Improved Reachability Analysis in DTMC via Divide and Conquer

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

[1] National University of Singapore
{songsongzheng,lin.gui}@nus.edu.sg, dongjs@comp.nus.edu.sg
[2] Singapore University of Technology and Design
sunjun@sutd.edu.sg
[3] Nanyang Technological University
yangliu@ntu.edu.sg

Abstract. Discrete Time Markov Chains (DTMCs) are widely used to model probabilistic systems in many domains, such as biology, network and communication protocols. There are two main approaches for probability reachability analysis of DTMCs, i.e., solving linear equations or using value iteration. However, both approaches have drawbacks. On one hand, solving linear equations can generate accurate results, but it can be only applied to relatively small models. On the other hand, value iteration is more scalable, but often suffers from slow convergence. Furthermore, it is unclear how to parallelize (i.e., taking advantage of multi-cores or distributed computers) these two approaches. In this work, we propose a divide-and-conquer approach to eliminate loops in DTMC and hereby speed up probabilistic reachability analysis. A DTMC is separated into several partitions according to our proposed cutting criteria. Each partition is then solved by Gauss-Jordan elimination effectively and the state space is reduced afterwards. This divide and conquer algorithm will continue until there is no loop existing in the system. Experiments are conducted to demonstrate that our approach can generate accurate results, avoid the slow convergence problems and handle larger models.

1 Introduction

As an automatic verification technique, model checking [7] has been applied to a variety of domains from hardware to software, and from concurrent systems to probabilistic systems. Different from traditional concurrent systems, probabilistic systems have stochastic characteristics in their behaviors, which means some behaviors follow specific probabilistic distributions. This kind of systems widely exist in many domains, from communication protocols to biology systems. For example, in the randomized leader election protocol [9], multiple processes want to elect one leader. Each process will first randomly choose a natural number from a specific range as its id. The process with a unique highest id will be elected as a leader. If several processes have the same highest id, the selection procedure will repeat. Therefore uniform distribution is necessary in this system. As a result, model checking probabilistic systems is an important topic in formal verification.

E.B. Johnsen and L. Petre (Eds.): IFM 2013, LNCS 7940, pp. 162–176, 2013.
© Springer-Verlag Berlin Heidelberg 2013

Discrete Time Markov Chain (DTMC) is a widely used formalism in probabilistic model checking. The difference between DTMC and traditional Labeled Transition System (LTS) is that non-determinism in LTS is replaced by probabilistic choices in DTMC. In a DTMC, at each step the transition from one state to another must follow specific probability distributions, and for each state there is exactly one probability distribution for the successor states. Reachability analysis plays a key role in DTMC verification, e.g., it is used to decide the probability of reaching certain disastrous state. Verification of properties such as Probabilistic Computational Tree Logic (PCTL) and Linear Temporal Logic (LTL) can be reduced to the reachability analysis problem [5]. E.g., for LTL properties a product construction with a deterministic Rabin/Muller-automaton is needed to obtain the target states. Therefore in this work we focus on improving reachability analysis in DTMC verification.

Given the transition relation of a DTMC, the transition probability matrix from one state to another can be built. After the target states are decided, each state in the matrix can be represented by a variable, which means the probability of reaching the target states from this state. Next, there are mainly two approaches to calculate the probability from initial states to the targets. One is solving linear equations directly. In this method, variables representing intermediate states (which are not target or initial) are eliminated gradually through equations operation, and finally variables representing the initial states' probability of reaching targets can be solved. The other approach is using value iteration method, which works by finding a better approximation iteratively until certain stopping criteria are satisfied. The approach based on solving linear equations is straightforward to understand and it guarantees to deliver accurate result. However, since we need one variable for each state in the system, a lot of variables are needed for large systems whereas state-of-the-art linear solvers are limited to thousands of variables only. Therefore the applicability of this approach is limited to small-scale systems. On the other hand, the value iteration method tries to find fix-points iteratively, and it has relatively better performance in handling systems with a large number of states. Therefore it is more popular in probabilistic model checkers such as PRISM [12] and MRMC [10,11]. However, this approach also has its drawback: slow convergence, i.e., it may take a large number of iterations before the approximations converge to a certain value. The phenomenon exists when there are complicated loops existing in the probabilistic systems, although the state space of such systems may not be very huge. The number of iterations is related to the subdominant eigenvalue of the probability transition matrix [18].

To tackle the above-mentioned problems, in this work we propose a new approach to verify DTMC models, especially for the ones with loops using a divide-and-conquer strategy. Instead of directly calculating the probability from initial states to targets, we divide the whole state space into several partitions, and solve them individually to eliminate loops. Afterwards, the remaining acyclic DTMC can be solved efficiently via value iteration method.

As we mentioned above, the slow convergence problem in value iteration comes from loops. Therefore, the first step of our approach is finding Strongly Connected Components (SCCs). This SCC-based approach is similar to previous work such as [3,6,1,13]. However, instead of using SCC's topology order [6,13], we solve each SCC independently by calculating the new transition probability from input states to output states of

the SCC, which is similar to work [3,1]. These new transitions are denoted as *abstract* transitions since SCCs are abstracted by transitions from input states to output states. However, [1] focuses on counterexample generation and abstracts SCCs via iteratively finding the smallest SCCs. On the contrary, we divide each SCC having a large number of states to several smaller partitions. For each partition, abstract transitions from its input to output are calculated via solving linear equations. Here we use Gauss-Jordan elimination [2]. Further, the states in each partition which are not input states will be removed, and thus the states in the SCC can be reduced. Afterwards, the new SCC is ready for next iteration of divide and conquer. This procedure for each SCC will be done iteratively until any of the following three criteria is satisfied. First, there is no more loop in the reduced SCC. Then this part will be left alone since it is already acyclic. Second, the number of remaining states in reduced SCC is small enough to be solved via a linear solver. Third, the last iteration does not reduce any states. In the second and third scenarios, the final SCC will be solved via linear equation again, and final abstract transitions will be generated. After all loops in SCCs are resolved, the whole DTMC becomes acyclic, and value iteration is used to calculate the probability from initial states to targets. Since the abstract transitions from each partition's input states to output states are determined by the partition itself and independent to other partitions, multi-cores or distributed computers can be straightforwardly used here to solve each partition simultaneously, which makes the verification faster.

Contributions Compared with previous work, our contribution is threefold, as we summarize below.

1. A new divide-and-conquer approach for DTMC reachability analysis is proposed, which combines solving linear equations and value iteration methods together and tackles the problem that huge loops make the DTMC verification inefficient.
2. Based on the fact that each SCC and even each group in one SCC is independent from others, we use parallel computation to further speed up the verification.
3. The new approach has been implemented into our model checking framework PAT, and several representative experiments are conducted to show the effectiveness of our approach.

Organization The paper is structured as follows. Section 2 recalls relative background. In Section 3, we introduce our algorithm in details. The evaluation is reported in Section 4. Section 5 surveys related work and concludes the paper.

2 Preliminaries

In this section, we recall some background knowledge, which is relevant in the rest of this paper.

2.1 Discrete Time Markov Chains

Discrete Time Markov Chains (DTMCs) are widely used in modeling stochastic systems. Meanwhile, time requirement in DTMC is discrete. Without loss of generality,

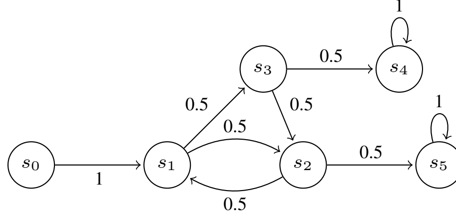

Fig. 1. An Example of SCC

we have the following two assumptions in this work. 1) There is only one initial state in the whole system and 2) DTMC is deadlock free. It is known that a deadlock state in a DTMC can add a self-loop having probability 1 without affecting the calculation result. The formal definition of DTMC is as follows.

Definition 1. *A Discrete Time Markov Chain is a tuple $\mathcal{M}$ = (S, s_{init}, Tr, AP, L) where S is a set of states; $s_{init} \in S$ is the initial state of the system; $Tr : S \times S \rightarrow [0, 1]$ is the probability transition relation between states, which satisfies $\forall s \in S$, $\Sigma_{s' \in S} Tr(s, s') = 1$; AP is a set of atomic propositions and L: $S \rightarrow 2^{AP}$ is a labeling function.*

An infinite or a finite *path* in $\mathcal{M}$ is defined as a sequence of states $\pi = \langle s_0, s_1, \cdots \rangle$ or $\pi = \langle s_0, s_1, \cdots, s_n \rangle$ respectively, such that $\forall i \leq 0$ (for finite paths, $i \in [0, n-1]$), $Tr(s_i, s_{i+1}) > 0$. The probability of exhibiting π in $\mathcal{M}$ is $\mathcal{P}_{\mathcal{M}}(\pi) = Tr(s_0, s_1) \times Tr(s_1, s_2) \times Tr(s_2, s_3) \times \cdots$. Given a set of paths Π of $\mathcal{M}$, $\mathcal{P}_{\mathcal{M}}(\Pi) = \sum_{\pi \in \Pi} \mathcal{P}_{\mathcal{M}}(\pi)$.

A set of states $C \subseteq S$ is called *connected* in $\mathcal{M}$ iff $\forall s, s' \in C$, there is 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 Connecte Components (SCCs) are those maximal sets of states which are mutually connected. An SCC is called *trivial* if it just has one state without a self-loop. An SCC is *nontrivial* iff it is not trivial. A DTMC is *acyclic* iff it only has *trivial* SCCs. Note that one state can only be in one SCC. In other words, SCCs are disjoint. In addition, we define an *adjacent group* (AG) $D \subseteq S$ such that $\exists s \in D$, $\forall s' \in D \wedge s' \neq s$, there is a finite path $\pi = \langle s_0, s_1, \cdots, s_n \rangle$ satisfying $s_0 = s \wedge s_n = s' \wedge \forall i \in [0, n], s_i \in D$, and s is called *root* state in D. In the following, we refer to adjacent groups simply as groups. The difference between these conceptions is illustrated by the example in Figure 1.

In Figure 1, $\{s_1, s_2\}$, $\{s_1, s_2, s_3\}$ are *connected*; $\{s_0\}$, $\{s_4\}$, $\{s_5\}$ and $\{s_1, s_2, s_3\}$ are the *SCCs* in the model; AGs are more complex, for example, $\{s_0, s_1, s_2\}$ and $\{s_1, s_2, s_5\}$ are AGs and there are other possible combinations. Note that a set of states like $\{s_0, s_1, s_4\}$ is not a valid AG because there is no root state. *Connected* subgraphs are AGs but the reverse is not always true, e.g., $\{s_0, s_1, s_2\}$ is an AG but not a *connected* subgraph.

Similar to [3,1], in a DTMC $\mathcal{M}$ = (S, s_{init}, Tr, AP, L), given a group of states $\mathcal{D} \subseteq S$, the input states of $\mathcal{D}$ are defined as the states in $\mathcal{D}$ having incoming transitions from states outside $\mathcal{D}$; the output states of $\mathcal{D}$ are defined as states outside $\mathcal{D}$ which have incoming transitions from states in $\mathcal{D}$. Formal definitions are as follows.

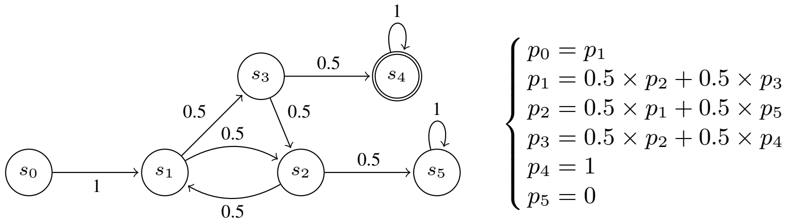

Fig. 2. Reachability Analysis

$$Inp(\mathcal{D}) = \{s' \in \mathcal{D} \mid \exists s \in S \backslash \mathcal{D}.Tr(s, s') > 0\}^1$$
$$Out(\mathcal{D}) = \{s' \in S \backslash \mathcal{D} \mid \exists s \in \mathcal{D}.Tr(s, s') > 0\}$$

2.2 Reachability Analysis

One critical question for quantitative analysis of DMTC models is to compute the probability of reaching a certain set of target states G from the initial state. Here $\Diamond G$ is used to denote the event of reaching G, and $\mathcal{P}_{\mathcal{M}}(s_{init} \models \Diamond G)$ represents the probability that G can be reached from initial state in a DTMC $\mathcal{M}$. Here $\mathcal{P}_{\mathcal{M}}$ can be written as $\mathcal{P}$ if $\mathcal{M}$ is clear. Let $\pi = \langle s_0, s_1, \cdots, s_n \rangle$ represent any finite path in $\mathcal{M}$. Then we have

$$\mathcal{P}(s_{init} \models \Diamond G) = \mathcal{P}(\{\pi \mid s_0 = s_{init} \wedge \exists i \in [0..n], s_i \in G \wedge \forall j \in [0..i-1], s_j \notin G\})$$

Given the transition relation Tr of $\mathcal{M}$, there are two approaches to calculate $\mathcal{P}(s_{init} \models \Diamond G)$. One is solving linear equations, while the other is using value iteration. We use p_i to represent the probability from state s_i to the targets. In the following we use the example in Figure 2 to show how these two approaches work. Note that state s_4 is the only target state, denoted by double cycles.

Solving Linear Equations From the model, the transition matrix between states can be built. For example, $p_1 = 0.5 \times p_2 + 0.5 \times p_3$ and $p_0 = p_1$. Since s_4 is target, $p_4 = 1$. s_5 cannot reach target obviously, therefore $p_5 = 0$. From these equations, each p_i can be solved through matrix operations. Although this approach can get accurate result, it has drawbacks. Because each state is represented by a variable, there may be a huge number of variables in large scale systems. The state-of-the-art linear solvers are limited to handle thousands of variables, therefore linear equation approach may not be scalable.

Using Value Iterations In this approach, p_i is calculated iteratively. Assume p_i^k is an approximation of p_i after the k-th iteration. Starting from the target state s_4, in k-th iteration we update the probability of states which could reach s_4 in exactly k steps. Obviously, $\forall i \in [0, 3], p_i^0 = 0$. As $p_4^k = 1$ and $p_5^k = 0$ for any k, k is ignored in these two states. In the first iteration, p_3 can be updated, and $p_3^1 = \{0.5 \times p_2^0 + 0.5 \times p_4\} = 0.5$; in

[1] If $s_{init} \in \mathcal{D}$, then $s_{init} \in Inp(\mathcal{D})$.

the second iteration, p_1 is updated since s_1 reaches s_3 in one step. It is trivial to show $p_1^2 = \{0.5 \times p_3^1 + 0.5 \times p_2^1\} = 0.25$. In the third iteration, both p_0 and p_2 can be updated since they can reach s_1 in one step. Afterwards, p_3 is updated again because of the update of p_2. Iteratively, p_i in the long run can be calculated. A user-defined threshold is usually necessary to terminate the calculation, according to the desired precision. The result of p_i will be approximated gradually. This approach has better scalability than the linear equations method, so it is more popular in existing model checkers. However, the existence of loops may make the convergence slow. The probability of each state in SCCs will be updated many times, which means a large number of iterations may be needed before the results satisfy the terminating criteria.

2.3 States Abstraction and Gauss-Jordan Elimination

Here we follow the idea of [1]. Given a DTMC $\mathcal{M} = (S, s_{init}, Tr, AP, L)$ and a group of states $\mathcal{D} \subseteq S$, $\mathcal{D}$ can be abstracted by calculating the transition probability from $Inp(\mathcal{D})$ to $Out(\mathcal{D})$. According to the proof in [1], the abstraction of any arbitrary set of states is independent from others, and the abstract transitions **do not affect** the probability of reaching target states G.

One example of the abstraction is in Figure 3. Figure 3 (a) is the original DTMC, which has one SCC $\mathcal{D} = \{s_1, s_2, s_3\}$. $Inp(\mathcal{D}) = \{s_1\}$ and $Out(\mathcal{D}) = \{s_4, s_5\}$. In order to abstract $\mathcal{D}$[2], the probability from $Inp(\mathcal{D})$ to each state $s_{out} \in Out(\mathcal{D})$ should be calculated. Theoretically, the calculation from an SCC's inputs to outputs can be solved via linear equations or value iteration approaches[3]. However, for value iteration approach, since there could be several output states in $Out(\mathcal{D})$, we have to separately calculate the probability from input states to each output state. If there are many output states, this method could be inefficient. In addition, the existence of loops still causes slow convergence issue. Furthermore, using value iteration, there will be some errors because of the user-defined precision, but there is no way to know the error bounds. Therefore, we use a specific linear equation solving technique: Gauss-Jordan elimination [2] to do the abstraction.

Gauss-Jordan elimination is an algorithm for getting matrices in reduced row echelon form that placing zeros above and below each pivot [2]. Here, we briefly introduce how it works in our setting.

Assume there are m states in a set of states, say $\mathcal{D}$, and $|Out(\mathcal{D})| = n$. Then two matrices A and B, containing linear equations information of all transitions in $\mathcal{D}$, are first introduced as follows.

$$A(i,j) = \begin{cases} 1, & \text{if } i = j; \\ -Tr(i,j), & \text{otherwise.} \end{cases} \qquad B(i,k) = -Tr(i,k).$$

Here, A is an $m \times m$ square matrix. $A(i,j)$ is a negative value of probability of transition from i^{th} state to j^{th} state in $\mathcal{D}$ if $i \neq j$. The diagonal elements of A are filled by 1.

[2] Here we take an SCC as an example. Actually this abstraction can be applied to arbitrary set of states, according to [1].

[3] Different from our previous discussion which focuses the calculation from the initial state to targets, here we discuss the probability from input states to every output state of an SCC.

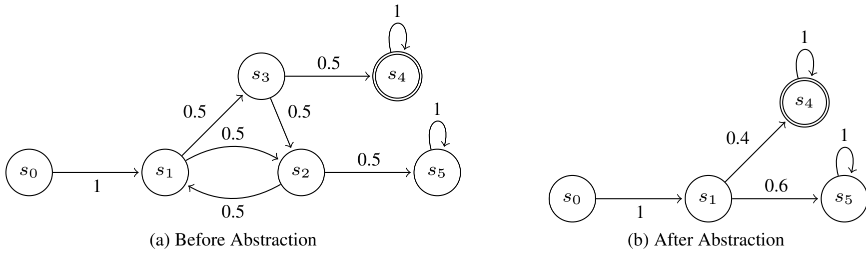

Fig. 3. States Abstraction via Gauss-Jordan Elimination

This records the transition relationship within $\mathcal{D}$. B is an $m \times n$ matrix to record the transition relationship from $\mathcal{D}$ to $Out(\mathcal{D})$. k represents the kth state in $Out(\mathcal{D})$.

Next, augmenting the square matrix A with matrix B, we will have $[A \mid B]$. Gauss-Jordan elimination on $[A \mid B]$ will then produces $[I \mid C]$. Here, I is the identity matrix with 1s on the main diagonal and 0s elsewhere. The new transition probability e.g., $Tr'(i,k)$, stores the transition probability from i^{th} state in $\mathcal{D}$ and k^{th} state in $Out(\mathcal{D})$, which is actually $-C(i,k)$. Now take Figure 3 (a) as an example. Its $[A \mid B]$ and resulting $[I \mid C]$ are listed as follows. In this example, $A(i,j)$ corresponds to $Tr(s_{i+1}, s_{j+1})$ and $B(i,k)$ indicates $Tr(s_{i+1}, s_{k+4})$.

$$[A|B] = \left[\begin{array}{ccc|cc} 1 & -0.5 & -0,5 & 0 & 0 \\ 0 & 1 & -0.5 & 0 & -0.5 \\ 0 & -0.5 & 1 & -0.5 & 0 \end{array}\right]; \ [I|C] = \left[\begin{array}{ccc|cc} 1 & 0 & 0 & -0.4 & -0.6 \\ 0 & 1 & 0 & -0.2 & -0.8 \\ 0 & 0 & 1 & -0.6 & -0.4 \end{array}\right]$$

Here the transitions from all the states in $\mathcal{D}$ to $Out(\mathcal{D})$ are obtained. Note that those states which are not in $Inp(\mathcal{D})$ will be removed. Therefore we are just interested in the new transitions from $Inp(\mathcal{D})$ to $Out(\mathcal{D})$, which are

$$Tr'(s_1, s_4) = 0.4; \quad Tr'(s_1, s_5) = 0.6;$$

We can obtain that $p_1 = 0.4 \times p_4 + 0.6 \times p_5$ in the abstracted DTMC, which is shown in Figure 3 (b). Given a group of states $\mathcal{D}$, this abstraction procedure is defined as a method $Abs(\mathcal{D})$.

Note that in practice, most transition matrices in probabilistic model checking have a very sparse structure that contains a large number of zeros. We adopt a compressed-row representation [14] as a data structure for matrices in Gauss-Jordan elimination.

3 Divide and Conquer Approach

From the analysis in Section 2, for a large DTMC with complicated loop structure, both linear equations and value iteration method are ineffective, even unworkable. In this section, we propose a divide and conquer approach which tackles the above-mentioned problem. Our main idea is similar to work [3,1], which transfers the original DTMC to an acyclic one by abstracting SCCs recursively so as to reduce the number of state and loops.

Algorithm 1. Divide and Conquer Approach

```
input : A DTMC M = (S, s_init, Tr, AP, L), target states G ⊆ S and a Bound B
output: P(s_init |= ◇G)
1  Let C be the set of all nontrivial SCCs in M;
2  while |C| > 0 do
3      Let D ∈ C;
4      if |D ≤ B| ∨ Out(D) ≤ 1 then
5          Abs(D) and C ← C\D
6      else
7          Divide D into a set of AGs denoted as A;
8          for each E ∈ A do Abs(E);
9          Let D' be the set of remaining states in D;
10         if |D'| ≤ B ∨ |D'| = |D| then
11             Abs(D') and C ← C\D
12         else
13             Let C_D' be the set of all nontrivial SCCs in D';
14             C ← (C\D) ∪ C_D';
15 return VI(M, G);
```

Intuitively, our approach divides large SCCs into smaller partitions, each of which will be solved via Gauss-Jordan elimination independently. Through this approach, loops will be eliminated. Afterwards, value iteration method is used to decide the final probability of reaching targets. In the following, we introduce our algorithm in details.

3.1 Overall Algorithm

Given a DTMC $\mathcal{M}(S, s_{init}, Tr, AP, L)$ and target states $G \subseteq S$, the probability of reaching G, denoted as $\mathcal{P}(s_{init} \models \Diamond G)$, can be solved by Algorithm 1. Note that B is an input parameter, which indicates SCCs having more than B states should be divided. $Abs(K)$ is defined in Section 2.3. $VI(\mathcal{M}, G)$ indicates calculating the probability of reaching G via value iteration. The procedure of the algorithm is explained in the following.

- The first step is to find all SCCs $\mathcal{C}$ in $\mathcal{M}$ by Tarjan's approach [17], and their input and output states are recorded as well. This is captured by Line 1.
- For each SCC $\mathcal{D} \in \mathcal{C}$, we will first check whether $|\mathcal{D}|$ exceeds B or whether $|Out(\mathcal{D})| > 1$. If not, $Abs(\mathcal{D})$ will be executed directly. States in $\mathcal{D}$ but not in $Inp(\mathcal{D})$ will be removed. Afterwards $\mathcal{D}$ will be removed from $\mathcal{C}$, as shown in Lines 4-5. The reason why we directly abstract cases $|Out(\mathcal{D})| \leq 1$ is as follows.
  - If $|Out(\mathcal{D})| = 0$, $\mathcal{D}$ has no outgoing transitions, then no matter whether $\mathcal{D}$ has target states or not, we do not need to solve $\mathcal{D}$. If $\mathcal{D} \cap G = \phi$, it is obvious that all states in $\mathcal{D}$ has probability 0 to reach G; otherwise, it is trivial to show that all states in $\mathcal{D}$ has probability 1 to reach G.

Fig. 4. Destruction of SCC during Abstraction

  - If $|Out(\mathcal{D})| = 1$, assume s_{out} is the output state. All paths entering $\mathcal{D}$ will leave it eventually. Therefore, for every $s_i \in Inp(\mathcal{D})$, the probability of paths entering $\mathcal{D}$ via s_i, staying in $\mathcal{D}$ and exiting $\mathcal{D}$ to s_{out} should be 1. So $\mathcal{D}$ can be abstracted directly.

– Lines 7-14 describe the case when $\mathcal{D}$ needs to be divided, i.e., when the SCC has more than B states. First we divide $\mathcal{D}$ into several groups based on some heuristics, each of which has a reasonably small number of state, i.e., less than B. Therefore, for each group $\mathcal{E}$ we use $Abs(\mathcal{E})$ to get the abstraction. Here we choose AG as the structure of each partition, because the existence of the root state, say s_r, may remove the most states after abstraction. In the extreme case where $Inp(\mathcal{E}) = \{s_r\}$, all states in $\mathcal{E}$ except s_r can be removed.

– By removing the states which are not input states of any $\mathcal{E}$, the number of states in $\mathcal{D}$ is often (not always) reduced. Line 10 checks two situations. 1) the size of $\mathcal{D}'$ is smaller than or equal to B, and 2) there is no reduction for $\mathcal{D}$ in this iteration. If 1) is true, then there is no need to divide $\mathcal{D}'$ again, and $Abs(\mathcal{D}')$ is executed directly. If 2) is true, i.e., no state is reduced after divide and conquer, the main reason should be that each state in $\mathcal{D}$ has a lot of pre-states. Therefore every state in one group is an input state and cannot be removed. In this case, $\mathcal{D}'$ should also be abstracted. Afterwards, $\mathcal{D}$ is removed from $\mathcal{C}$. If 1) and 2) are both false, Lines 13-14 will be executed.

– Because of the abstraction, $\mathcal{D}$ may not be an SCC now. An example is shown in Figure 4. On the left hand side, $\mathcal{D} = \{s_1, s_2, s_3\}$; if we group s_1 and s_2 together, then s_3 is this group's output. It is easy to get the abstract transitions between them, as shown in right hand side. Because both s_1 and s_2 are input states, no state is removed. However, it is obvious that $\mathcal{D}' = \{s_1, s_2, s_3\}$ is not an SCC anymore. Tarjan's algorithm is used again to find new SCCs in the $\mathcal{D}'$, captured by Line 13. New SCCs will be added to $\mathcal{C}$ for another iteration.

– When the iteration terminates, there is only trivial SCCs in $\mathcal{M}$ now; in other words, $\mathcal{M}$ is acyclic. Value iteration approach can be used to calculate the probability from the initial state to targets efficiently, and this is captured by Line 15.

As we mentioned in Section 2.3, the iterative abstraction will not affect the final result of the probability calculation. The following theorem establishes that the algorithm is always terminating.

Theorem 1. *Given a finite state DTMC $\mathcal{M}$, Algorithm 1 always terminates.*

Proof. We assume $\hat{S} = \Sigma_{\mathcal{D} \in \mathcal{C}} |\mathcal{D}|$, in other words, $\hat{S}$ is the total number of states in $\mathcal{C}$. Then the theorem can be proved by showing (1) $\hat{S}$ is finite at the beginning, and (2) $\hat{S}$ monotonically decreases after each iteration.

(1) is obviously true because $\mathcal{M}$ has finite number of states, and $\hat{S} \leq |S|$ where S is the set of states of $\mathcal{M}$.

Given an SCC $\mathcal{D} \in \mathcal{C}$, if it satisfies the condition in Line 4, then $\mathcal{D}$ will be removed from $\mathcal{C}$, thus $\hat{S}$ is reduced. Otherwise, from Line 6, there are two possible outputs. (i) $\exists \mathcal{E} \in \mathcal{A}$, $Abs(\mathcal{E})$ reduces its number of states, or (ii) $\forall \mathcal{E} \in \mathcal{A}$, $Abs(\mathcal{E})$ does not reduce its number of states. If (i) is true, then $\hat{S}$ is also reduced. If (ii) is true, then $|\mathcal{D}'| = |\mathcal{D}|$. According to Line 8, $\mathcal{D}$ will be abstracted directly and be removed from $\mathcal{C}$. Thus $\hat{S}$ is still reduced. Therefore (2) is true, and the theorem holds. □

3.2 Dividing Strategies

Although the divide-and-conquer approach is correct and terminating, its efficiency is highly dependent on how an SCC is divided. Assume $\mathcal{A}$ is the set of partitions after dividing an SCC, then a suitable partition, say $\mathcal{E} \in \mathcal{A}$, should satisfy the following conditions.

1. $\mathcal{E}$ should not have too many states, since each partition is abstracted using Gauss-Jordan elimination which is limited to a relatively small number of states;
2. $\mathcal{E}$ should not have too few states as well, otherwise there will be too many partitions to be solved, and the states reduction for $\mathcal{E}$ is inefficient;
3. The smaller $|Out(\mathcal{E})|$ is, the better reduction is achieved. Too many output states will make the input states of $\mathcal{E}$ have too many abstract transitions, which makes the remaining structure complicated, and affects the efficiency of the following abstraction.

As a result, the remaining issue is that given an SCC $\mathcal{D}$, is there any *optimal* strategy to divide it into *suitable* AGs? In practice, the structure of $\mathcal{D}$ could be arbitrary. This increases the difficulty of finding a general strategy for all cases.

The simplest division method is to try to set each AG to have the same number of states. Assume each AG should have N states. Then starting from one input state of $\mathcal{D}$, depth first search (DFS) or breadth first search (BFS) can be used to group every N states together. Afterwards, each AG can be abstracted, and the remaining states are combined together to do the next iteration. The advantage of this strategy is that the number of states in each partition is easily controlled. It can be very efficient in cases where the states in $\mathcal{D}$ has few transitions. However, this method cannot control the number of output states of each partition, and a predefined N may not be suitable for $\mathcal{D}$'s structure.

Therefore, another improved strategy is used to automatically decide the number of states in each AG. Instead of picking a constant N in the beginning, we set a lower bound B_L and an upper bound B_U for each partition. Thus the number of states in each partition should be between B_L and B_U. At first, B_L states will be grouped into $\mathcal{E}$, and $|Out(\mathcal{E})|$ is recorded. Afterwards, some states in $Out(\mathcal{E})$ are added into $\mathcal{E}$, and $|Out(\mathcal{E})|$ is updated. If $|Out(\mathcal{E})|$ keeps unchanged or even becomes smaller after the update, we will try to add more states into $\mathcal{E}$ again. If $|Out(\mathcal{E})|$ is increased but the

increase is not significant, a few states will be added into $\mathcal{E}$ but the number should be small. Otherwise $\mathcal{E}$ is confirmed and ready for $Abs(\mathcal{E})$. Note the number of states in $\mathcal{E}$ should be always below B_U. This strategy guarantees

1. the number of states in $\mathcal{E}$ is under control. B_L and B_U guarantee that the size of $\mathcal{E}$ should not be too large or too small.
2. the outputs of $\mathcal{E}$ are also manageable. This guarantees the states structure after abstraction is not too complicated, and is suitable for next iteration.

Parameters B, N, B_L and B_U can be adjusted according to the specific DTMC to get the optimal efficiency.

3.3 Parallel Computation

Previous work such as [6,13] depends on the topological order between different SCCs. Therefore, parallel computation is not so easy to use in their setting. On the contrary, our algorithm eliminates loops via abstracting every SCC one by one, without considering their order. The independence between different SCCs can be proved following the proof in [1]. What is more, even each AG in one SCC is also independent from others, and the proof actually follows the same idea of SCC's independence. Thus, parallelization is suitable in our setting in order to solve different AGs simultaneously.

In details, after finding all SCCs, they are stored with their input and output states. For each SCC, a spare thread can be used to solve it. Therefore, Lines 2-14 in Algorithm 1 can be solved via parallel computation. In addition, whenever an AG is grouped, another spare thread, if there is any, can be used to abstract it. Thus Line 8 in Algorithm 1 can also be handled in parallel.

4 Implementation and Evaluation

We have implemented the algorithm into our model checking framework PAT [15], which supports explicit probabilistic model checking [16] and can be freely downloaded at `http://www.patroot.com`.

In the following, several experiments are conducted to show the efficiency of our new approach. Note that we show the improvement via comparing to PAT itself, which was based on value iteration method previously. Since the only difference between these two versions is the algorithm of reachability analysis, it is fair to check the effectiveness of the new method. Besides, several cases used in our experiment have dynamically updated probabilistic distributions, and the modeling of them by other model checkers is highly nontrivial.

In these experiments, we use the **improved** dividing strategy, and B, B_L, B_U are set to be 300, 100, 150 respectively. In other words, an SCC with more than 300 states should be divided; each group has states between 100 and 150. These parameters are manually selected based on our experimental experience, i.e., generally these parameters have better performance compared with others. The testbed is a server running Windows Server 2008 64 Bit with Intel Xeon 4-Core CPU$\times$2 and 32 GB memory.

First, we use a simple example to show that our approach gets accurate results, resolves the slow convergence problem and results in huge speedup. Assume there are

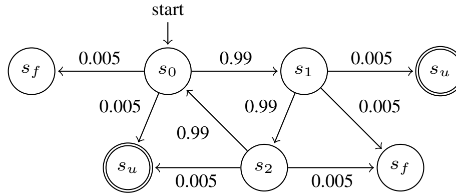

Fig. 5. A Simple Example: N = 3. s_u and s_f are copied for better demonstration.

Table 1. Experiments: A Simple Example

System	PAT (w)			PAT (w/o)		
	Prob	Time (s)	Memory (MB)	Prob	Time (s)	Memory (MB)
N = 500	0.5	0.03	71	0.49987	0.5	24
N = 5000	0.5	0.3	83	0.49987	5.5	63
N = 50000	0.5	2.6	151	0.49987	125.2	111
N = 500000	0.5	29.7	885	0.49987	1612.8	838

$N + 2$ states $\{s_0, s_1, ..., s_{N-1}, s_u, s_f\}$ existing in this example. Each state $s_i, i \in [0..N-1]$, has probability 0.99 to reach $s_{(i+1)\%n}$, and also has probability 0.005 to reach s_u and s_f separately. The case $N = 3$ is shown in Figure 5. Obviously, all states $s_i, i \in [0..N-1]$ compose an SCC, and s_u and s_f are this SCC's outputs. We check the probability from s_0 to s_u, and several experiments are executed based on different value of N as listed in Table 1.

In Table 1, columns $Prob$ represents the probability returned by the model checking algorithms. Columns $PAT(w)$ ($PAT(w/o)$) show the experimental information taken with (without) the new approach. Columns $Time$ represent the total time cost in the verification. For these cases, our new approach outperforms value iteration approach dramatically by reducing the verification time to less than 10%. On the other hand, the memory used in new approach is higher than that used in the previous method, which is reasonable since solving linear equations consumes more memory than value iteration approach. Through the manual analysis, we know that 0.5 is the accurate result while 0.4998 is only an approximation.

Next, we apply our approach to several more meaningful systems and demonstrate that our approach can still improve the efficiency significantly.

In multi-agent systems, dispersion games [8] represent an important scenario, i.e., dispersion games are the generalization of anti-coordination games to an arbitrary number of players and actions. Here we use two strategies designed for dispersion games: *bisic simple strategy* (BSS) and *extend simple strategy* (ESS). BSS assumes the number of players and the number of actions are the same, while ESS does not have this assumption. In each round of the game, every player chooses one action following specific probabilistic distribution, which is updated roundly according to the output of last round. There is a desired outcome in this game called Maximal Dispersion Outcome (MDO), and one property is to calculate the probability that MDO can be achieved.

Another case used in our experiments is coin flipping protocol for polynomial randomized consensus [4] (CS). This case focuses on modeling and verifying the shared

Table 2. Experiments: Benchmark Systems

System	States	Prob	PAT (w)			PAT (w/o)		
			Time (s)	BMR	Memory (MB)	Time (s)	BMR	Memory (MB)
BSS (4)	4196	1	1.3	92.3%	39	0.2	50%	35
BSS (5)	49572	1	3.5	94.3%	297	4.4	11.4%	142
BSS (6)	605890	1	41.4	72.7%	1297	105.3	6.7%	417
BSS (7)	7462639	1	1671	30.1%	6350	2073.1	4.1%	5039
ESS (6, 4)	32662	1	1.4	92.8%	16.3	2.7	14.8%	5.6
ESS (6, 5)	162945	1	6.7	91.1%	48.5	11.4	16.7%	13.9
ESS (7, 5)	463460	1	27.9	84.9%	310	75.8	7.1%	292
ESS (8, 5)	1114480	1	70.5	74.7%	619	278.5	6.1%	643
ESS (8, 6)	6476524	1	438.0	68.5%	4209	1168.1	7.5%	3904
CS (4, 3)	4966	0.023	0.8	87.5%	45	2.4	8.3%	35
CS (6, 3)	34529	0.023	15.7	81.5%	214	124.1	0.9%	108
CS (6, 4)	45281	0.015	24.8	86.7%	324	243.8	0.6%	81
CS (6, 5)	56033	0.012	38.6	91.2%	312	432.1	0.4%	104
CS (7, 4)	99265	0.014	102.3	87.6%	1062	983.1	0.4%	97
CS (7, 5)	122785	0.011	161.7	92.1%	1145	1384.8	0.3%	97
CS (7, 6)	146305	0.01	245.5	94.9%	1404	2409.5	0.2%	156
CS (8, 4)	200083	0.013	585.1	93.4%	1974	-	-	-

coin protocol of the randomized consensus algorithm. CS is used as a benchmark system in the state-of-the-art probabilistic model checker PRISM [12]. Here we use a safety property in the system as our target.

The experiments based on these three models are listed in Table 2. $BSS(N)$ indicates there are N players (also N actions) in the game; $ESS(N, K)$ means there are N players and K actions; $CS(N, K)$ indicates there are N processes and K is a constant used in the model. Here we are interested in the ratio of model building (*BM*) time to the total time, which is denoted as *BMR* in the table. In $PAT(w)$, *BM* means the time for building **acyclic** DTMC, i.e., the overall time consumed by eliminating loops in DTMC; in $PAT(w/o)$, it indicates the time for building the whole system. In both PAT versions, value iteration is used to get the final result after building the model. '-' indicates the verification takes more than 1 hour thus the result is not taken into consideration. From the table, we have several observations.

1. For some small examples such as $BSS(4)$, our new approach is slower. This is due to the overhead taken by the SCC searching algorithm, and value iteration approach is efficient when loops are small.
2. As the examples become larger, the verification speed is increased by our proposed approach. This improvement is obvious especially in large-scale systems such as $ESS(8, 5)$, $ESS(8, 6)$ and $CS(8, 4)$.
3. CS consumes more resource than BSS and ESS when they have similar size of state space, such as $CS(7, 6)$ and $ESS(6, 5)$. The reason is that CS has more complicated SCCs, and both our new approach and traditional value iteration method have to use more time and memory to solve it. As a result, the SCCs' structure affects the verification efficiency to a large extent.

4. According to *BMR*, we can see that in the previous version of PAT, building the model costs small portion of the overall verification time compared with the value iteration procedure. The average value of *BMR* is less than 10%, which means slow convergence indeed exists in systems having large SCCs. CS has very small *BMR* and this is consistent with the fact that CS has complicated SCCs. In the new approach, time is mainly used by abstractions, as average *BMR* is more than 80%. It indicates that the efficiency of the divide and conquer strategy is critical in the whole verification now, and optimal dividing strategy is worthy to explore.

On the other hand, we want to share some limitations of our approach according to the experimental information. The efficiency of this approach is dependent on whether large SCCs exist in the system. During our experiment, the new approach performs slower than value iteration method in several cases. The main two reasons include 1) there is no loops in the system, thus the SCC searching algorithm makes the whole verification slow; 2) the system just has small SCCs while the whole state space is large, thus the gain of the abstraction is limited.

5 Related Work and Conclusion

SCCs are an important structure in both concurrent and probabilistic verification. For probability calculation, those loops in SCCs are one of the key factors affecting the efficiency. Some previous work has been done based on SCC decomposition for probabilistic systems, including DTMCs and Markov Decision Processes (MDPs) [5], and we are mainly inspired by this work.

To speed up the verification of MDP, the authors of [6] have proposed to decide the topological order of all SCCs in the MDP, and value iteration method is used to solve the SCCs from the bottom upwards. Based on this work, the authors of [13] have used SCC decomposition to handle the incremental quantitative verification of MDP. The topological order between SCCs guarantees that some changes in one SCC will not affect those SCCs after it. Compared to their work, ours does not consider the orders of SCCs via treating each SCC independently. This makes parallel computation approach feasible. In addition, Gaussian-Jordan elimination is used to remove loops. Different from value iteration, which needs a user defined precision, our approach generates accurate result.

Besides, there are several work based on SCC focusing on probabilistic counterexample generation, such as [3,1]. Their idea of abstracting each SCC from its input to output is the biggest inspiration of our work. Compared with these work, ours is more focusing on improving reachability analysis in DTMC. Therefore, we divide SCCs into smaller partitions and solve them directly.

Conclusion. In this work, we proposed a divide-and-conquer approach to speed up reachability analysis of DTMCs. Because SCCs are one of main reasons that the probability calculation is slow, we focus on abstracting SCCs via calculating the transition probability from their inputs to outputs. We divide every SCC, whose states exceed some specific bound, into several AGs having reasonable number of states, and can be solved efficiently via Gauss-Jordan elimination. We have implemented our approach in PAT, and some benchmark systems are used to show its effectiveness and efficiency.

For future work, there are two possible directions. Currently, the parameters used in the algorithm such as B, B_L and B_U are mainly decided via experience, and are manually defined before the experiments. Therefore, one topic is to find the more efficient division strategies, which are automatic and suitable for general cases. Another direction is extending our approach to MDP. Concurrency also exists in many probabilistic systems, so nondeterminism is unavoidable in some cases. SCCs in MDP can also be eliminated via calculating the probability distributions from inputs to outputs. Due to the nondeterminism in MDP, one challenge is that the number of resulting distributions may be exponential, thus a suitable divide and conquer approach for MDP is needed.

References

1. Ábrahám, E., Jansen, N., Wimmer, R., Katoen, J.-P., Becker, B.: DTMC Model Checking by SCC Reduction. In: QEST, pp. 37–46 (2010)
2. Althoen, S.C., McLaughlin, R.: Gauss - Jordan reduction: a brief history. The American Mathematical Monthly 94(2), 130–142 (1987)
3. Andrés, M.E., D'Argenio, P., van Rossum, P.: Significant Diagnostic Counterexamples in Probabilistic Model Checking. In: Chockler, H., Hu, A.J. (eds.) HVC 2008. LNCS, vol. 5394, pp. 129–148. Springer, Heidelberg (2009)
4. Aspnes, J., Herlihy, M.: Fast Randomized Consensus Using Shared Memory. Journal of Algorithms 15(1), 441–460 (1990)
5. Baier, C., Katoen, J.: Principles of Model Checking. The MIT Press (2008)
6. Ciesinski, F., Baier, C., Größer, M., Klein, J.: Reduction Techniques for Model Checking Markov Decision Processes. In: QEST, pp. 45–54 (2008)
7. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press (1999)
8. Grenager, T., Powers, R., Shoham, Y.: Dispersion Games: General Definitions and Some Specific Learning Results. In: AAAI, pp. 398–403 (2002)
9. Itai, A., Rodeh, M.: Symmetry Breaking in Distributed Networks. Information and Computation 88, 150–158 (1981)
10. Katoen, J.-P., Khattri, M., Zapreev, I.S.: A Markov Reward Model Checker. In: QEST, pp. 243–244 (2005)
11. Katoen, J.-P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The Ins and Outs of The Probabilistic Model Checker MRMC. In: QEST, pp. 167–176 (2009)
12. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of Probabilistic Real-Time Systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)
13. Kwiatkowska, M.Z., Parker, D., Qu, H.: Incremental Quantitative Verification for Markov Decision Processes. In: DSN, pp. 359–370 (2011)
14. Stoer, J., Bulirsch, R.: Introduction to Numerical Analysis. Springer, Berlin (2002)
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. Sun, J., Song, S., Liu, Y.: Model Checking Hierarchical Probabilistic Systems. In: Dong, J.S., Zhu, H. (eds.) ICFEM 2010. LNCS, vol. 6447, pp. 388–403. Springer, Heidelberg (2010)
17. Tarjan, R.E.: Depth-First Search and Linear Graph Algorithms. SIAM J. Comput. 1(2), 146–160 (1972)
18. Younes, H.L.S., Clarke, E.M., Zuliani, P.: Statistical Verification of Probabilistic Properties with Unbounded Until. In: Davies, J. (ed.) SBMF 2010. LNCS, vol. 6527, pp. 144–160. Springer, Heidelberg (2011)