nature, not applicable. However, as soon as real-life
constraints are incorporated, many low-probability events are introduced to the model,

TABLE 4. Comparison of our heuristics on each of the model groups of the PRISM benchmark suite. In the first four columns, we report the number of instances, the average number of states (of the models where the complete construction succeeded), the absolute number of failures of the complete construction and the average number of components of the model instances (MECs or SCCs, respectively). For each heuristic, we then report the average fraction of states, compared to the complete construction (the previous tables' "Fraction" column). We omit the % sign to save space. For simplicity, whenever a heuristic failed, we simply assumed a ratio of 100%, independent of the actual outcome of the complete construction. Further, we report the number of times where the heuristic approach succeeded over the complete construction (the previous tables' "Success" column). For readability, we sorted the models by the average state-reduction achieved by our methods.

Model	#	States	Comp.	Fail	W		P		D		GW		GD	
zeroconf	16	383919	5476	0	12	0	11	0	15	0	13	0	15	0
zeroconf_dl	10	91996	3820	0	23	0	19	0	29	0	26	0	34	0
embedded	7	6013	4529	0	52	0	100	0	100	0	52	0	100	0
wlan	7	969161	1	0	77	0	100	0	77	0	77	0	77	0
nand	10	193247	193247	6	75	6	100	0	96	4	75	6	96	4
brp	12	2302	2302	0	97	0	96	0	100	0	97	0	100	0
wlan_dl	7	596250	11671	4	100	4	100	0	100	4	100	4	100	4
crowds	16	87287	56140	4	100	3	100	0	100	3	100	3	100	3
coin	6	11616	27	0	100	0	100	0	100	0	100	0	100	0
csma	9	389136	11	3	100	0	100	0	100	0	100	0	100	0
egl	16	95230	95230	12	100	0	100	0	100	0	100	0	100	0
firewire	20	342520	1214	4	100	1	100	1	100	1	100	1	100	1
herman	7	6241	5	0	100	0	100	0	100	0	100	0	100	0
leader_sync	9	758	633	0	100	0	100	0	100	0	100	0	100	0
cluster	9	396800	1	1	100	0	100	0	100	0	100	0	100	0
fms	10	339031	1	3	100	0	100	0	100	0	100	0	100	0
kanban	7	612813	1	2	100	0	100	0	100	0	100	0	100	0
mapk_cascade	8	316918	1	2	100	0	100	0	100	0	100	0	100	0
poll	18	419430	1	3	100	0	100	0	100	0	100	0	100	0
tandem	11	310465	1	2	100	0	100	0	100	0	100	0	100	0

for example hardware failures, sensor noise, or transmission errors due to environmental influences. These low probability errors allow for non-trivial cores, as is the case with, for example, the zeroconf model.