11-2014

SCC-based improved reachability analysis for Markov decision processes

Lin GUI

Jun SUN
Singapore Management University, junsun@smu.edu.sg

Songzheng SONG

Yang LIU

Jin Song DONG

SCC-Based Improved Reachability Analysis for Markov Decision Processes*

Lin Gui[1], Jun Sun[2], Songzheng Song[3], Yang Liu[3], and Jin Song Dong[1]

[1] National University of Singapore, Singapore
[2] Singapore University of Technology and Design, Singapore
[3] Nanyang Technological University, Singapore

Abstract. Markov decision processes (MDPs) are extensively used to model systems with both probabilistic and nondeterministic behavior. The problem of calculating the probability of reaching certain system states (hereafter reachability analysis) is central to the MDP-based system analysis. It is known that existing approaches on reachability analysis for MDPs are often inefficient when a given MDP contains a large number of states and loops, especially with the existence of multiple probability distributions. In this work, we propose a method to eliminate strongly connected components (SCCs) in an MDP using a divide-and-conquer algorithm, and actively remove redundant probability distributions in the MDP based on the convex property. With the removal of loops and parts of probability distributions, the probabilistic reachability analysis can be accelerated, as evidenced by our experiment results.

1 Introduction

Markov decision processes (MDPs) are extensively used to model a system with both non-determinism and probabilistic behavior. One fundamental task in probabilistic model checking is to decide the probability of reaching a set of target states in an MDP. We refer to this as the *reachability analysis* problem. A discrete time Markov chain (DTMC) can be considered as a special form of MDPs with unique reachability probability since it has only one probability distribution at each state. For general MDPs, there are multiple probability distributions in a state, and thus the practical interests for the reachability analysis focus on the maximum and minimum reachability probabilities.

Given an MDP and a set of target states, a variable can be created for each state to present the probability of that state reaching the target states. There are two main methods to calculate or approximate the values of these variables [4]. One method encodes the probabilistic reachability problem into a linear optimization problem where each probability distribution is encoded into an inequality. Thus, the goal is to maximize or minimize the sum of the variables. It should be noted that the state-of-the-art linear solvers are limited to small systems. However, a practical MDP model is often resulted from parallel composition of several MDPs/DTMCs, which would have an even larger number of states.

* This project is partially supported by project IDD11100102A/IDG31100105A from SUTD.

S. Merz and J. Pang (Eds.): ICFEM 2014, LNCS 8829, pp. 171–186, 2014.

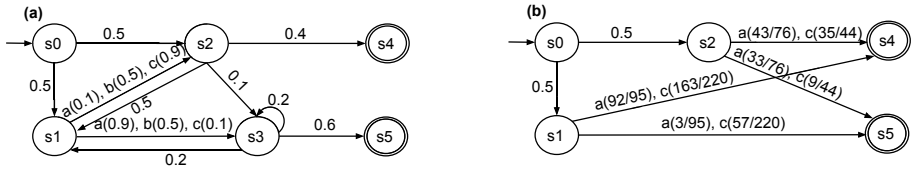

Fig. 1. Running examples of (a) an MDP and (b) an acyclic MDP

The other method is based on value iteration by finding a better approximation iteratively until the result satisfies a certain stopping criterion, and performs generally better in system with a large number of states [4]. The approximation of the variable of a state needs to be updated whenever any of its successive states are changed. When there are loops in an MDP, this approach tends to require many iterations before converging to a value, and thus lead to *slow convergence*. Fig. 1(a) shows an example of a simple MDP with loops among states s_1, s_2 and s_3. Suppose the task is to calculate the probability of reaching state s_4 from state s_0. If the approximation in s_2 is updated during the k^{th} iteration, the approximation in s_1 will be updated during the $(k+1)^{th}$ iteration as s_2 is successive to s_1. The update of s_1 will trigger s_3 to update its value subsequently, which requires s_2 to be updated again. This iteration can only be stopped by enforcing a stopping criterion, thus one major issue associated with such an approach is that the difference between the approximated and 'actual' probabilities remains unknown even after the iteration is stopped [9]. On the other hand, in an acyclic MDP in Fig. 1(b), each state will be visited only a few rounds for backward calculation without iterations. In this case, the exact maximum and minimum probabilities can be calculated without the necessity of approximation. Therefore, we are motivated to improve reachability analysis by removing loops in an MDP.

Some foundation has been established by recent works on the elimination of strongly connected components (SCCs) in DTMC [3,2,13]. To remove the loops, SCCs are first identified, and the transition probabilities from every input to output states of each SCC are calculated. The loops can then be removed by connecting the inputs to the outputs with the computed probability transitions (i.e., *abstraction* of SCC). After all the SCCs are abstracted, the whole model becomes acyclic. With such an acyclic set of states, value iteration can be used to calculate the probability from initial states to the target states. Although this approach works for DTMCs, eliminating loops in an MDP is particularly challenging due to the existence of multiple probability distributions. In an MDP, the number of memoryless schedulers increases exponentially with the number of the states that have multiple probability distributions. During the abstraction of a group of states, a probability distribution must be calculated under different memoryless scheduler in the group. As a result, the total number of probability distributions can increase exponentially after abstraction. Therefore, directly applying the existing approaches [3,2,13] to MDPs is often infeasible.

To overcome this challenge, we propose a divide-and-conquer algorithm to remove loops in an MDP. For each SCC in the MDP, we first construct *partitions*, i.e., each state in the SCC forms a partition. By solving sets of linear equations, new probability distributions can be calculated from each partition to replace the loops without varying the overall reachability probabilities. With the new equivalent probability distributions, the

new partition will be free of loops, and have the same reachability probabilities with the original model. We repeatedly merge a few partitions into one partition, and eliminate loops in this new partition by performing the above abstraction until only one partition is left in the SCC. After the reduction for all the SCCs, the remaining acyclic MDP can be solved efficiently via the value iteration approach. After this reduction, the maximum and minimum reachability probabilities of the reduced MDP remain unchanged as compared with those of the original MDP. As introduced earlier, reducing states in SCCs of an MDP may result in exponentially many probability distributions, and our algorithm is thus designed to eliminate redundant or infeasible probability distributions on-the-fly to achieve better performance. The underlying observation is that, a probability distribution will not affect the maximum or the minimum reachability probability, if it is not a vertex of the convex hull of a set of probability distributions. Our contributions are three-fold and are summarized as follows.

1. To tackle the problem of slow convergence, we propose a divide-and-conquer approach to eliminate SCCs in an MDP. Our approach works on the partitions and can effectively avoid generating large number of schedulers.
2. To reduce the cost of loop eliminations within each partition of an MDP, we remove redundant nondeterministic choices/probability distributions based on convex hull theory.
3. The new approach has been implemented in our model checking framework PAT [15], and two practical case studies (i.e., software reliability assessment and tennis tournament prediction) have been conducted to show its effectiveness.

2 Preliminaries

2.1 Markov Decision Processes

Markov Decision Processes. (MDPs) are popular choices to model a system exhibiting both probabilistic and nondeterministic behavior [4]. Given a set of states S, a probability distribution (PD) is a function $u : S \to [0, 1]$ such that $\Sigma_{s \in S} u(s) = 1$. The PD can also be expressed in the vector form as $\mathbf{u}$, and $Distr(S)$ denotes the set of all discrete probability distributions over S. The formal definition of MDP is introduced as follows.

Definition 1 (Markov Decision Process). *A Markov decision process is a tuple $\mathcal{M} = (S, init, Act, Pr)$ where S is a set of states; $init \in S$ is the initial state; Act is an alphabet; and $Pr : S \times Act \to Distr(S)$ is a labeled transition relation.* □

Without loss of generality, in this work, we assume that MDP has an unique initial state and is always deadlock free. It is known that we can add a self-looping transition with a probability of 1 to a deadlock state without affecting the calculation result [4]. A state without any outgoing transitions to other states is called an $absorbing$ state, which has only a self-loop with a probability of 1. An example of MDP is shown in Fig. 1(a), where states s_4 and s_5 are both absorbing states, denoted by circles with double lines, and state s_0 is the initial state. Given a state s, we denote the set of probability distributions of s as $\mathbf{U}_s$, s.t., $\mathbf{U}_s = \{Pr(s, a) | a \in Act\}$. An infinite or a finite path in $\mathcal{M}$ is defined as a sequence of states $\pi = s_0, s_1, \cdots$ or $\pi = s_0, s_1, \cdots, s_n$, respectively, such

that $\forall i \geq 0$ (for finite paths, $i \in [0, n-1]$), $\exists a \in Act, Pr(s_i, a)(s_{i+1}) > 0$. An MDP is nondeterministic if any state has more than one probability distribution. As a special MDP, a discrete time Markov chain (DTMC) has only one event (and one probability distribution) at each state, and thus is deterministic.

Similar to [3,2,13], in an MDP $\mathcal{M} = (S, init, Act, Pr)$, we define inputs and outputs of a group of states $\mathcal{K} \subseteq \mathcal{S}$ as follows.

$$Inp(\mathcal{K}) = \{s' \in \mathcal{K} \mid \exists s \in S \backslash \mathcal{K}, \exists a \in Act \cdot Pr(s, a)(s') > 0\},$$
$$Out(\mathcal{K}) = \{s' \in S \backslash \mathcal{K} \mid \exists s \in \mathcal{K}, \exists a \in Act \cdot Pr(s, a)(s') > 0\}.$$

Here, the set of input states of $\mathcal{K}$, $Inp(\mathcal{K})$, contains the states in $\mathcal{K}$ that have incoming transitions from states outside $\mathcal{K}$; and the set of output states of $\mathcal{K}$, $Out(\mathcal{K})$, contains all the states outside $\mathcal{K}$ that have direct incoming transitions from states in $\mathcal{K}$. In addition, without loss of generality, if a group contains the initial state $init$, we include $init$ to its input states (with an imaginary transition leading to $init$ from outside). Furthermore, given a set $\mathcal{K}$, if a state is not an input state, we call it as an inner state. We can eliminate all the inner states by calculating the direct transition probabilities from $Inp(\mathcal{K})$ to $Out(\mathcal{K})$. This process is called *abstraction*. It eliminates all loops in $\mathcal{K}$, and meanwhile, *preserves* the maximum and minimum reachability probabilities from inputs to the outputs of $\mathcal{K}$. There are known algorithms in [2,13] to perform the abstraction. However, they are only applicable to DTMCs. In this work, we extend the abstraction to MDPs.

Schedulers. A scheduler is used to resolve the non-determinism in each state. Intuitively, given a state s, an action is first selected by a scheduler. Once an action is selected, the respective PD is also determined; and then one of the successive states is reached according to the probability distribution. Formally, a *memoryless scheduler* for an MDP $\mathcal{M}$ is a function $\sigma : S \to Act$. At each state, a memoryless scheduler always selects the same action in a given state. This choice is independent of the path that leads to the current state. In the following, unless otherwise specified, the terms 'schedulers' and 'memoryless schedulers' are used interchangeably. An induced MDP, $\mathcal{M}_\sigma$, is a DTMC defined by an MDP $\mathcal{M}$ and a scheduler σ. A non-memoryless scheduler is the scheduler that can select different action in a given state according to the execution history.

Strongly Connected Components. A set of states $C \subseteq S$ is called *strongly connected* in $\mathcal{M}$ iff $\forall s, s' \in C$, there exists a finite path $\pi = \langle s_0, s_1, \cdots, s_n \rangle$ satisfying $s_0 = s \wedge s_n = s' \wedge \forall i \in [0, n], s_i \in C$. *Strongly connected components* (SCCs) are the maximal sets of the strongly connected states. All SCCs can be automatically identified by Tarjan's approach [16], with a complexity of $\mathcal{O}(n+l)$, where n and l are the numbers of states and transitions, respectively. In Fig. 1(a), $\{s_0\}$, $\{s_4\}$, $\{s_5\}$ and $\{s_1, s_2, s_3\}$ are the SCCs in the model. We define SCCs as trivial if they do not have any outgoing transitions (e.g., $\{s_4\}$, $\{s_5\}$) or are not involved in loops (e.g., $\{s_0\}$, an SCC of one single state without any loop). As a result, $\{s_1, s_2, s_3\}$ is the only nontrivial SCC in Fig. 1(a). An MDP is considered *acyclic* if it contains only trivial SCCs. An example of an acyclic MDP is shown in Fig. 1(b). Note that an acyclic MDP may still have absorbing states, but it does not affect the computation of reachability probabilities.

2.2 Probability Reachability Analysis in MDPs

One fundamental question in quantitative analysis of MDPs is to compute the probability of reaching target states G from the initial state. Noted that with different schedulers, the result may be different. The measurement of interest is thus the maximum and minimum reachability probabilities. The maximum probability of reaching any state in G is denoted as $P^{max}(\mathcal{M} \models \Diamond G)$, which is defined as: $P^{max}(\mathcal{M} \models \Diamond G) = \sup_\sigma P(\mathcal{M}_\sigma \models \Diamond G)$. Similarly, the minimum is defined as: $P^{min}(\mathcal{M} \models \Diamond G) = \inf_\sigma P(\mathcal{M}_\sigma \models \Diamond G)$, which yields the lower bound of the probability of reaching G. The supremum/infimum ranges over all and potentially infinitely many schedulers. Rather than considering all schedulers, it suffices to consider only memoryless schedulers, in order to obtain maximum and minimum reachability probabilities [4].

In the following, with the MDP in Fig. 1(a), we demonstrate how to numerically calculate the maximum probability of reaching any state in G from the initial state. The minimum probability can be obtained similarly. Here, state s_0 is the initial state, and G contains a single target state s_4. Let V be a vector such that, given a state s, $V(s)$ is the maximum probability of reaching G from a state s. For instance, $V(s_0)$ is the maximum probability of reaching G from the initial state. First of all, $V(s) = 1$, for all $s \in G$. Using backward reachability analysis, we can identify the set of states $X = \{s_0, s_1, s_2, s_3, s_4\}$ such that G is reachable from any state in X; and a set of states $Y = \{s_5\}$ from where G is unreachable, i.e., $V(s) = 0$ for $\forall s \in Y$. Therefore, $V(s_4) = 1$ and $V(s_5) = 0$. There are two main approaches on calculating the reachability probabilities for states $X \setminus G$, i.e., $\{s_0, s_1, s_2, s_3\}$.

Linear Programming. The method encodes each probability distribution (PD) for a state in $X \setminus G$ into a linear inequality. This is defined as

$$V(s) \geqslant \sum\nolimits_{t \in S} P(s, \alpha)(t) \cdot V(t), \quad \text{for } s \in X \setminus G \tag{1}$$

with an additional constraint $V(s) \in [0, 1]$, and the goal is to minimize the sum of V. Taking state s_2 for example, there is a unique PD $\{0.5 \mapsto s_1, 0.1 \mapsto s_3, 0.4 \mapsto s_4\}$, which can be encoded as: $V(s_2) \geqslant 0.5V(s_1) + 0.1V(s_3) + 0.4V(s_4)$. Noted that state s_1 has three PDs, thus three inequalities are required. $V(s_0)$ is then obtained by solving such linear programming using standard algorithms.

Value Iteration. This method iteratively builds an approximation of V based on the previous approximation. Let V^i be the i-th approximation. For $\forall s \in X \setminus G$, we have $V^0(s) = 0$; $V^{i+1}(s) = \max\{\sum_{t \in S} Pr(s, a)(t) \cdot V^i(t) \mid a \in Act(s)\}$. For example, at the 1^{st} iteration, $V^1(s_2) = 0.5V^0(s_1) + 0.4 + 0.1V^0(s_3) = 0.4$ and the others remains unchanged; at the 2^{nd} iteration, $V^2(s_1) = \max\{0.1V^1(s_2) + 0.9V^1(s_3), 0.5V^1(s_2) + 0.5V^1(s_3), 0.9V^1(s_2) + 0.1V^1(s_3)\} = 0.36$, $V^2(s_0)$ = 0.2; at the 3^{rd} iteration, $V^3(s_0) = 0.38$ and $V^3(s_3) = 0.2V^2(s_1) = 0.072$; at the 4^{th} iteration, since the value of state s_3 has been updated in the previous round, $V^4(s_1)$ and $V^4(s_2)$ shall be computed again and the similar iterations repeat. Notice that states s_1, s_2 and s_3 form a loop, within which an update of any state will trigger the updates of other states in the next few iterations. After 39 iterations, $V^{39}(s_0)$ is calculated to be 0.74627.

It can be shown that for every state s, $V^{i+1}(s) \geqslant V^i(s)$ and we can obtain V in the limit, $\lim_{i\to\infty} V^i = V$. In reality, it may take many iterations before V^i converges and thus value iteration is often stopped when the absolute/relative difference between two successive iterations falls below a certain threshold ϵ. The number of iterations required is related to the subdominant eigenvalue of the transition matrix [14]. Each iteration involves a series of matrix-vector multiplications, with a complexity of $\mathcal{O}(n^2 \cdot m)$ in the worst case, where n is the number of states in S and m is the maximum number of actions/distributions from a state. However, as stressed in [9], value iteration does not guarantee the resulting values to be within ϵ of the true answer. Although theoretically guaranteed precisions are base on the denominators of the (rational) numbers, it is still unclear if these are practically applicable.

3 SCC Reductions on Markov Decision Processes

As both approaches based on solving linear programming and value iteration have their own limitations, we propose a new approach to abstract away the loops in each strongly connected component (SCC) of an MDP based on a divide-and-conquer algorithm, and then apply value iteration to the resulting acyclic MDP. Without loops, the calculation of reachability probabilities will be faster, and also will be more accurate than the pure value iteration case with an unspecified amount of errors.

Reducing SCCs in an MDP while preserving the results of reachability analysis is highly nontrivial, and may lead to extra schedulers and an exponential increase in the number of probability distributions (PDs) if not handled properly. In this work, the proposed divide-and-conquer algorithm works on partitions; hence effectively avoids the generation of extra PDs. Moreover, we can further reduce the redundant PDs based on the convex property.

In the following, we will use a running example to illustrate the main idea of the divide-and-conquer approach, and then present the overall algorithm and detailed methodologies on performing state abstraction in an MDP, followed by its optimization on the reductions of probability distributions.

3.1 A Running Example

To reduce an SCC, our reduction approach starts from adding each state in the SCC into a new partition. It then divides these partitions into groups. For each group, it eliminates loops within the group and merges its components into a new partition. We call this process abstraction. This step repeats until the whole SCC becomes one partition, which is guaranteed to be free of loops. In this part, we demonstrate our main idea with a running example that transfers the MDP in Fig. 1(a) to the acyclic MDP in Fig. 1(b). The execution of each step is demonstrated in Fig. 2.

First, the states $\{s_1, s_2, s_3\}$ are identified as the only nontrivial SCC in the MDP, and there are three partitions, i.e., $\{s_1\}, \{s_2\}, \{s_3\}$, labeled using different grayscale in Fig. 2(a). Let Λ be the set of all current partitions in the SCC, i.e., $\Lambda = \{\{s_1\}, \{s_2\}, \{s_3\}\}$. We then divide Λ into two groups, as enclosed by dashed lines, such that the partitions $\{s_1\}$ and $\{s_2\}$ form one group, and $\{s_3\}$ alone forms the other.

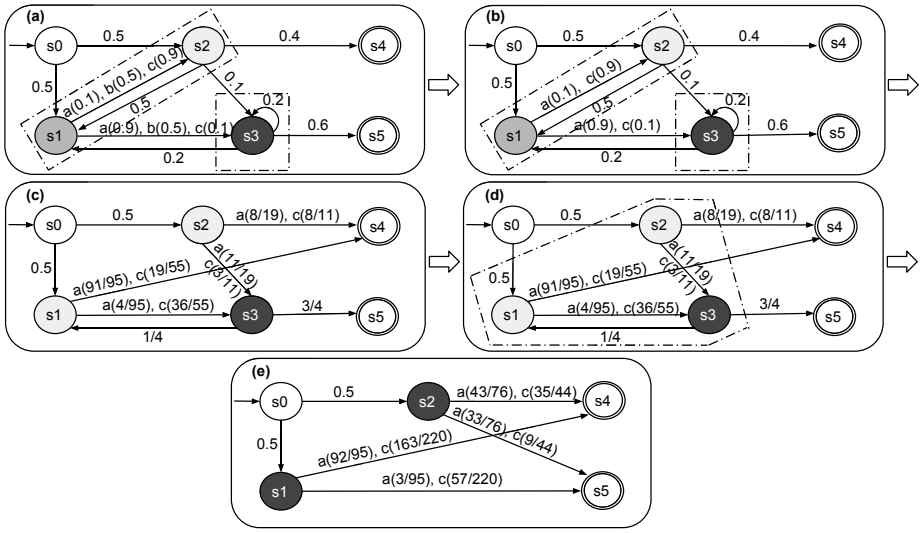

Fig. 2. A running example of transforming the MDP in Fig. 1(a) to the acyclic MDP in Fig. 1(b)

Subsequently, abstraction is performed on both groups. The main idea of the abstraction is to eliminate loops in the group by connecting the inputs and outputs using equivalent non-redundant probability distributions (PDs). In the first step, we need to remove the redundant probability distributions in each partition of the group. Recall that each PD can form a linear constraint according to Eq. (1) in Section 2.2. According to the PDs in Fig. 2(a), it can be proved that the constraint from PD b of state s_1 is redundant as it can be represented by a linear combination of the constraints from PDs a and c. As a result, PD b can be removed. The updated MDP is shown in Fig. 2(b).

The second step of the abstraction is to calculate the equivalent PDs. In the present case, partition $\{s_1\}$ has two actions and partition $\{s_2\}$ has only one action, thus there are two $(2 \cdot 1)$ schedulers in total. We define σ_1 as the scheduler selecting PD a at partition $\{s_1\}$, based on which a set of linear equations can be formed as

$$V(s_1) = 0.1V(s_2) + 0.9V(s_3); \quad V(s_2) = 0.5V(s_1) + 0.1V(s_3) + 0.4V(s_4) \quad (2)$$

Similar definition applies to scheduler σ_2 for PD c, we have

$$V(s_1) = 0.9V(s_2) + 0.1V(s_3); \quad V(s_2) = 0.5V(s_1) + 0.1V(s_3) + 0.4V(s_4) \quad (3)$$

To eliminate the transitions between s_1 and s_2, we need to first select a particular scheduler, and then perform Gauss Jordan elimination. Under the selection of scheduler σ_1, we can have the following new transitions based on Eq. (2),

$$V(s_1) = \frac{4}{95}V(s_3) + \frac{91}{95}V(s_4); \quad V(s_2) = \frac{11}{19}V(s_3) + \frac{8}{19}V(s_4) \quad (4)$$

Similarly, with the selection of σ_2, we have the following based on Eq. (3),

$$V(s_1) = \frac{36}{55}V(s_3) + \frac{19}{55}V(s_4); \quad V(s_2) = \frac{3}{11}V(s_3) + \frac{8}{11}V(s_4) \quad (5)$$

As a result, the updated PDs can be established based on Eq. (4) and Eq. (5). As illustrated in Fig. 2(c), a new partition can then be formed by grouping states s_1 and s_2, and states s_3 and s_4 continue to serve as outputs. Each state (s_1 or s_2) in the new partition now has two PDs (i.e. a and c), which appears to create a larger number ($2 \cdot 2 = 4$) of schedulers. However, it should be noted that the newly generated PDs in s_2 are derived based on the choice of scheduler in s_1 and thus **not** independent. For example, Eq. (4) and (5) are derived based on Eq. (2) and (3), respectively. That means a scheduler selects action a in s_1 and action c in s_2 is equivalent to a non-memoryless scheduler in the original MPD (with both selections of a and c at s_1). Therefore, two of the four schedulers in the new partition are equivalently non-memoryless and thus redundant for obtaining the maximum and minimum reachability probabilities (please refer to Section 2.2). Effectively, the number of schedulers to be handled in the new partitions remains as two. To easily allocate these schedulers, we denote the PDs for s_1 and s_2 obtained from the same set of equations by the same index or the same action name. Thus, given a partition, a scheduler only selects an index or an action, which means the PD with that index or action will be selected at each state. Similarly, we can obtain the abstraction on the other partition $\{s_3\}$. The resulting MDP as shown in Fig. 2(c) has only two partitions ($\Lambda = \{\{s_1, s_2\}, \{s_3\}\}$) in the SCC, both of which are free of loops and redundant PDs.

To finally achieve a single partition, another round of grouping and abstraction needs to be performed. There are now two partitions, and we combine them into one group as shown in Fig. 2(d). As explained above, during the calculation of the maximum and minimum reachability probabilities, partition $\{s_1, s_2\}$ can be described using two schedulers, and the other partition $\{s_3\}$ has only one scheduler. Therefore, the total number of schedulers within the group is two (i.e., $2 \cdot 1$). Let σ_3 be a scheduler selecting the PD of action a, i.e., $\sigma_3(\{s_1, s_2\}) = a$, and σ_4 be the other scheduler selecting the PD with action c, i.e., $\sigma_4(\{s_1, s_2\}) = c$. A set of linear equations can be formed similarly as Eq. (2) and (3), and the solutions connect the input states s_1 and s_2 directly to the output states s_4 and s_5. With such a new partition, the inner state s_3 can be removed from the MDP. Up to this point, there is only one partition left and our reduction finishes. The final acyclic MDP is shown in Fig. 2(e).

3.2 Overall Algorithm

The overall algorithm for SCCs reduction is presented in Algorithm 1. It is based on a divide-and-conquer approach that works on *partitions* of an MDP. Given a set of states S, a partition $\mathcal{E}$ is a subset of S such that $\bigcup_i \mathcal{E}_i = S$; and $\forall \mathcal{E}_i, \mathcal{E}_j, \mathcal{E}_i \neq \mathcal{E}_j, \mathcal{E}_i \cap \mathcal{E}_j = \emptyset$. Given an MDP $\mathcal{M} = (S, S_{init}, Act, Pr)$ and target states $G \subset S$, Algorithm 1 removes all loops in $\mathcal{M}$ (i.e., resulting an acyclic MDP $\mathcal{M}'$) and computes reachability probabilities in $\mathcal{M}'$. We remove loops according to the following steps.

- Line 1 finds all SCCs by Tarjan's approach [16], and adds all nontrivial SCCs to $\mathcal{C}$. Lines 2–12 present the divide-and-conquer procedure for each SCC in $\mathcal{C}$. Let Λ be a set of partitions of an SCC. Initially, each state of SCC forms a partition in Λ, as shown in lines 3–4.
- Lines 5–12 perform the divide-and-conquer in the partitions of Λ until there is only one partition left. Within each round, line 6 first divides all the partitions Λ into several groups, denoted by $\mathcal{A}$. Here, the groups are formed dynamically that each

Algorithm 1. SCC Reduction in an MDP via Divide-and-Conquer

input : An MDP $\mathcal{M} = (S, s_{init}, Act, Pr)$, target states $G \subseteq S$
output: $\mathcal{P}(\mathcal{M} \models \Diamond G)$

```
1  M' = M; C := the set of all nontrivial SCCs in M';
2  for each D ∈ C do
3      Λ := ∅;      //to record a set of partitions
4      ∀s ∈ D, Λ := Λ ∪ {{s}};   //each state is a partition initially
5      repeat
6          Divide Λ into a set of groups of partitions denoted as A;
7          Λ' = ∅;
8          for each J ∈ A do
9              E' = Abstraction(J);      //J is a set of partitions
10             Λ' = Λ' ∪ E';
11         Λ = Λ';
12     until |Λ| == 1;
13 return ValueIteration(M', G) ;
```

has relatively small number of output states. Each element $\mathcal{J}$ in $\mathcal{A}$ is a group of partitions. There is always a group containing more than one partitions unless there is only one partition in $\mathcal{A}$. Next, lines 8–10 remove loops and the inner states in each $\mathcal{J}$ through $Abstraction()$ method, which takes a group of partitions as the input and returns a new acyclic partition that can represent the previous group. As a result, after each round, the number of partitions decreases and loops inside each partition are eliminated. Details for the abstraction process will be presented in Section 3.3.

– After the iteration terminates, the resulting MDP becomes acyclic. The standard value iteration method, detailed in Section 2.2, can then be applied to calculate the probability from the initial state to the target states efficiently.

As we can see, in order to support the divide-and-conquer algorithm for MDPs, the overall algorithm incorporates methods like abstraction and PD reduction. In the following parts, we will introduce details of these two methods.

3.3 States Abstraction

Given a set of partitions, denoted by $\mathcal{J}$, the *abstraction* process removes the inner states in each partition, and merges all partitions into a new partition, denoted by $\mathcal{E}'$. The detailed algorithm of abstraction is presented in Algorithm 2. It takes $\mathcal{J}$ as the input and returns a new acyclic partition $\mathcal{E}'$. The procedure works as follows.

– The first step, as shown in lines 1–7, is to reduce redundant PDs in each partition. As demonstrated in Section 3.1, within a partition, the PDs of the same index are originated from the same scheduler in the original model. Thus, they are not independent and can only be removed if they are all redundant. The detailed operations are as follows. For each partition, we use a Boolean set $\mathcal{I}$ to record whether a PD is redundant. Initially, line 2 sets all elements in $\mathcal{I}$ to $false$. For each state of the partition, line 4 gets all indices of the non-redundant PDs, and line 5 sets the respective elements in

Algorithm 2. Abstraction

input : A set of partitions of states $\mathcal{J}$ in an MDP
output: A new partition $\mathcal{E}'$

```
    //step 1: remove redundant PDs in each partition
1   for each E ∈ J do
        //I is to record whether a PD is non-redundant
2       Let I be a set of Boolean variables initialized with false;
3       for each s ∈ E do
4           Indices := indices non-redundant PDs of s;
5           for each index ∈ Indices do I'[index] =: true; ;
6           I = I';
7       for each s ∈ E do Update PDs according to I;;
    //step 2: calculate new PDs from inputs to outputs
8   K = ⋃_{E∈J} E;
9   ∀s ∈ Inp(K) · U'_s := ∅;
10  Σ := all the schedulers in J based on partitions;
11  for each σ ∈ Σ do
12      calculate PDs from Inp(K) to Out(K) according to σ;
13      Let u_s be the calculated PD of a input state s;
14      ∀s ∈ Inp(K) · U'_s := U'_s ∪ {u_s};
    //step 3: form a new partition
15  E' = Inp(K) ;
16  ∀s ∈ E', replace PDs of s by U'_s;      //re-connect Inp(E') to Out(E')
17  return E';
```

$\mathcal{I}$ to $true$. Here, the non-redundant PDs can be identified by finding the vertices of the convex hull, detailed in Section 3.4. After the **for** loop in lines 3 - 6, a $false$ in $\mathcal{I}$ means the corresponding PD in each state is redundant. As a result, line 7 removes the respective PDs at the indices for all states.

– Line 8 combines states in all partitions of $\mathcal{J}$ into one group $\mathcal{K}$. The second step is to calculate new PDs from $Inp(\mathcal{K})$ to $Out(\mathcal{K})$ for all schedulers. Line 9 creates an empty set for each state in $Inp(\mathcal{K})$, which is used to store new PDs. Line 10 finds all the schedulers in $\mathcal{J}$ and assigns them to Σ. As reviewed in Section 2.1, for any given state, a scheduler is used to select a PD, and the total number of the schedulers is exponential to the number of states. As mentioned, within a partition, the PDs with the same index are not independent, we thereby create a scheduler in such a way that it can only select PDs with the same index at all states in the partition. This can avoid the generation of extra schedulers by including all the combinations of PDs. Lines 11 –14 calculate the new equivalent PDs by calculating the transition probabilities, from $Inp(\mathcal{K})$ to $Out(\mathcal{K})$. For each scheduler σ, we calculate the probabilities from any input to output states in the DTMC $\mathcal{K}^\sigma$, which can be done by the standard algorithm, e.g., Gaussian Jordan elimination. Line 14 adds the new PDs to each state.
– Since the sets of PDs from $Inp(\mathcal{K})$ to $Out(\mathcal{K})$ have been obtained, the inner states of $\mathcal{K}$ are then redundant for the calculation of reachability probabilities. As a result,

line 15 creates a new partition $\mathcal{E}'$ by adding only the inputs states of $\mathcal{K}$, and updates the PDs of each state in $\mathcal{E}'$ by $\mathbf{U}'_s$. The new partition $\mathcal{E}'$ is free of loops.

3.4 Reduction of Probability Distributions Based on Convex Hull

Within a set of probability distributions (PDs), if a PD can be represented by a convex combination of the other PDs, we call it a *redundant* PD. As demonstrated, PD b in Fig. 1(a) is redundant as it can be represented by a combination of 50% of PD a and 50% of PD b. It can be proved that the redundant PDs are irrelevant to the maximum and minimum reachability probabilities [6].

There are two scenarios that might introduce redundant PDs. One is during system modeling. For instance, PDs could be originated from a set of working profiles (modeling complex system environment) and some of working profiles are indeed redundant for calculating the maximum or minimum probability. The other is during the removal of the inner states within a group of states $\mathcal{K}$. The equivalent PDs are created to connect inputs to outputs of $\mathcal{K}$, the number of those is equal to the total number of schedulers in $\mathcal{K}$. As a result, there could be redundant PDs, especially when obtained PDs of a state have only a few successive states. In fact, the number of PDs of a state can be minimized and replaced by a *unique* and *minimal* set of PDs. If we consider PDs as a set of points in a Euclidean space and each successive state in a PD provides a dimension in the Euclidean space, finding the set of non-redundant PDs is equivalent to the problem of identifying all the vertices of the convex hull of all the PDs. This has been already proved in [6]. In the following, we have a brief review on the convex hull property.

The *convex hull* of a set Q of points, denoted by $CH(Q)$, is the smallest convex polygon or polytope in the Euclidean plane or Euclidean space that contains Q [8]. Mathematically, the convex hull of a finite point set, e.g., $Q = \{\mathbf{q}_1, \cdots, \mathbf{q}_n\}$, is a set of all *convex combinations* of each point $\mathbf{q}_i$ assigned with a coefficient r_i, in such a way that the coefficients are all non-negative with a summation of one; i.e., $CH(Q) = \{\sum_{i=1}^{n} r_i \cdot \mathbf{q}_i | (\forall i : r_i \geqslant 0) \wedge \sum_{i=1}^{n} r_i = 1\}$. We denote the set of *vertices of a convex hull* as $V_{CH}(Q)$. Each $\mathbf{q}_i \in V_{CH}(Q)$ is also in Q, but it is not in the convex hull of the other points (i.e., $\mathbf{q}_i \notin CH(Q \setminus \{\mathbf{q}_i\})$). In other word, the points $V_{CH}(Q)$ are the essential points that generate all the other points in $CH(Q)$ via a convex combination. Given a set of n points (Q) in d-dimension, the algorithms to determine the vertices of the convex hull are also known as the redundancy removal for a point set Q in $\mathbb{R}^d$. This problem can be reduced to solving $\mathcal{O}(n)$ linear programming problems with many polynomial time algorithms available [6].

To further accelerate the calculation, we adopt an approximation algorithm proposed by Bentley et al. [5], who use the convex hull of some *subset* of given points as an approximation to the convex hull of all the points. Here, a user-defined parameter β controls degree of approximation. For instance, in xy-plane, we first divide the area between the minimum and maximum (i.e., extreme) values in x-dimension into 'strips', with a width of β. We then select the points with the extreme values in y-dimension within each strip, and the points with x-dimension extreme. Last, we construct the convex hull based on these selected points (in the worst case, there are only $2(1/\beta + 2)$ points). Here, β specifies the relative approximation error; i.e., any point outside the approximate hull is within β distance of the 'true' hull, as proved in [5]. Hence, a larger

β implies a faster calculation but a coarser approximation. In terms of reachability analysis, the schedulers, after approximation, are only a subset of original ones. Ignoring some of the PDs means the maximum or minimum reachability probability will be a safe approximation; i.e., the maximum probability is smaller than the 'true' maximum, and the minimum probability is larger than the 'true' minimum.

3.5 Termination and Correctness

In this section, we discuss the termination and the correctness of our approach.

Theorem 1. *Given a finite states MDP, Algorithm 1 always terminates.*

Proof: Given a finite number of states, the **for** loop in Algorithm 1 always terminates as the number of SCCs is finite. The theorem can then be proved by showing (1) the **repeat** loop can terminate and (2) $Abstraction()$ can also terminate.

For (1), the proof for the one state SCC is trivial. For an SCC having more than one states, there are at least one group in $\mathcal{A}$ that has more than one partition, which can be merged into one new partition through $Abstraction()$. The total number of partitions is guaranteed to decrease after each round of the **repeat** loop. Thus the termination condition $|\Lambda| == 1$ can always be fulfilled. For (2), the abstraction, as in Algorithm 2, always terminates because all **for** loops work on a finite set of elements. As both conditions are fulfilled, the theorem holds. □

Theorem 2. *Given a finite states MDP, Algorithm 1 always produces an acyclic MDP.*

Proof: To prove the theorem, it is equal to show that Algorithm 1 can remove all loops in each SCC. As proved above, Algorithm 1 always transfers each SCC into one partition, the theorem can be proved by showing that the abstraction process always returns a loop-free partition. Assuming a set of partitions $\mathcal{J}$ are the input, Algorithm 2 always creates a new partition by recalculating the probability distributions from $Inp(\mathcal{J})$ to $Out(\mathcal{J})$. As $Inp(\mathcal{J}) \cap Out(\mathcal{J}) = \emptyset$, the new partition is guaranteed to be acyclic. Therefore, the theorem holds. □

As Algorithm 1 always terminates with an acyclic MDP, our approach can always provide an accurate result. Recall that loops in each SCC of the MDP are resolved by solving sets of equations, which is based on an accurate method. Further, we could trade off a certain level of accuracy for better performance with approximate convex hull.

4 Implementation and Evaluation

We implement the algorithm in our model checking framework PAT [15]. As the only difference between the ordinary and our proposed value iteration methods is the algorithm of reachability analysis, it is fair to check the effectiveness of the new method through direct comparison of their performance. Hereafter, we refer the implementations with and without our approach as PAT(w) and PAT(w/o), respectively. For the value iteration method, we use the default stopping criterion in PAT, i.e., the maximum ratio of difference is $1E$-6. For the new approach, we set the maximum number of partitions in a group to 3, and the parameter for convex hull approximation to 0.001. The

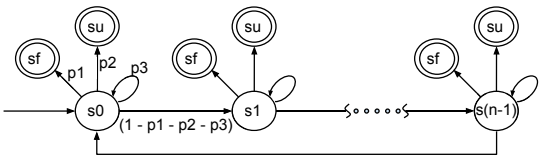

Fig. 3. A reliability model, the states s_u and s_f are copied for a clear demonstration

testbed is an Intel Xeon® CPU at 2.67 GHz with 12 GB RAM. All related materials, including the tools, models, and evaluation results, are available at [1]. We perform an analysis on two case studies: one is software reliability assessment model and the other is tennis tournament prediction model. Both systems have many probability transitions and loops, thus may encounter slow convergence issue especially when the systems become large. Thus, we evaluate how our new approach can benefit those cases.

4.1 Case Study on Software Reliability Assessment

Reliability and fault tolerance are central concerns to many software systems. The reliability problem can be transferred into a reachability problem in an MDP [10,12]. In this case study, we model a system that undergoes n tasks and then standbys at the initial state. Each task is exposed to a certain probability of failure or self-recovering situation, before successfully transferring to the next task or service. A highly abstracted reliability model is shown in Fig. 3, which consists of $n + 2$ states, i.e., $\{s_f, s_u, s_0, s_1, \cdots, s_{n-1}\}$, representing different system status. The failure state s_f is the state that the system fails, and the success state s_u is the state that the system finishes a requirement successfully. Each state s_i transits to s_f with a probability of p_1; to s_u with a probability of p_2; to itself with a probability of p_3; and otherwise, to the next state $s_{(i+1)\%n}$. Multiple sets of values for $\{p_1, p_2, p_3\}$ are considered. We then perform reachability analysis, e.g., computing the maximum probability of reaching state s_u, under different scale by varying the parameters n and m, where n controls the number of states and m is the number of probability distributions of each state.

The experiments are summarized in Table 1. The number of states being generated is approximately equal to n; $Trans.$ represents the total number of transitions in the model; P_{max} represents the maximum reachability probability; and $Time$ represents the total time spent on the verification. We have the following observations.

– The overall verification time of the new approach (PAT(w)) is much less than that of the previous approach (PAT(w/o)). Three factors here can affect the rate of value iteration in this model: (1) the self-loops at each state s_i; (2) the large SCC formed by $\{s_0, s_1, \cdots, s_{n-1}\}$; and (3) the various probability distributions in the model. Our approach reduces loops prior to value iteration, as detailed in Section 3. With PAT(w), the resulting acyclic MDP consists of only three states, s_0 (the only input of the SCC), and s_f and s_u (the outputs of the SCC). Thus, time spent on value iteration can almost be negligible (less than 0.001s). In addition, due to the PD reductions based on the convex hull, our reduction approach can work under many probability distributions without much overhead, as evidenced by the cases with $m = 10$.

Table 1. Comparison between PAT with and without SCC reduction for reliability model

Parameters			PAT (w/o)		PAT (w)	
m	n	#Trans.	Pmax	Time(s)	Pmax	Time(s)
4	40	0.6K	0.499985	0.03	0.500000	0.01
	400	6K	0.499999	0.22	0.500000	0.13
	20K	320K	0.499999	547.52	0.500000	55.97
	40K	640K	0.499999	1389.55	0.500000	314.73
10	40	2K	0.499985	0.04	0.500000	0.11
	400	16K	0.499999	0.41	0.500000	0.20
	20K	800K	0.499999	894.34	0.500000	111.62
	40K	1600K	0.499999	2168.04	0.500000	597.44

States $\approx n$

– The result obtained from the new approach is closer to the true value. Through manual analysis, we know that 0.5 is the accurate result. In fact, our reduction approach removes loops by solving a set of linear equations, which yields accurate results. As mentioned above, the resulting model is an acyclic MDP of only three states, on which value iteration stops naturally without using any stopping criterion. On the other hand, the ordinary value iteration approach keeps iterating over loops until a stopping criterion is met, thus the result is an approximation.

The experiment above considers only one SCC in the reliability model. However, often, a system may have a large number of SCCs in its reliability model. Our preliminary result shows that, with the increase of SCCs, the total time increases exponentially for the ordinary value iteration approach, while remains at a low level with our approach [1]. This is because our approach resolves each SCC independently while the ordinary approach has to iterate over all SCCs until converging to a stable result.

4.2 Case Study on Tennis Tournament Prediction

A tennis match is won when a player wins the majority of prescribed sets. At a score of 6 - 6 of a set, an additional 'tiebreaker' game is played to determine the winner of the set. In this case study, we model a 7 point tiebreaker. Our model encodes the outcomes of individual player's actions (e.g., serve and baseline) according to the past scoring profiles available at `http://www.tennisabstract.com`, and predicts the winning probability for one player against the other. In particular, we predict the game between two tennis giants Federer and Nadal. A play wins the set if he wins one tiebreaker, or best of 3 (or 5) tiebreakers. Thus, we analyze all the three situations. For each situation, we calculate four probabilities: (a) Federer scores the first point in any tiebreaker; (b) Nadal scores the first point in any tiebreaker; (c) Federer wins the set; and (d) Nadal wins the set.

The verification results are shown in Table 2. $\#$ represents the numbers of tiebreakers; $Pro.$ represents the properties to be verified; $\#States$ and $\#Trans.$ represent the total numbers of states and transitions in the system, respectively; P_{min}/P_{max} records the minimum/maximum reachability probability; and B and V record the time costs on building the MDP model (for PAT(w), it includes the additional time spent on SCC

Table 2. Comparison between PAT with and without SCC reduction for tennis prediction model

#	Pro.	#States	#Trans.	PAT (w/o)				PAT (w)			
				Pmin	Pmax	B (s)	V (s)	Pmin	Pmax	B (s)	V (s)
1	a	15K	26K	0.4585	0.5077	0.16	0.01	0.4585	0.5077	0.22	0.00
	b	15K	26K	0.4923	0.5415	0.14	0.01	0.4923	0.5415	0.24	0.00
	c	17K	30K	0.4678	0.4786	0.19	13.44	0.4678	0.4786	0.58	0.33
	d	17K	30K	0.5214	0.5322	0.16	13.34	0.5214	0.5322	0.50	0.32
3	a	62K	108K	0.7877	0.8075	0.66	64.72	0.7877	0.8075	1.55	2.94
	b	62K	108K	0.8116	0.8303	0.64	65.54	0.8116	0.8303	1.48	2.96
	c	71K	123K	0.4576	0.4649	0.74	133.89	0.4576	0.4649	1.95	9.32
	d	71K	123K	0.5351	0.5424	0.72	133.03	0.5351	0.5424	1.98	8.45
5	a	141K	278K	0.9194	0.9271	1.42	266.26	0.9194	0.9271	3.66	23.25
	b	141K	245K	0.9332	0.9401	1.43	265.80	0.9332	0.9401	3.65	23.35
	c	160K	279K	0.4486	0.4554	1.58	434.29	0.4486	0.4554	4.37	41.65
	d	160K	278K	0.5446	0.5514	1.53	428.62	0.5446	0.5514	4.32	36.93

reduction) and on value iteration, respectively. Notice that the summation of these two time costs is the total time spent on the verification. We have the following observations.

Comparing the time costs in B and V columns, for the ordinary approach, though the time for building an MDP model is very short, the verification time increases quickly when the size of system becomes large. On the other hand, with slightly longer time spent on model building, our new approach reduces the value iteration time significantly. This is because the new approach removes all SCCs prior to value iteration and the probability computation is thereby accelerated. In this case study, both approaches generate the same results up to four decimal points.

5 Related Work and Conclusion

In recent years, some approaches [11,7,3,2,13] have been proposed to improve probability reachability calculation. The key idea is to reduce iterations on the state space. [11,7] improve value iteration in MDPs by backward iterating over each SCC in topological order, i.e., an SCC will not be visited until the reachability probabilities of all its successive SCCs converge. However, since it requires iterating over each SCC (i.e., SCC-based value iteration), this approach only alleviates the slow convergence problem to a certain degree without completely solving the problem. Compared to their SCC based value iteration approach, our approach eliminates SCCs and produces an acyclic MDP where the standard value iteration is applied. Moreover, our reduction on each SCC is independent to others, so that multi-cores or distributed computers can be directly applied, which can make the verification even faster.

The approaches [3,2,13] are on SCCs elimination by connecting inputs to outputs of an SCC with equivalent probability transitions. But they are only applicable to DTMCs. In particular, the algorithms proposed in [2] and [13] can both work with large SCCs. [2] iteratively searches for and solves the smallest loops within an SCC. [13] uses a divide-and-conquer algorithm that iteratively divides an SCC into several smaller parts and resolves loops in each part. However, eliminating loops in an MDP is particularly challenging due to the existence of many probability distributions. To the best of our knowledge, there has been no previous work on SCC reductions for MDP. Instead of a

simple extension of the divide-and-conquer for DTMC in [13], our divide-and-conquer algorithm for MDP is carefully designed to avoid generation of extra schedulers. To further accelerate the elimination of loops, we actively detect and remove redundant probability distributions of each state based on the convex hull property.

Conclusion. In this work, we have proposed a divide-and-conquer algorithm to eliminate SCCs in MDPs, for achieving an efficient reachability analysis. To cope with the non-determinism in MDPs, our divide-and-conquer algorithm is designed to work on partitions. Initially, each state in an SCC is considered as a partition. The partitions are repeatedly merged together until there is only one left. During the abstraction, loops within a partition are replaced by equivalent probability distributions between inputs and outputs. The convex hull property is applied to further reduce the redundant probability distributions. We have implemented this algorithm in a model checker PAT. The evaluation results on two practical case studies show that our method can improve reachability analysis.

References

1. `http://www.comp.nus.edu.sg/~pat/rel/mdpcut`
2. Abrahám, E., Jansen, N., Wimmer, R., Katoen, J., Becker, B.: DTMC model checking by SCC reduction. In: QEST, pp. 37–46. IEEE (2010)
3. Andrés, M.E., D'Argenio, P.R., Rossum, P.V.: Significant diagnostic counterexamples in probabilistic model checking. In: HCV, pp. 129–148 (2008)
4. Baier, C., Katoen, J.: Principles of model checking. The MIT Press (2008)
5. Bentley, J.L., Preparata, F.P., Faust, M.G.: Approximation algorithms for convex hulls. Communications of the ACM 25(1), 64–68 (1982)
6. Cattani, S., Segala, R.: Decision algorithms for probabilistic bisimulation. In: Brim, L., Jančar, P., Křetínský, M., Kučera, A. (eds.) CONCUR 2002. LNCS, vol. 2421, pp. 371–386. Springer, Heidelberg (2002)
7. Ciesinski, F., Baier, C., Grosser, M., Klein, J.: Reduction techniques for model checking Markov decision processes. In: QEST, pp. 45–54. IEEE (2008)
8. De Berg, M., Van Kreveld, M., Overmars, M., Schwarzkopf, O.C.: Computational geometry. Springer, Heidelberg (2000)
9. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: FMENSS, pp. 53–113. Springer, Heidelberg (2011)
10. Gui, L., Sun, J., Liu, Y., Si, Y.J., Dong, J.S., Wang, X.Y.: Combining model checking and testing with an application to reliability prediction and distribution. In: ISSTA, pp. 101–111. ACM (2013)
11. Kwiatkowska, M., Parker, D., Qu, H.: Incremental quantitative verification for Markov decision processes. In: DSN, pp. 359–370. IEEE (2011)
12. Liu, Y., Gui, L., Liu, Y.: MDP-based reliability analysis of an ambient assisted living system. In: FM Industry Track, Singapore (May 2014)
13. Song, S., Gui, L., Sun, J., Liu, Y., Dong, J.S.: Improved reachability analysis in DTMC via divide and conquer. In: Johnsen, E.B., Petre, L. (eds.) IFM 2013. LNCS, vol. 7940, pp. 162–176. Springer, Heidelberg (2013)
14. Stewart, W.J.: Introduction to the numerical solution of Markov chains. Princeton University Press (1994)
15. Sun, J., Liu, Y., Dong, J.S., Pang, J.: PAT: Towards flexible verification under fairness. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 709–714. Springer, Heidelberg (2009)
16. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM J. Comput. 1(2), 146–160 (1972)