POWVER

Technical Report 2018-09

Title: **Continuous-Time Markov Decisions based on Partial Exploration**

Authors: Pranav Ashok, Yuliya Butkova, Holger Hermanns, Jan Křetínský

Report Number: 2018-09

ERC Project: Power to the People. Verified.

ERC Project ID: 695614

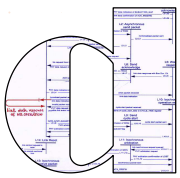
Funded Under: H2020-EU.1.1. – EXCELLENT SCIENCE

Host Institution: Universität des Saarlandes, Dependable Systems and Software
Saarland Informatics Campus

Published In: ATVA 2018

Please cite this publication as follows:

Pranav Ashok, Yuliya Butkova, Holger Hermanns, Jan Křetínský.
Continuous-Time Markov Decisions based on Partial Exploration.
Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings. Lecture Notes in Computer Science 11138, Springer 2018, ISBN 978-3-030-01089-8. 317-334.



POWER TO THE PEOPLE.
VERIFIED.
erc

Continuous-Time Markov Decisions based on Partial Exploration*

Pranav Ashok[1][0000−0002−1083−4741], Yuliya Butkova[2][0000−0002−9678−1467], Holger Hermanns[2][0000−0002−2766−9615], and Jan Křetínský[1][0000−0002−8122−2881]

[1] Technical University of Munich, Germany
{ashok | jan.kretinsky}@in.tum.de
[2] Saarland University, Saarbrücken, Germany
{butkova | hermanns}@cs.uni-saarland.de

Abstract. We provide a framework for speeding up algorithms for time-bounded reachability analysis of continuous-time Markov decision processes. The principle is to find a small, but almost equivalent *subsystem* of the original system and only analyse the subsystem. Candidates for the subsystem are identified through simulations and iteratively enlarged until runs are represented in the subsystem with high enough probability. The framework is thus dual to that of abstraction refinement. We instantiate the framework in several ways with several traditional algorithms and experimentally confirm orders-of-magnitude speed ups in many cases.

1 Introduction

Continuous-time Markov decision processes (CTMDP) [Ber95, Sen99, Fei04] are the natural real-time extension of (discrete-time) Markov decision processes (MDP). They can likewise be viewed as non-deterministic extensions of continuous-time Markov chains (CTMC). As such, CTMDP feature probabilistic and non-deterministic behaviour as well as random time delays governed by exponential probability distributions. Prominent application areas of CTMDP include operations research [BDF81, Fei04], power management and scheduling [QQP01], networked, distributed systems [HHK00, GGL03], as well as epidemic and population processes [Lef81]. Moreover, CTMDPs are the core semantic model underlying formalisms such as generalised stochastic Petri nets, Markovian stochastic activity networks, and interactive Markov chains [EHKZ13].

A large variety of properties can be expressed using logics such as CSL [ASSB96]. Apart from classical techniques from the MDP context, the analysis of such properties relies fundamentally on the problem of time-bounded reachability (TBR), i.e. *what is the maximal/minimal probability to reach a given state within a given time bound.* Since this is the cornerstone of the analysis, a manifold of algorithms have been proposed for TBR [BHHK04, BFK$^+$09, NZ10, FRSZ11, BS11, HH13, BHHK15]. While the algorithmic approaches are diverse, relying on uniformisation and various

* This research was supported in part by Deutsche Forschungsgemeinschaft (DFG) through the TUM International Graduate School of Science and Engineering (IGSSE) project 10.06 *PARSEC*, the ERC through Advanced Grant 695614 (POWVER), the Czech Science Foundation grant No. 18-11193S, and the DFG project 383882557 *Statistical Unbounded Verification.*

forms of discretization, they are mostly back-propagating the values computed, i.e. in the form of value iteration.

Not surprisingly, all these algorithms naturally process the state space of the CTMDP in its entirety. In this work we instead suggest a framework that enables TBR analysis with guaranteed precision while often exploring only a small, property-dependent part of the state space. In contrast to widely used abstraction-based approaches, which consider a quotient of the system, we consider subsystems of the original system, where only some behaviour is present. This approach is thus dual to that of abstraction. Similar ideas have appeared for (discrete-time) MDPs and unbounded reachability [BCC$^+$14] or mean payoff [ACD$^+$17]. These techniques are based on *asynchronous* value-iteration approaches, originally proposed in the probabilistic planning world, such as bounded real-time dynamic programming (BRTDP) [MLG05]. Intuitively, the back-propagation of values (value iteration steps) are not performed on all states in each iteration (synchronously), but always only the "interesting" ones are considered (asynchronously); in order to bound the error in this approach, one needs to compute both an under- and an over-approximation of the actual value.

In other words, the main idea is to keep track of (under- and over-)approximation of the value when accepting that we have no information about the values attained in certain states. Yet if we can determine that these states are reached with very low probability, their effect on the actual value is provably negligible and thus the lack of knowledge only slightly increases the difference between the under- and over-approximations. To achieve this effect, the algorithm of [BCC$^+$14] alternates between two steps: (i) simulating a run of the MDP using a (hopefully good) scheduler, and (ii) performing the standard value iteration steps on the states visited by this run.

It turns out that this idea cannot be transferred to the continuous-time setting easily. In technical terms, the main issue is that the value iteration in this context takes the form of synchronous back-propagation, which when implemented in an asynchronous fashion results in memory requirements that tend to dominate the memory savings expectable due to partial exploration.

Therefore, we twist the above approach and present a yet simpler algorithmic strategy in this paper. Namely, our approach alternates between several simulation steps, and a subsequent run of TBR analysis only focussed on the already explored subsystem, instead of the entire state space. If the distance between under- and over-approximating values is small enough, we can terminate; otherwise, running more simulations extends the considered state subspace, thereby improving the precision in the next round. If the underlying TBR analysis provides an optimal scheduler along with the value of time-bounded reachability, then our solution as well provides the optimal scheduler for the TBR problem on the given CTMDP.

There are thus two largely independent components to the framework, namely (i) a heuristic how to explore the system via simulation, and (ii) an algorithm to solve time-bounded reachability on CTMDP. The latter is here instantiated with some of the classic algorithms mentioned above, namely the first discretization-based algorithm [NZ10] and the two most competitive improvements over it [BS11, BHHK15], based on uniformisation and untimed analysis. The former basically boils down to constructing a scheduler resolving the non-determinism effectively. We instantiate this exploration heuristics in two ways. Firstly, we consider a scheduler returned by the most recent run of the respective TBR algorithm, assuming this to yield a close-to-optimal

scheduler, so as to visit the most important parts of the state space, relative to the property in question. Secondly, since this scheduler may not be available when working with TBR algorithms that return only the value, we also employ a scheduler resolving choices uniformly. Although the latter may look very straightforward, it turns out to already speed up the original algorithm considerably in many cases. This is rooted in the fact that that scheduler best represents the available knowledge, since the uniform distribution is the one with maximimal entropy.

Depending on the model and the property under study, different ratios of the state space entirety need to be explored to achieve the desired precision. Furthermore, our approach is able to exploit that the reachability objective is of certain forms, in stark contrast to the classic algorithm that needs to perform the same computation irrespective of the concrete set of target states. Still, the approach we propose will naturally profit from future improvements in effectiveness of classic TBR analysis.

We summarize our contribution as follows:

- We introduce a framework to speed up TBR algorithms for CTMDP and instantiate it in several ways. It is based on a partial, simulation-based exploration of the state space spanned by a model.
- We demonstrate its effectiveness in combination with several classic algorithms, obtaining orders of magnitude speed ups in many experiments. We also illustrate the limitations of this approach on cases where the state space needs to be explored almost in its entirety.
- We conclude that our framework is a generic add-on to arbitrary TBR algorithms, often saving considerably more work than introduced by its overhead.

2 Preliminaries

In this section, we introduce some central notions. A *probability distribution* on a finite set X is a mapping $\rho : X \to [0, 1]$, such that $\sum_{x \in X} \rho(x) = 1$. $\mathcal{D}(X)$ denotes the set of all probability distributions on X.

Definition 1. *A* continuous-time Markov decision process *(CTMDP) is a tuple* $\mathcal{M} = (s_{init}, S, Act, \mathbf{R}, G)$ *where* S *is a finite set of* states*,* s_{init} *is the initial state,* Act *is a finite set of* actions*,* $\mathbf{R} : S \times Act \times S \to \mathbb{R}_{\geq 0}$ *is a rate matrix and* $G \subseteq S$ *is a set of* goal *states.*

For a state $s \in S$ we define the set of *enabled actions* $\mathrm{Act}(s)$ as follows: $\mathrm{Act}(s) = \{\alpha \in \mathrm{Act} \mid \exists s' \in S : \ \mathbf{R}(s, \alpha, s') > 0\}$. States s' for which $\mathbf{R}(s, \alpha, s') > 0$ form the set of *successor states of* s *via* α, denoted as $\mathrm{Succ}(s, \alpha)$. W. l. o. g. we require that all sets $\mathrm{Act}(s)$ and $\mathrm{Succ}(s, \alpha)$ are non-empty. A state s, s. t. $\forall \alpha \in \mathrm{Act}(s) : \ \mathrm{Succ}(s, \alpha) = \{s\}$ is called *absorbing*.

For a given state s and action $\alpha \in \mathrm{Act}(s)$, we denote by $\lambda(s, \alpha) = \sum_{s'} \mathbf{R}(s, \alpha, s')$ the *exit rate* of α in s and $\Delta(s, \alpha, s') = \mathbf{R}(s, \alpha, s')/\lambda(s, \alpha)$.

An example CTMDP is depicted in Fig. 1a. Here states are depicted in circles and are labelled with numbers from 0 to 5. The goal state G is marked with a double circle. Dashed transitions represent available actions, e.g. state 1 has two enabled actions α and β. A solid transition labelled with a number denotes the rate, e.g. $\mathbf{R}(1, \beta, G) = 1.1$, therefore there is a solid transition from state 1 via action β to state G with rate 1.1. If there is only one enabled action for a state, we only show

the rates of the transition via this action and omit the action itself. For example, state 0 has only 1 enabled action (lets say α) and therefore it only has outgoing solid transition with rate $1.1 = \mathbf{R}(0, \alpha, 1)$.

The system starts in the initial state $s_0 = s_{\text{init}}$. While being in a state s_0, the system picks an action $\alpha_0 \in \text{Act}(s)$. When an action is picked the CTMDP resides in s_0 for the amount of time t_0 which is sampled from exponential distribution with parameter $\lambda(s_0, \alpha_0)$. Later in this paper we refer to this as *residence time* in a state. After t_0 time units the system transitions into one of the successor states $s_1 \in \text{Succ}(s_0, \alpha_0)$ selected randomly with distribution $\Delta(s_0, \alpha_0, \cdot)$. After this transition the process is repeated from state s_1 forming an *infinite path* $\rho = s_0 \xrightarrow{\alpha_0, t_0} s_1 \xrightarrow{\alpha_1, t_1} s_2 \ldots$. A finite prefix of an infinite path is called a *(finite) path*. We will use $\rho\downarrow$ to denote the last state of a finite path ρ. We will denote the set of all finite paths in a CTMDP with $Paths^*$, and the set of all infinite paths with $Paths$.

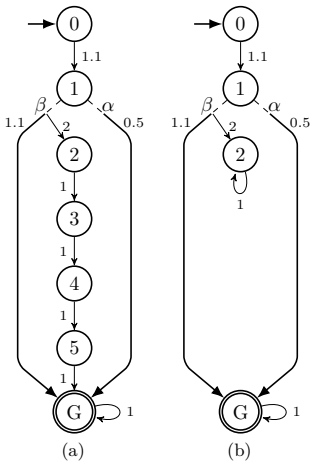


Fig. 1. Example CTMDPs.

CTMDPs pick actions with the help of *schedulers*. A scheduler is a measurable[3] function $\pi : Paths^* \times \mathbb{R}_{\geqslant 0} \to \mathcal{D}(\text{Act})$ such that $\pi(\rho, t) \in \text{Act}(\rho\downarrow)$. Being in a state s at time point t the CTMDP samples an action from $\pi(\rho, t)$, where ρ is the path that the system took to arrive in s. We denote the set of all schedulers with Π.

Fixing a scheduler π in a CTMDP $\mathcal{M}$, the unique probability measure $\Pr_\pi^{\mathcal{M}}$ over the space of all infinite paths can be obtained [Neu10], denoted also by $\Pr_\pi$ when $\mathcal{M}$ is clear from context.

Optimal Time-Bounded Reachability

Let $\mathcal{M} = (s_{\text{init}}, S, \text{Act}, \mathbf{R}, G)$ be a CTMDP, $s \in S$, $T \in \mathbb{R}_{\geqslant 0}$ a time bound, and $\text{opt} \in \{\sup, \inf\}$. The *optimal (time-bounded) reachability probability* (or *value*) of state s in $\mathcal{M}$ is defined as follows:

$$\text{val}_{\mathcal{M}}^s(T) := \text{opt}_{\pi \in \Pi} \Pr_\pi^{\mathcal{M}} \left[\Diamond^{\leqslant T} G \right],$$

where $\Diamond^{\leqslant T} G = \{ s_0 \xrightarrow{\alpha_0, t_0} s_1 \xrightarrow{\alpha_1, t_1} s_2 \ldots \mid s_0 = s \wedge \exists i : s_i \in G \wedge \sum_{j=0}^{i-1} t_j \leq T \}$ is the set of paths starting from s and reaching G before T.

The *optimal (time-bounded) reachability probability* (or *value*) of $\mathcal{M}$ is defined as $\text{val}_{\mathcal{M}}(T) = \text{val}_{\mathcal{M}}^{s_{\text{init}}}(T)$. A scheduler that achieves optimum for $\text{val}_{\mathcal{M}}(T)$ is the *optimal scheduler*. A scheduler that achieves value v, such that $||v - \text{val}_{\mathcal{M}}(T)||_\infty < \varepsilon$ is called *ε-optimal*.

3 Algorithm

In this work we target CTMDPs that have large state spaces, but only a small subset of those states is actually contributing significantly to the reachability probability.

[3] Measurable with respect to the standard σ-algebra on the set of paths [NZ10].

Consider, for example, the *polling system* represented schematically in Figure 2. Here two stations store continuously arriving tasks in a queue. Tasks are to be processed by a server. If the task is processed successfully it is removed from the queue, otherwise it is returned back into the queue. State space of the CTMDP $\mathcal{M}$ modelling this polling system is a tuple (q_1, q_2, s), where q_i is the amount of tasks in queue i and s is a state of the server (could be e. g. *processing task, awaiting task*, etc.).

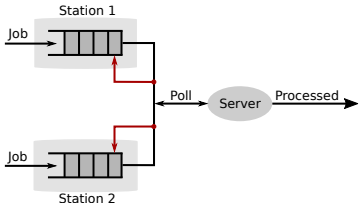


Fig. 2. Schematic representation of polling system.

One of the possible questions could be, for example, *what is the maximum probability of both queues to be full after a certain time point*. This corresponds to goal states being of the form (N, N, s), where N is the maximal queue capacity and s – any state of the server. Given that both queues are initially empty, all the paths reaching goal states have to visit states $(q_1, q_2, \cdot)$, where $q_i = [0..N]$. However, for similar questions, for example, *what is the maximum probability of the first queue to be full after a certain time point*, the situation changes. Here goal states are of the form (N, q_2, s), where $q_2 = 0..N$ and s – any state of the server. The scheduler that only extracts tasks from the second queue is the fastest to fill the first one and is therefore the optimal one. The set of states that are most likely visited when following this scheduler are those states where the size of the second queue is small. This naturally depends on the rates of task arrival and processing. Assuming that the size of the queue rarely exceeds 2 tasks, all the states $(\cdot, q_2, \cdot)$, where $q_2 = 3..N$ do not affect the reachability probability too much.

As a more concrete example, consider the CTMDP of Fig. 1a. Here all the states in the center have exit rate 1 and form a long chain. Due to the length of this chain the probability to reach the goal state via these states within time 2 is very small. In fact, the maximum probability to reach the target state within 2 time units in the CTMDP on the left and the one on the right are exactly the same and equal 0.4584. Thus, on this CTMDP, 40% of the state space can be reduced without any effect on the reachability value.

Classical model checking algorithms do not take into account any information about the property and perform exhaustive state-space exploration. Given that only a subset of states is relevant to the reachability value, these algorithms may perform many unnecessary computations.

Our Solution

Throughout this section we work with a CTMDP $\mathcal{M} = (s_{\text{init}}, S, \text{Act}, \mathbf{R}, G)$ and a time bound $T \in \mathbb{R}_{\geqslant 0}$. The proofs of all the results can be found in [ABHK18].

The main contribution of this paper is a simple framework for solving the time-bounded reachability objective in CTMDPs without considering their whole state-space. This framework in presented in Algorithm 1. The algorithm involves the following major steps:

Step 1 A "relevant subset" of the state-space $S' \subseteq S$ is computed (line 6).

Algorithm 1 SubspaceTBR

Input: CTMDP $\mathcal{M} = (s_{\text{init}}, S, \text{Act}, \mathbf{R}, G)$, time bound T, precision ε
Output: $(\ell, u) \in [0,1]^2$ such that $\ell \leqslant \text{val}(T) \leqslant u$ and $u - \ell < \varepsilon$ and $\varepsilon-$ optimal scheduler π for $\text{val}_{\mathcal{M}}(T)$

1: **if** $s_{\text{init}} \in G$ **then return** $(1,1)$, and an arbitrary scheduler $\pi \in \Pi$
2: $\ell = 0, u = 1$
3: $\pi_{\texttt{sim}} = \pi_{\texttt{uniform}}$
4: $S' = \{s_{\text{init}}\}$
5: **while** $u - \ell \geqslant \varepsilon$ **do**
6: $\quad S' = S' \cup \text{getRelevantSubset}(\mathcal{M}, T, \pi_{\texttt{sim}})$
7: $\quad \underline{\mathcal{M}} = \mathsf{lower}(\mathcal{M}, S')$, $\overline{\mathcal{M}} = \mathsf{upper}(\mathcal{M}, S')$
8: $\quad \ell = \text{val}_{\underline{\mathcal{M}}}(T)$, $u = \text{val}_{\overline{\mathcal{M}}}(T)$
9: $\quad \overline{\pi}_{\texttt{opt}} \leftarrow$ optimal scheduler for $\text{val}_{\overline{\mathcal{M}}}(T)$, $\underline{\pi}_{\texttt{opt}} \leftarrow$ optimal scheduler for $\text{val}_{\underline{\mathcal{M}}}(T)$
10: $\quad \pi_{\texttt{sim}} = \text{ChooseScheduler}(\pi_{\texttt{uniform}}, \overline{\pi}_{\texttt{opt}})$ // choose a scheduler for simulations
11: $\forall t \in [0,T], \forall s \in S' : \pi(s,t) = \underline{\pi}_{\texttt{opt}}(s,t)$
12: $\forall t \in [0,T], \forall s \in S \setminus S' : \pi(s,t) \leftarrow$ any $\alpha \in \text{Act}(s)$ // extend optimal scheduler to S
13: **return** $(\ell, u), \pi$

Step 2 Using this subset, CTMDPs $\underline{\mathcal{M}}$ and $\overline{\mathcal{M}}$ are constructed (line 7). We define functions $\mathsf{upper}(\mathcal{M}, S')$ and $\mathsf{lower}(\mathcal{M}, S')$ later in this section.

Step 3 The reachability values of $\underline{\mathcal{M}}$ and $\overline{\mathcal{M}}$ are under- and over-approximations of the reachability value $\text{val}_{\mathcal{M}}(T)$. The values are computed in line 8 along with the optimal schedulers in line 9.

Step 4 Scheduler $\pi_{\texttt{sim}}$, used for obtaining the relevant subset, is selected at line 10.

Step 5 If the two approximations are sufficiently close, i. e. $\text{val}_{\overline{\mathcal{M}}}(T) - \text{val}_{\underline{\mathcal{M}}}(T) < \varepsilon$, $\left[\text{val}_{\underline{\mathcal{M}}}(T), \text{val}_{\overline{\mathcal{M}}}(T)\right]$ is the interval in which the actual reachability value lies. The algorithm is stopped and this interval along with the ε-optimal scheduler are returned. If not, the algorithm repeats from line 6, growing the relevant subset in each iteration.

In the following section, we elucidate these steps and discuss several instantiations and variations of this framework.

3.1 Step 1: Obtaining the Relevant Subset

The main challenge of the approach is to extract a relatively small *representative* set $S' \subseteq S$, for which $\text{val}_{\overline{\mathcal{M}}}(T)$ and $\text{val}_{\underline{\mathcal{M}}}(T)$ are close to the value $\text{val}_{\mathcal{M}}(T)$ of the original model. If this is possible, then instead of computing the probability of reaching goal in $\mathcal{M}$, we can compute the same in $\overline{\mathcal{M}}$ and $\underline{\mathcal{M}}$ to get an ε-width interval in which the actual value is guaranteed to lie. If the sizes of $\overline{\mathcal{M}}$ and $\underline{\mathcal{M}}$ are relatively small, then the computation is generally much faster.

In this work we propose a heuristics for selecting the relevant subset based on simulations. Simulation of continuous-time Markov chains (CTMDPs with singleton set $\text{Act}(s)$ for all states) is a widely used approach that performs very well in many practical cases. It is based on sampling a path of the model according to its probability space. Namely, upon entering a state s the residence time is sampled from the exponential distribution and then the successor state s' is sampled randomly from the distribution $\Delta(s, \alpha, s')$. Here α is the only action available in state s. The process is repeated from state s' until a goal state is reached or the cumulative time over this path exceeds the time-bound.

Algorithm 2 GETRELEVANTSUBSET($\mathcal{M}, T, \pi_{\texttt{sim}}$)

Input: CTMDP $\mathcal{M} = (s_{\text{init}}, S, \text{Act}, \mathbf{R}, G)$, time bound T, a scheduler $\pi_{\texttt{sim}}$
Parameters: $\text{n}_{\text{sim}} \in \mathbb{N}$
Output: $S' \subseteq S$

1: **for** $(i = 0;\ i < \text{n}_{\text{sim}};\ i = i + 1)$ **do**
2: $\quad \rho = s_{\text{init}},\ t = 0$
3: $\quad$ **while** $t < T$ **and** $\rho{\downarrow} \notin G$ **do**
4: $\quad\quad s = \rho{\downarrow}$
5: $\quad\quad$ Sample action α from distribution $\mathcal{D}(\text{Act}(s)) = \pi_{\texttt{sim}}(\rho, 0)$
6: $\quad\quad$ Sample t' from exponential distribution with parameter $\lambda(s, \alpha)$
7: $\quad\quad$ Sample a successor s' of s with distribution $\Delta(s, \alpha, \cdot)$
8: $\quad\quad \rho = \rho \xrightarrow{t'} s',\ t = t + t'$
9: $\quad$ add all states of ρ to S'

However this approach only works for fully stochastic processes, which is not the case for arbitrary CTMDPs due to the presence of multiple available actions. In order to make the process fully stochastic one has to fix a scheduler that decides which actions are to be selected during the run of a CTMDP.

Our heuristic is presented in Algorithm 2. It takes as input the CTMDP, time bound and a scheduler $\pi_{\texttt{sim}}$. The algorithms performs n_{sim} simulations and outputs all the states visited during the execution. Here $\text{n}_{\text{sim}} \in \mathbb{N}$ is a parameter of the algorithm. Each simulation run starts in the initial state. At first an action is sampled from $\mathcal{D}(\text{Act}(s)) = \pi_{\texttt{sim}}(\rho, 0)$ and then the simulation proceeds in the same way as described above for CTMCs by sampling residence times and successor states. Notice that even though time-point 0 is used for the scheduler, this does not affect the correctness of the approach, since it is only used as a heuristic to sample the subspace. In fact, one could instantiate GETRELEVANTSUBSET with an arbitrary heuristic (e. g. from artificial intelligence domain, or one that is more targeted towards a specific model). Correctness of the lower and upper bounds will not be affected by this. However, termination of the algorithm cannot be ensured for any arbitrary heuristic. Indeed, one has to make sure that the bounds will eventually converge to the value.

Example 1. Consider the CTMDP from Figure 3a. Figures 3b and 3c show two possible sampled paths. The path in 3c reaches the target within the given time-bound and the path in 3b times out before reaching the goal state. The relevant subset is thus all the states visited during the two simulations.

3.2 Step 2: Under- and Over-Approximating CTMDP

We will now explain line 7 of Algorithm 1. Here we obtain two CTMDPs, such that the value of $\underline{\mathcal{M}}$ is a guaranteed lower bound, and the value of $\overline{\mathcal{M}}$ is a guaranteed upper bound on the value of $\mathcal{M}$.

Let $S' \subseteq S$ be the subset of states obtained in line 6. We are interested in extracting some information regarding the reachability value of $\mathcal{M}$ from this subset. In order to do this, we consider two cases. (i) A pessimistic case, where all the unexplored states are non-goal states and absorbing (or *sink states*); and (ii) an optimistic case, where all the unexplored states are indeed goals. It is easy to see that the "pessimistic" CTMDP $\underline{\mathcal{M}}$ will have a smaller (or equal) value than the original

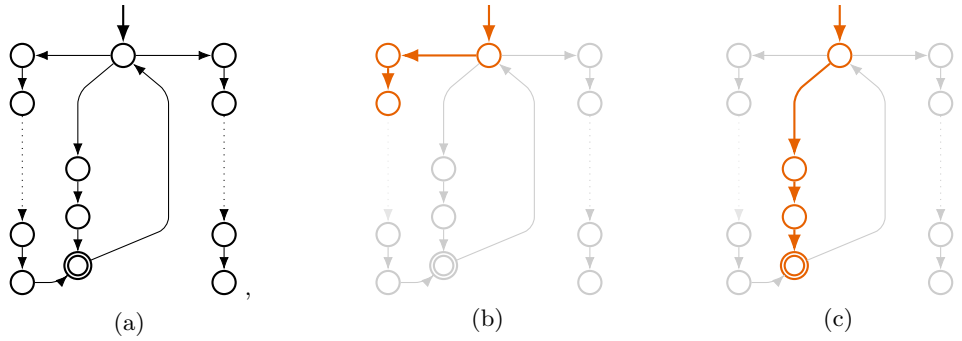


Fig. 3. A simple CTMDP is presented in Fig. (3a) with rates and action labels ignored. Fig. (3b) shows a sampled run which ends on running out of time while exploring the left-most branch. Fig. (3c) shows a simulation which ends on discovering a target state.

CTMDP, which in turn will have a value smaller (or equal) than the "optimistic" CTMDP $\overline{\mathcal{M}}$. Notice that for the reachability value the goal states can also be made absorbing and this will not change the value[4]. Before we define the two CTMDPs formally, we illustrate the construction on an example. Note that the fringe "one-step outside" of the relevant subset is still a part of the considered sub-CTMDPs.

Example 2. Let S' be the state space of the CTMDP from Figure 3a explored in Example 1. Figure 4a depicts the sub-CTMDP obtained by restricting the state space of the original model to S'. Figures 4b and 4c demonstrate how the "pessimistic" and "optimistic" CTMDPs can be obtained. All the states that are not part of S' are made absorbing for the "pessimistic" CTMDP (4b) and are made goal states for the "optimistic" CTMDP (4c).

Formally, we define methods $\mathsf{lower}(\mathcal{M}, S')$ and $\mathsf{upper}(\mathcal{M}, S')$ that return the pessimistic and optimistic CTMDP, respectively. The $\mathsf{lower}(\mathcal{M}, S')$ method returns a CTMDP $\underline{\mathcal{M}} = (s_{\text{init}}, \widetilde{S}, \text{Act}, \widetilde{\mathbf{R}}, G)$, where $\widetilde{S} = S' \cup \text{Succ}(S')$, and $\forall s', s'' \in \widetilde{S}$:

$$\widetilde{\mathbf{R}}[s', \alpha, s''] = \begin{cases} \mathbf{R}[s', \alpha, s''] & \text{if } s' \in S' \\ \overline{\lambda} & \text{if } s' \notin S', s'' = s' \\ 0 & \text{otherwise,} \end{cases}$$

where $\overline{\lambda}$ is the maximum exit rate in $\mathcal{M}$. And the method $\mathsf{upper}(\mathcal{M}, S')$ returns CTMDP $\overline{\mathcal{M}} = (s_{\text{init}}, \widetilde{S}, \text{Act}, \widetilde{\mathbf{R}}, \overline{G})$, where $\overline{G} = G \cup (\widetilde{S} \setminus S')$, and state space $\widetilde{S}$ and the rate matrix $\widetilde{\mathbf{R}}$ are the same as for $\mathsf{lower}(\mathcal{M}, S')$.

Since many states are absorbing now large parts of the state space may become unreachable, namely all the states that are not in $\widetilde{S}$.

Lemma 1. $\text{val}_{\underline{\mathcal{M}}}(T) \leqslant \text{val}_{\mathcal{M}}(T) \leqslant \text{val}_{\overline{\mathcal{M}}}(T)$

[4] This is due to the fact that for the reachability value, only what happens before the first arrival to the goal matters, and everything that happens afterwards is irrelevant.

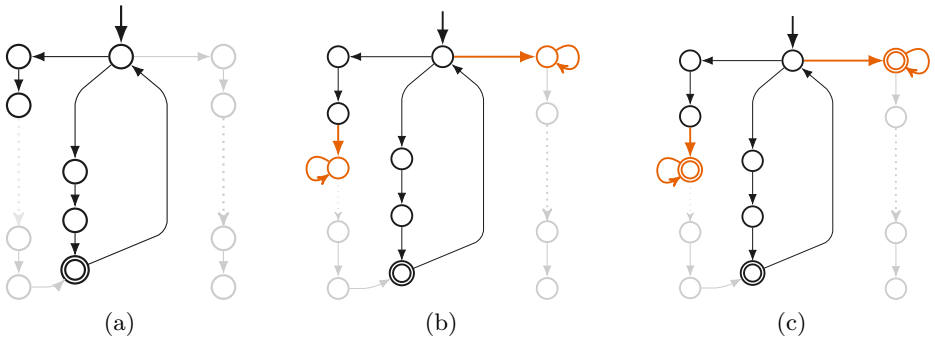


Fig. 4. Fig. 4a depicts the relevant subset obtained at line 6 of Algorithm 1. Fig. 4b and Fig. 4c show the addition of successors (in highlight) of the states at the fringe. In Fig. 4b, the appended states are made absorbing by adding a self-loop of rate $\overline{\lambda}$. Meanwhile in Fig. 4c, the newly added states are made goals.

3.3 Step 3: Computing the Reachability Value

Algorithm 1 requires computing the reachability values for CTMDPs $\underline{\mathcal{M}}$ and $\overline{\mathcal{M}}$ (line 9). This can be done by any algorithm for reachability analysis, e. g. [BHHK15, NZ10, HH13, BS11, FRSZ11, BHHK04] which approximate the value up to an arbitrary precision ε. These algorithms usually also compute the ε-optimal scheduler along with the approximation of the reachability value. In the following we will use interchangeably the notions of the value and its ε-approximation, as well as an optimal scheduler and an ε-optimal scheduler.

Notice that some of the algorithms mentioned above compute optimal reachability value only w. r. t. a subclass of schedulers, rather than the full class Π. In this case the result of Algorithm 1 will be the optimal reachability value with respect to this subclass and not class Π.

3.4 Step 4: The Choice of Scheduler $\pi_{\texttt{sim}}$

At line 10 of Algorithm 1 the scheduler $\pi_{\texttt{sim}}$ is selected that is used in the subsequent iteration for refining the relevant subset of states. We propose two ways of instantiating the function ChooseScheduler($\pi_{\texttt{uniform}}, \overline{\pi}_{\texttt{opt}}$), one with the uniform scheduler $\pi_{\texttt{uniform}}$, and another with the scheduler $\overline{\pi}_{\texttt{opt}}$. Depending on the model, its goal states and the time bound one of the options may deliver smaller relevant subset than another:

Example 3. Consider, the CTMDP in Figure 5a and the time bound 3.0. Assuming that the goal state has not yet been sampled from the right and left chains, action α delivers higher reachability value than action β. For example, if states a_1 to a_2 are sampled from the chain on the left and c_1 to c_2 from the chain on the right, the reachability value of the respective over-approximating CTMDP when choosing action β is 0.1987 and when choosing action α is 0.1911. And this situation persists also when states $b_1 - b_{10}$ are sampled due to high exit rates of the respective transitions. However if state b_{11} is sampled, the reachability value when following α becomes 0.1906. Only at this moment the optimal behaviour is to choose action β. However, when following the uniform scheduler, there is a chance that the whole

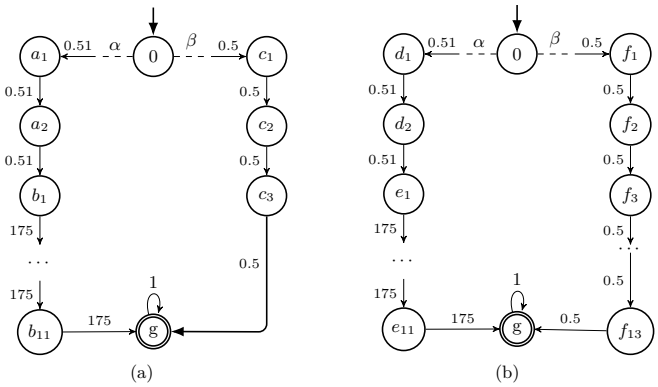


Fig. 5.

chain on the right is explored before any of the states b_i are visited. If the precision $\varepsilon = 0.01$, then at the moment the goal state is reached via the right chain and at least states a_1 to a_2 are sampled on the left, the algorithm has converged. Thus using the uniform scheduler SubspaceTBR may in fact explore fewer states than when using the optimal one.

Naturally, there are situation when following the optimal scheduler is the best one can do. For example, in the CTMDP in Figure 5b it is enough to explore only state f_1 on the right to realise that action β is sub-optimal. From this moment on only action α is chosen for simulations, which is in fact the best way to proceed. At the moment the goal state is reached the algorithm has converged for precision 0.01.

One of the main advantages of the uniform scheduler is that it does not require too much memory and is simple to implement. Moreover, since some algorithms to compute time-bounded reachability probability do not provide an optimal scheduler in the classical way as defined in Section 2 ([BHHK15]), the use of $\pi_{\texttt{uniform}}$ may be the only option. In spite of its simplicity, in many cases this scheduler generates very succinct state spaces, as we will show in Section 4.

Using the uniform scheduler is beneficial in those cases when, for example, different actions of the same state have exit rates that differ drastically, e. g. by an order of magnitude. If the goal state is reachable via actions with high rates, choosing an action with low rate leads to higher residence times (due to properties of the exponential distribution) and therefore fewer states will be reachable within the time bound, compared to choosing an action with a high exit rate. In this case using the uniform scheduler may lead to larger sub-space, compared to using the optimal scheduler. However, the experiments show this difference is typically negligible.

The drawback of the uniform scheduler is that the probability of it choosing each action is positive. Thus it will choose also those actions that are clearly suboptimal and could be omitted during the simulations. The uniform scheduler $\pi_{\texttt{uniform}}$ does not take this information into account while the scheduler $\overline{\pi}_{\texttt{opt}}$ does. The latter is optimal on the sub-CTMDP obtained during the previous iterations. This scheduler will thus pick only those actions that look most promising to be optimal. Using this scheduler may induce smaller sampled state space than the one generated by $\pi_{\texttt{uniform}}$, as we also show in Section 4.

Notice that it is possible to alternate between using $\pi_{\texttt{uniform}}$ and $\overline{\pi}_{\texttt{opt}}$ at different iterations of Algorithm 1, for instance, when $\overline{\pi}_{\texttt{opt}}$ is costly to obtain or simulate. However, in our experiments, we always choose either one of the two, with the exception for the first iteration when only the uniform scheduler is available.

3.5 Step 5: Termination and Optimal Schedulers

The algorithm runs as long as the values of $\underline{\mathcal{M}}$ and $\overline{\mathcal{M}}$, as computed in Step 3 are not sufficiently close. It terminates when the difference becomes less than ε. The scheduler $\underline{\pi}_{\texttt{opt}}$ obtained in line 9 is ε-optimal for $\underline{\mathcal{M}}$ since it is obtained by running a standard TBR algorithm on $\underline{\mathcal{M}}$. From this scheduler one can obtain ε-optimal scheduler π for $\mathcal{M}$ itself by choosing the same actions as $\underline{\pi}_{\texttt{opt}}$ on the relevant subset of states (S' in Algorithm 1) and any arbitrary action on other states.

Lemma 2. *Scheduler π computed by Algorithm 1 is ε-optimal.*

Theorem 1. *Algorithm 1 converges almost surely.*

On any CTMDP, if $\pi_{\texttt{sim}} = \pi_{\texttt{uniform}}$, Algorithm 1 will, in the worst case, eventually explore the whole CTMDP. In such a situation, $\underline{\mathcal{M}}$ and $\overline{\mathcal{M}}$ will be the same as $\mathcal{M}$. The algorithm would then terminate since the condition on line 5 would be falsified. If $\pi_{\texttt{sim}} = \overline{\pi}_{\texttt{opt}}$, the system is continuously driven to the fringe as long as the condition on line 5 holds. This is because all unexplored states act as goal states in the upper-bound model. Such a scheduler will eventually explore the state-space reachable by the optimal scheduler on the original model and leave out those parts that are only reachable with suboptimal decisions.

4 Experiments

The framework described in Section 3 was evaluated against 5 different benchmarks available in the MAPA language [TKvdPS12]:

Fault Tolerant Work Station Cluster (`ftwc-n`) [HHK00]: models two networks of n workstations each. Each network is interconnected by a switch. The switches communicate via a backbone. All the components may fail and can be repaired only one at a time. The system starts in a fully functioning state and a state is goal if in both networks either all the workstations or the switch are broken.

Google File System (`gfs-n`) [HCH$^+$02, GGL03]: in this benchmark files are split into chunks, each maintained by one of n chunk servers. We fix the number of chunks a server may store to 5000 and the total number of chunks to 10000. The GFS starts in the state where for one of the chunks no replica is stored and the target is to have at least 3 copies of the chunk available.

Polling System (`ps-j-k-g`): We consider the variation of the polling system case [GHH$^+$13] [TvdPS13], that consists of j stations and one server. Incoming requests of j types are buffered in queues of size k each, until they are processed by the server and delivered to their station. The system starts in a state with all the queues being nearly full. We consider 2 goal conditions: (i) all the queues are empty (g=`all`) and (ii) one of the queues is empty (g=`one`).

Table 1. An overview of the experimental results along with the state-space sizes. Runtime (in seconds) for the various algorithms are presented. '-' indicates a timeout (1800 secs). $\mathtt{U}_{\mathsf{uni}}$, $\mathtt{A}_{\mathsf{uni}}$ and $\mathtt{A}_{\mathsf{opt}}$ perform quite well on `erlang`, `gfs` and `ftwc` while only $\mathtt{A}_{\mathsf{opt}}$ is better than `U` and `A` on the `ps-one` family of models. `ps`-4-8-`all` and `sjs` are hard instances for both $\pi_{\mathtt{uniform}}$ and $\overline{\pi}_{\mathtt{opt}}$. `D` times out on all benchmarks except on `sjs` due to its small state-space.

Benchmark	States	U	$\mathtt{U}_{\mathsf{uni}}$	A	$\mathtt{A}_{\mathsf{uni}}$	$\mathtt{A}_{\mathsf{opt}}$	D	$\mathtt{D}_{\mathsf{uni}}$
`erlang`-10^6-10	1,000k	71	**1**	4	**1**	**1**	-	299
`gfs`-120	1,479k	-	**2**	-	**2**	**2**	-	-
`ftwc`-128	597k	251	**10**	114	**11**	**15**	-	-
`ps`-4-24-`one`	7,562k	507	-	171	-	**105**	-	-
`ps`-4-8-`all`	119k	1,475	-	826	-	-	-	-
`sjs`-2-9	18k	6	99	2	139	-	1,199	-

Erlang Stages (`erlang-k-r`): this is a synthetic model with known characteristics [ZN10]. It has two different paths to reach the goal state: a fast but risky path or a slow but sure path. The slow path is an Erlang chain of length k and rate r.

Stochastic Job Scheduling (`sjs-m-j`) [BDF81]: models a multiprocessor architecture running a sequence of independent jobs. It consists of m identical processors and j jobs. As goal we define the states with all jobs completed;

Our algorithm is implemented as an extension to `PRISM` [KNP11] and we use `IMCA` [GHKN12] in order to solve the sub-CTMDPs ($\underline{\mathcal{M}}$ and $\overline{\mathcal{M}}$). We would like to remark, however, that the performance of our algorithm can be improved by using a better toolchain than our `PRISM-IMCA` setup (see [ABHK18, Appendix A.2]).

In order to instantiate our framework, we need to describe how we perform Steps 1 and 3 (Section 3). Recall from Section 3.1 that we proposed two different schedulers to be used as the simulating scheduler $\pi_{\mathtt{sim}}$: the uniform scheduler $\pi_{\mathtt{uniform}}$ and the optimal scheduler $\overline{\pi}_{\mathtt{opt}}$ obtained by solving $\overline{\mathcal{M}}$.

For Step 3, we select three algorithms for time-bounded reachability analysis: the first discretisation-based algorithm [NZ10] (`D`), and the two most competitive algorithms according to the comparison performed in [BHHK15], namely the adaptive version of discretization [BS11] (`A`) and the uniformisation-based [BHHK15] (`U`). SubspaceTBR instantiated with these algorithms and with $\pi_{\mathtt{sim}} = \pi_{\mathtt{uniform}}$ is referred to with $\mathtt{D}_{\mathsf{uni}}$, $\mathtt{A}_{\mathsf{uni}}$ and $\mathtt{U}_{\mathsf{uni}}$ respectively. For $\pi_{\mathtt{sim}} = \overline{\pi}_{\mathtt{opt}}$, the instantiations are referred to as $\mathtt{D}_{\mathsf{opt}}$, $\mathtt{A}_{\mathsf{opt}}$ and $\mathtt{U}_{\mathsf{opt}}$. Since `U` does not provide the scheduler in a classical form as defined in Section 2, we omit $\mathtt{U}_{\mathsf{opt}}$. We also omit experiments on $\mathtt{D}_{\mathsf{opt}}$ as our experience with `D` and $\mathtt{D}_{\mathsf{uni}}$ suggested that $\mathtt{D}_{\mathsf{opt}}$ would also run out of time on most experiments.

We compare the performance of the instantiated algorithms with their originals, implemented in `IMCA`. We set the precision parameter for SubspaceTBR and the original algorithms in `IMCA` to 0.01. Indicators such as the median model checking time (excluding the time taken to load the model into memory) and explored state-space are measured.

Tables 1 and 2 summarize the main results of our experiments. Table 1 reports the runtime of the algorithms on several benchmarks, while Table 2 reports on the state-space complexity. Here the last column refers to the smallest relevant subset of

Table 2. For each benchmark, we report (i) the size of the state-space; (ii) total states explored by our instantiations of SUBSPACETBR; (iii) size of the final over-approximating sub-CTMDP $\overline{\mathcal{M}}$; and (iv) size of the relevant subset returned by the greedy search of Section 4.1. We use `ps-4-4-one` and `sjs-2-7` instead of larger models in their respective families as running the greedy search is a highly computation-intensive task.

Benchmark	States	Explored		Size of last $\overline{\mathcal{M}}$	Post greedy reduction
		by $\pi_{\texttt{sim}}$	%		
`erlang`-10^6-10	1,000k	559	0.06	561	496
`gfs`-120	1,479k	105	0.01	200	85
`ftwc`-128	597k	296	0.05	858	253
`sjs`-2-7	2k	2,537	93.86	2,704	1,543
`ps`-4-4-`one`	10k	697	6.63	2,040	696
`ps`-4-8-`all`	119k	-	-	-	-
`ps`-4-24-`one`	7,562k	23,309	0.31	-	-

$\overline{\mathcal{M}}$ that we can obtain with reasonable effort. This subset is computed by running the greedy algorithm described in Section 4.1 on $\overline{\mathcal{M}}$. It attempts to reduce more states of the explored subset without sacrificing the precision too much. We run the greedy algorithm with a precision of $\varepsilon/10$, where ε is the precision used in SUBSPACETBR.

We recall that our framework is targeted towards models which contain a small subset of valuable states. We can categorize the models into three classes:

Easy with Uniform Scheduler ($\pi_{\texttt{sim}} = \pi_{\texttt{uniform}}$). Surprisingly enough, the uniform scheduler performs well on many instances, for example `erlang`, `gfs` and `ftwc`. For `erlang` and `gfs`, it was sufficient to explore a few hundred states no matter how the parameter which increased the state-space was changed (see description of the models above). Here the running time of the instantiations of our framework outperformed the original algorithms due to the fact that less than 1% of the state-space is sufficient to approximate the reachability value up to precision 0.01.

Easy with Optimal Scheduler ($\pi_{\texttt{sim}} = \overline{\pi}_{\texttt{opt}}$). Predictably, there are cases in which uniform scheduler does not provide good results. For example consider the case of `ps`-4-24-`one`. Here the goal condition requires that one of the queues be empty. An action in this benchmark determines the queue from which the task to be processed is picked. Choosing tasks uniformly from different queues, not surprisingly, leads to larger explored state spaces and longer runtimes. Notice that all the instantiations that use uniform scheduler run out of time on this instance. On the other hand, targeted exploration with the most promising scheduler (column $\texttt{A}_{\texttt{opt}}$) performs even better than the original algorithm `A`, finishing within 105 s compared to 171 s and exploring only 0.31% of the state space.

Hard Instances. Naturally there are instances where it is not possible to find a small sub-CTMDP that preserves the properties of interest. For example in `ps`-4-8-`all`, the system is started with all queues being nearly full and the property queried requires all of the queues in the polling system to be empty.

As discussed in the beginning of Section 3, most of the states of the model have to be explored in order to reach the goal state. In this model there is simply no small sub-CTMDP that preserves the reachability probabilities. As expected, all instantiations timed out and nearly all the states had to be explored. The situation is similar with `sjs`. We identified (using the greedy algorithm in Section 4.1) that on some small instances of this model, only 30% to 40% of the state-space can be sacrificed.

Explored State Space and Running Time. In general, as we have mentioned in Section 3, the problem is heavily dependent not only on the structure of the model, but also on the specified time-bound and the goal set. Increasing the time-bound for `erlang`, for example, leads to higher probability to explore fully the states of the Erlang chain. This in turn affects the optimal scheduler and for some time-bounds no small sub-CTMDP preserving the value exists.

Naturally, whenever the algorithm explored only a small fraction of the state space, the running time was usually also smaller than the running time of the respective original algorithm. The performance of our framework is heavily dependent on the parameter n_{sim}. This is due to the fact that computation of the reachability value is an expensive operation when performed many times even on small models. Usually in our experiments the amount of simulations was in the order of several thousands. For more details please refer to [ABHK18, Appendix A.2].

4.1 Smallest sub-CTMDP

In this section, we provide an argument that in the cases where our techniques do not perform well, the reason is not a poor choice of the relevant subsets, but rather that in such cases there are no small subsets which can be removed, at least not such that can be easily obtained. An ideal brute-force method to ascertain this would be to enumerate all subsets of the state space, make the states of the subset absorbing ($\underline{\mathcal{M}}$) or goal ($\overline{\mathcal{M}}$) and then to check whether the difference in values of $\underline{\mathcal{M}}$ and $\overline{\mathcal{M}}$ is ε-close only for small subsets. Unfortunately, this is computationally infeasible. As an alternative, we now suggest a greedy algorithm which we use to search for the largest subset of states one could remove in reasonable time.

The idea is to systematically pick states and observe their effect on the value when they are made absorbing ($\underline{\mathcal{M}}(s)$) or goal ($\overline{\mathcal{M}}(s)$). If a state does not influence the value of the original CTMDP too much, then $\delta(s) = \text{val}_{\overline{\mathcal{M}}(s)}(T) - \text{val}_{\underline{\mathcal{M}}(s)}(T)$ would be small. We first sort all the states in ascending order according to the value $\delta(s)$. And then iteratively build $\underline{\mathcal{M}}$ and $\overline{\mathcal{M}}$ by greedily picking states in this order and making them absorbing (for $\underline{\mathcal{M}}$) and goal (for $\overline{\mathcal{M}}$). The process is repeated until $\text{val}_{\overline{\mathcal{M}}}(T) - \text{val}_{\underline{\mathcal{M}}}(T)$ exceeds ε.

The results of running this algorithm is presented in the right-most column of Table 2. The comparison of the last two columns of the table shows that the portion of the state space our heuristic explored is of the same order of magnitude as what can be obtained with high computational effort. Consequently, this suggests that the surprising choice of the simple uniform scheduler is not poor, but typically indeed achieves the desired degree of reduction.

5 Conclusion

We have introduced a framework for time-bounded reachability analysis of CTMDPs. This framework allows us to run arbitrary algorithms from the literature on a subspace of the original system and thus obtain the result faster, while not compromising its precision beyond a given ε. The subspace is iteratively identified using simulations. In contrast to the standard algorithms, the amount of computation needed reflects not only the model, but also the property to be checked.

The experimental results have revealed that the models often have a small subset which is sufficient for the analysis, and thus our framework speeds up all three considered algorithms.For the exploration, already the uninformed uniform scheduler proves efficient in many settings. However, the more informed scheduler, fed back from the analysis tools, may provide yet better results. In cases where our technique explores the whole state space, our conjecture, confirmed by the preliminary results using the greedy algorithm, is that these models actually do not posses any small relevant subset of states and cannot be exploited by this approach.

This work is agnostic of the structure of the models. Given that states are typically given by a valuation of variables, the corresponding structure could be further utilized in the search for the small relevant subset. A step in this direction could follow the ideas of [PBU13], where discrete-time Markov chains are simulated, the simulations used to infer invariants for the visited states, and then the invariants used to identify a subspace of the original system, which is finally analyzed. An extension of this approach to a non-deterministic and continuous setting could speed up the subspace-identification part of our approach and thus decrease our overhead. Another way to speed up this process is to quickly obtain good schedulers (with no guarantees), e.g. [BBB$^+$17], use them to identify the subspace faster and only then apply a guaranteed algorithm.

References

ABHK18. P. Ashok, Y. Butkova, H. Hermanns, and J. Křetínský. Continuous-Time Markov Decisions based on Partial Exploration. *ArXiv e-prints*, 2018.

ACD$^+$17. P. Ashok, K. Chatterjee, P. Daca, J. Kretínský, and T. Meggendorfer. Value iteration for long-run average reward in markov decision processes. In *CAV*, 2017.

ASSB96. A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton. Verifying continuous time markov chains. In *CAV*, 1996.

BBB$^+$17. E. Bartocci, L. Bortolussi, T. Brázdil, D. Milios, and G. Sanguinetti. Policy learning in continuous-time markov decision processes using gaussian processes. *Perform. Eval.*, 116:84–100, 2017.

BCC$^+$14. T. Brázdil, K. Chatterjee, M. Chmelik, V. Forejt, J. Kretínský, M. Z. Kwiatkowska, D. Parker, and M. Ujma. Verification of Markov decision processes using learning algorithms. In *ATVA*, 2014.

BDF81. J. L. Bruno, P. J. Downey, and G. N. Frederickson. Sequencing tasks with exponential service times to minimize the expected flow time or makespan. *J. ACM*, 28(1):100–113, 1981.

Ber95. D. P. Bertsekas. *Dynamic Programming and Optimal Control. Vol. II*. Athena Scientific, 1995.

BFK$^+$09. T. Brázdil, V. Forejt, J. Krčál, J. Křetínský, and A. Kučera. Continuous-time stochastic games with time-bounded reachability. In *FSTTCS*, 2009.

BHHK04. C. Baier, B. R. Haverkort, H. Hermanns, and J. Katoen. Efficient computation of time-bounded reachability probabilities in uniform continuous-time markov decision processes. In *TACAS*, 2004.

BHHK15. Y. Butkova, H. Hatefi, H. Hermanns, and J. Krcál. Optimal continuous time markov decisions. In *ATVA*, 2015.

BS11. P. Buchholz and I. Schulz. Numerical analysis of continuous time Markov decision processes over finite horizons. *Computers & OR*, 38(3):651–659, 2011.

EHKZ13. C. Eisentraut, H. Hermanns, J. Katoen, and L. Zhang. A semantics for every GSPN. In *Petri Nets*, 2013.

Fei04. E. A. Feinberg. Continuous time discounted jump markov decision processes: A discrete-event approach. *Mathematics of Operations Research*, 29(3):492–524, 2004.

FRSZ11. J. Fearnley, M. Rabe, S. Schewe, and L. Zhang. Efficient Approximation of Optimal Control for Continuous-Time Markov Games. In *FSTTCS*, 2011.

GGL03. S. Ghemawat, H. Gobioff, and S. Leung. The google file system. In *SOSP*, 2003.

GHH+13. D. Guck, H. Hatefi, H. Hermanns, J. Katoen, and M. Timmer. Modelling, reduction and analysis of markov automata. In *QEST*, 2013.

GHKN12. D. Guck, T. Han, J. Katoen, and M. R. Neuhäußer. Quantitative timed analysis of interactive markov chains. In *NFM*, 2012.

HCH+02. B. R. Haverkort, L. Cloth, H. Hermanns, J. Katoen, and C. Baier. Model checking performability properties. In *DSN*, 2002.

HH13. H. Hatefi and H. Hermanns. Improving time bounded reachability computations in interactive Markov chains. In *FSEN*, 2013.

HHK00. B. R. Haverkort, H. Hermanns, and J. Katoen. On the use of model checking techniques for dependability evaluation. In *SRDS'00*, 2000.

KNP11. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, pages 585–591, 2011.

Lef81. C. Lefèvre. Optimal control of a birth and death epidemic process. *Operations Research*, 29(5):971–982, 1981.

MLG05. H. B. McMahan, M. Likhachev, and G. J. Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In , *ICML*, 2005.

Neu10. M. R. Neuhäußer. *Model checking nondeterministic and randomly timed systems*. PhD thesis, RWTH Aachen University, 2010.

NZ10. M. R. Neuhäußer and L. Zhang. Time-bounded reachability probabilities in continuous-time Markov decision processes. In *QEST*, 2010.

PBU13. E. Pavese, V. A. Braberman, and S. Uchitel. Automated reliability estimation over partial systematic explorations. In *ICSE*, pages 602–611, 2013.

QQP01. Q. Qiu, Q. Qu, and M. Pedram. Stochastic modeling of a power-managed system-construction and optimization. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 20(10):1200–1217, 2001.

Sen99. L. I. Sennott. *Stochastic Dynamic Programming and the Control of Queueing Systems*. Wiley-Interscience, New York, NY, USA, 1999.

TKvdPS12. M. Timmer, J.-P. Katoen, J. van de Pol, and M. I. A. Stoelinga. Efficient modelling and generation of markov automata. In *CONCUR*, 2012.

TvdPS13. M. Timmer, J. van de Pol, and M. Stoelinga. Confluence reduction for markov automata. In *FORMATS*, 2013.

ZN10. L. Zhang and M. R. Neuhäußer. Model checking interactive markov chains. In *TACAS*, 2010.