Journal of Computer and Knowledge Engineering, Vol. 3, No. 2, 2020. (31-36)
DOI: 10.22067/cke.2020.39288

Accelerating Iterative Methods for Bounded Reachability Probabilities in Markov Decision Processes[*]

Research Article

Mohammadsadegh Mohagheghi [1]

Abstract: Probabilistic model checking is a formal method for verification of the quantitative and qualitative properties of computer systems with stochastic behaviors. Markov Decision Processes (MDPs) are well-known formalisms for modeling this class of systems. Bounded reachability probabilities are an important class of properties that are computed in probabilistic model checking. Iterative numerical computations are used for this class of properties. A significant draw-back of the standard iterative methods is the redundant computations that do not affect the final results of the computations, but increase the running time of the computations. The study proposes two new approaches to avoid redundant computations for bounded reachability analysis. The general idea of these approaches is to identify and avoid useless numerical computations in iterative methods for computing bounded reachability probabilities.

Keywords: Probabilistic model checking, Iterative numerical methods, Bounded Reachability probabilities, Markov decision processes.

1. Introduction

Model checking is a well-known formal approach for verifying quantitative and qualitative properties of computer systems. In this way, the system is modeled by a labeled transition system and the properties of the system are specified in temporal logic. Because of some stochastic behaviors of the computer systems, the probabilistic alternative of transitions systems are used for modeling the underlying system and probabilistic model checking is used to analyze the quantitative property specifications of these systems [1&2]. Markov chains and Markov Decision Processes (MDPs) are widely used as probabilistic transition systems [13]. On the other hand, probabilistic reachabilities (also called reachability probabilities) and expected rewards are considered to specify the desired properties of the system [3&4]. In probabilistic reachability, the maximum of minimum probability of reaching a goal state should be computed [20]. Probabilistic reachability can be bounded or unbounded. In bounded reachability, the number of steps is limited but in unbounded reachability the number of steps of the system is not limited to any bound [11].

Iterative numerical methods are the standard approaches to compute the bounded or unbounded probabilistic reachability. These methods start from an initial vector of values and iteratively updates the reachability probabilities according to the computed values from the previous iteration [6&18]. A main drawback of the iterative methods for probabilistic reachability is the redundant (useless) computations, that is, those computations that do not affect the computed value of a state [3&15]. Several approaches have been proposed in other works to avoid the useless computations and accelerate the iterative computations [5, 9, 14, 16, & 21]. In most cases, the proposed approaches are used for computing unbounded reachability probabilities and do not show promising improvements for bounded properties. The study focuses on the bounded probabilistic reachabilities and proposes two new techniques to accelerate the standard iterative method for these properties. The main contributions of this work are as follows:

- A technique is proposed that considers the set of transitions that lead to a Dirac distribution (a transition with probability one) to improve the performance of the iterative computations. This technique directly uses the related reachability probability to avoid redundant computations. In this case, the proposed technique divides the transitions of a model into Dirac and non-Dirac ones. For Dirac transition, the method only needs to consider the value of the destination state and can avoid the multiplications that are performed in the standard approach.
- A new approach is proposed to avoid useless updates in each iteration. In this case, the value of a state in iteration should be updated only if the value of some successors of the state has been updated in the previous iteration. Otherwise, the method avoids the update and its related computations.

The two techniques are considered as an improved method for computing bounded reachability probabilities. The experiments of the study show more than 50% improvement in the running time when the improved method is used.

The remainder of this paper is structured as follows: Section 2 reviews some related definitions and methods. In section 3, the techniques for accelerating the standard iterative method for bounded reachability probabilities are proposed. Section 4 proposes the experimental results and Section 5 concludes the paper.

2. Review of Related Definitions and Methods

In this section the definitions and standard algorithm for bounded reachability probabilities are reviewed [2, 8].

2.1. Definition 1 (MarkovDecision Process)

A Markov Decision Process (MDP) is a tupple $M = (S, s_0, Act, \delta, G)$ where S is a finite set of states, $s_0 \in S$ is an initial state, Act is a finite set of actions, $\delta: S \times Act \rightarrow Dist(S)$ is a probabilistic transition function and $G \subset S$ is the set of goal states. The size of M which is shown as $|M|$ is defined as the number of states of M plus the number of its transitions. For every state $s \in S$ of an MDP M, one or more

* Manuscript received Jun, 5; 2019, accepted. October, 30, 2020.

[1] Assistant Professor, Department of Computer science, Vali-e-Asr University of Rafsanjan, Rafsanjan, Iran.
Email: mohagheghi@vru.ac.ir

actions of Act are defined as enabled actions. This set is defined as $Act(s) = \{\alpha \in Act | \delta(s,\alpha) \text{ is defined}\}$. For a state $s \in S$ and $\alpha \in Act(s)$ we use $Post(s,\alpha)$ for the set of a successor states of s, $Post(s)$ for all possible successor states of s and $Pre(s)$ for possible predecessor states of s [2]:

$$Post(s,\alpha) = \{s' \in S \mid \delta(s,\alpha,s') > 0\}, \quad (1)$$
$$Post(s) = \cup_{\alpha \in Act(s)} Post(s,\alpha), \quad (2)$$
$$Pre(s) = \{s' \in S \mid s \in Post(s)\}, \quad (3)$$

The semantic of an MDP is defined as its possible transtions and paths. Two steps are considered to take a transition from a state $s \in S$. First, one enabled action $\alpha \in Act(s)$ is selected non-deterministically. Second, using the probability distribution $\delta(s,\alpha)$, a successor state s' is chosen randomly. Note that $\delta(s,\alpha)(s')$ determines the probability of a transition from s to s' by the action $\alpha \in Act(s)$. A transition (s,α,s') is called Dirac transition if $\delta(s,\alpha)(s') = 1$. A discrete-time Markov chain (DTMC) is an MDP for which every state has exactly one enabled action [17, 18, & 23]. A finite or infinite path in M is a non-empty sequence of the form $\pi = s_0 \overset{a_0}{\rightarrow} s_1 \overset{a_1}{\rightarrow} \dots$ where $s_i \in S$ and $a_i \in Act(s_i)$ and $s_{i+1} \in Post(s_i, a_i)$ for every $i \geq 0$. $Path_s$ is used to denote the set of all infinite paths of M that start in a state s and $FPath_s$ for all finite paths of M. $\pi[i]$ is used to denote the $(i+1)-th$ state in the path π, that is, $\pi[i] = s_i$.

A policy is a function that maps each state $s \in S$ to one of its enabled actions $\alpha \in Act(s)$. Policies are used to resolve the non-deterministic selections of MDPs [8]. Depending on the selected actions of an MDP, a probability measure is defined on the set of related paths. More details about policies and their induced probability measure are available in [2, 13, 8, & 11].

For bounded reachabilities and for any state $s \in S$, it is normal to define $reach_s^{\leq k}(G)$ as the set of all paths that start from s and reach a state in G within at most k steps:

$$reach_s^{\leq k}(G) = \{\pi \in Paths_s | \pi[i] \in G \; for \; some \; i \leq k\}$$

$Pr_{max}(reach_s^{\leq k}(G))$ is defined as the maximal probability of reaching to one state in G in at most k iterations. The maximal (or minimal) probability is defined over the set of policies of the model [2&8].

2.1. Iterative Methods for Probabilistic Model Checking

The aim of probabilistic model checking is to verify a desired quantitative or qualitative property of a system with stochastic behavior, which is modeled by an MDP. To this end, a probabilistic extension of transition systems is used to model the system and probabilistic temporal logics are used to specify the desired properties of the system [1&2]. In most cases, verifiation of the proposed properties is reduced to the computation of the optimal probability of reaching or the optimal expected cost before reaching a goal state [2, 6, & 8].

The standard approach for computing the optimal bounded or unbounded reachability probabilities is to use a vector $\bar{x}$ of reachability probability values and update the vector iteratively [4]. For a state $s \in S$, let x_s^k denote the maximum probability of reaching at least one state of G within at most k steps, that is, $x_s^k = \Pr_{max}(reach_s^{\leq k}(G))$. An iterative method computes the bounded reachability by computing the value of each state $s \in S$ according to the value of the states in $Post(s)$. For initiating $\bar{x}$, we set $x_s^0 = 1$ if $s \in G$ and $x_s^0 = 0$ otherwise. For any iteration $k > 0$ se have: so we have:

$$x_s^k = \begin{cases} 1 & if \; s \in G \\ \max\limits_{\alpha \in Act(s)} \sum\limits_{s' \in S} \delta(s,\alpha)(s').x_{s'}^{k-1} & if \; s \notin G \end{cases}$$

Note that in this computations, the policies are considered implicitly, that is, for each state $s \in S$, the method selects the action that maximizes the reachability probability. In practice, a model checker does not need to store all vectors. It only needs a vector for the current iteration and another for the previous iteration [18]. Algorithm 1 presents the standard iterative method to compute $\Pr_{max}(reach_s^{\leq k}(G))$ [4].

The algorithm uses two vectors $\bar{x}$ and $\overline{x'}$ to store the reachability probabilities of each state. In each iteration, it first stores the computed values of all states in the vector $\overline{x'}$ and after the computation of all values, it updates the vector $\bar{x}$. The algorithm performs k iterations and in each iteration, updates the value of all state.

Algorithm 1 Bounded value iteration for $Pr_{max}^M(reach_s^k(G))$.

```
1: input: an MDP $M = (S, \hat{s}, Act, \delta, G)$ and a bound $k$
2: output: $Pr^(_{max} reach_s^k(G))$ for all $s \in S$
3: for all $s \in S$ do
4:     $x_s \leftarrow \begin{cases} 1, & \text{if } s \in G, \\ otherwise \end{cases}$
5: end for
6: for i = 1 to k do
7:     for all $s \in S \backslash G$ do
8:         $x'_s \leftarrow \max_{\alpha \in Act(s)} \sum_{s' \in Post(s)} \delta(s, \alpha(s))(s') \times x_{s'}$;
9:     end for
10:    for all $s \in S \backslash G$ do
11:        $x_s \leftarrow x'_s$;
12:    end for
13: end for
14: return $(x_{s_i})_{s_i \in S}$;
```

3. Avoiding Redundant Computations for Bounded Reachability Probabilities

In most model checkers, the standard iterative method (Algorithm 1) is used to compute the bounded reachability probabilities [10, 12, & 17]. One drawback of this method is that it considers all states of the model in every iteration. A learning based method has been proposed in [4] for accelerating the computations of bounded reachability probabilities, but it is more useful for small values of the bound k. Two new techniques are proposed to accelerate the iterative computations for bounded reachability probabilities. The approaches outperform the standard method (Algorithm 1) and the learning-based method of [4]. The idea of these approaches is to identify and avoid *useless* updates and *redundant* multiplications in each iteration.

3.1. Avoiding Redundant Multiplications for Dirac Transitions

Algorithm 1 performs at least one multiplication for each action $\alpha \in Act(s)$ (line 8). However, if there is only one state $s \in Post(s, \alpha)$ the method can directly use the value of s' because (s, α, s') is a Dirac transition, that is, $\delta(s, \alpha)(s') = 1$. To use this idea in the proposed method, the set Act of actions of M are separated to *DiracAct* and *nonDiracAct* sets:

$$DiracAct = \{\alpha \in Act | \exists s, s' \in S, \delta(s, \alpha)(s') = 1\}$$
$$nonDiracAct = \frac{Act}{DiracAct}$$

The proposed approach is to use each set of actions separately. For each state $s \in S$, the method first considers those transitions that are of the form (s, α, s'), where $\alpha \in DiracAct(s)$. For such transitions (which are a main part of transitions in most cases) the method considers the value of s' from the previous iteration to update the value of s in the current iteration. In this case, the method does not perform any multiplication. The intuition behind this approach is to reduce the total number of multiplications as the main time consuming operation in computations. For non-Dirac actions, the method uses the standard approach (the same as lines 5-7 of Algorithm 1).

3.2. Avoiding Useless Updates

An update of a state $s \in S$ in an iterative method is useless if it does not change the value of s. To avoid useless updates, in each iteration, the proposed method (explained in Algorithm 2). If the value of any state $s \in S$ is changed in an iteration k, the method marks the states in $Pre(s)$ as enabled for iteration $k + 1$. To have an efficient access to the members of $Pre(s)$, these states should be stored for each state $s \in S$. However, the method only needs the list of predecessor states of each state and do not need to restore their actions and probabilities. In this case, the memory overhead reduces to the number of states of the model.

Algorithm 2 shows the proposed approach for accelerating bounded reachability probabilities. For the first iteration, the algorithm marks those states $s \in S$ that have at least one transition to a state in G. In each iteration $i \leq k$and for each marked state $s \in S$, the algorithm first considers the set of actions $\alpha \in DiracAct(s)$ and uses the value of the state $s' \in Post(s, \alpha)$ (lines 15-20). For simplicity, $s' = Post(s, \alpha)$ is used because $Post(s, \alpha)$ has only one member. It next considers non-Dirac actions. For each of these actions the algorithm computes the related reachability probability in line 22 and compares the computed value with the maximum previous one (which is stored in d). For each state s, if the computed value is more than its previous one (that is checked in line 27), the new value is stored in x'_s and the algorithm marks the states in $Pre(s)$ as enabled for the next iteration (line 29).

Algorithm 2 Improved Bounded value iteration for $Pr^M_{max}(reach^k_s(G))$.

```
1: input: an MDP $M = (S, \hat{s}, Act, \delta, G)$ and a bound $n$
2: output: $Pr^M_{max}(reach^k_s(G))$ for all $s \in S$
3: for all $s \in G$ do
4:     for all $s' \in$ Pre(s) do
5:         Mark $s'$ as enabled for the first iteration
6:     end for
7: end for
8: for all $s \in S$ do
9:     $x_s \leftarrow \begin{cases} line: ps_I nit1, & \text{if } s \in G0, \\ otherwise \end{cases}$
10: end for
11: for i = 1 to k do
12:     for all $s \in S \setminus G$ do
13:         if $s$ is marked as enabled for iteration $i$ then
14:             $d = 0$;
15:             for all $\alpha \in DiracAct(s)$ do
16:                 Let $s' = Post(s, \alpha)$
17:                 if $x_{s'} > d$ then
18:                     $d = x_{s'}$
19:                 end if
20:             end for
21:             for all $\alpha \in nonDiracAct(s)$ do
22:                 $d' = \sum_{s' \in Post(s,\alpha)} P(s, \alpha)(s').x_{s'}$
23:                 if $d' > d$ then
24:                     $d = d'$
25:                 end if
26:             end for
27:             if $d > x_s$ then
28:                 $x'_s = d$
29:                 Mark every $s' \in Pre(s)$ as enabled for iteration $i + 1$
30:             else
31:                 $x'_s = x_s$
32:             end if
33:             for all $s \in S \setminus G$ do
34:                 $x_s = x'_s$
35:             end for
36:         end if
37:     end for
38: end for
39: return $(x_{s_i})_{s_i \in S}$;
```

Table 1. Description of case study models

Model (Parameters)	Parameter Value(s)	Number of States	Number of Actions	Number of Transitions	Number of Dirac Transitions
Consensus (n, k)	5, 15	1,179K	3,944K	4,927K	2,961K
	5, 32	2,496K	8,350K	10,435K	6,266K
	6, 8	4,612K	18,445K	23,032K	13,857K
	6, 20	11,322K	45,318K	45,318K	34,012K
leader (K)	7	2,095K	6,729K	7,714K	5,745K
	8	18,674K	67,761K	77,708K	57,815K
phil_lss (K)	40	3,574K	11,689K	13,122K	10,256K
	60	8,115K	26,817K	30,106K	23,527K
	80	14,491K	48,139K	54,045K	42,232K

4. Implementation and Experimental Results

The proposed approaches have been implemented as a package in the PRISM model checker. This implementation is based on the sparse engine of PRISM [17] which is developed in C and JAVA. Three classes of standard case studies are considered which have been used in previous works [3, 7, 8, 16, 18, 20, & 21]. The *Consensus*, *leader* and *phil_lss* case studies are used to compare the performance of the proposed methods for bounded reachabilities. Table 1 shows the common characteristics of case study models. More information about theses case studies are available in [8&21]. Table 1 shows the name of the model, the parameter values, the number of states, and total number of actions of each model. In addition, the total number of transitions and number of Dirac transitions are proposed in the last two columns. K is used for 10^3. We see that more than 50% of transitions of each model are Dirac transitions.

The running time of the iterative method with the proposed approaches (which is called the improved method in this section), are compared with the running time of the standard iterative method and the learning based method from [4]. All benchmarks have been run on a machine with Corei7 CPU (2.8 GHz, 4 main cores) and 8GB RAM running Ubuntu 16. Figure 1 to Figure 9 show the results of the experiments. Three values for the bound of iterations are considered, where are shown in the horizontal axis of figures. The vertical axis shows the running times in second.

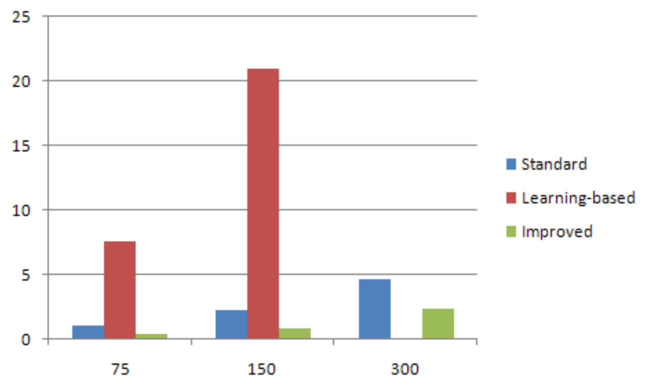

Figure 1. Running times for consensus_5_16

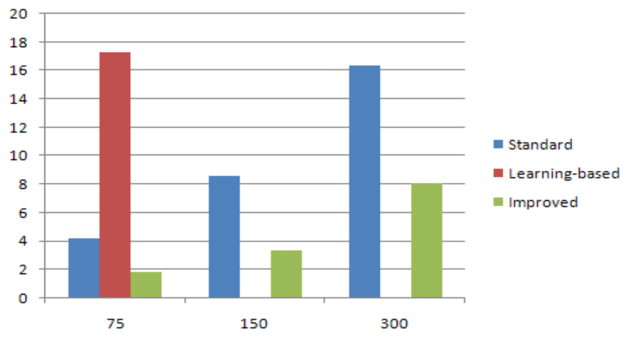

Figure 2. Running times for consensus_5_32

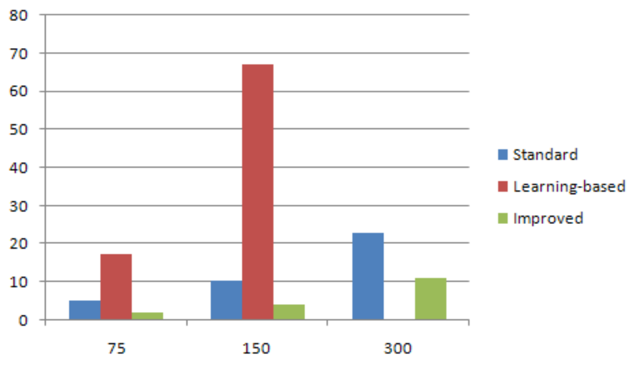

Figure 3. Running times for consensus_6_12

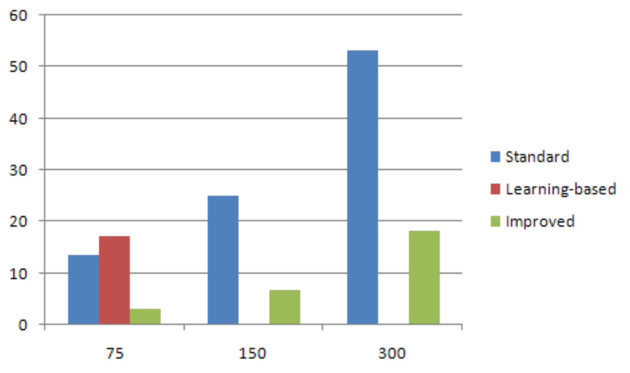

Figure 4. Running times for consensus_6_20

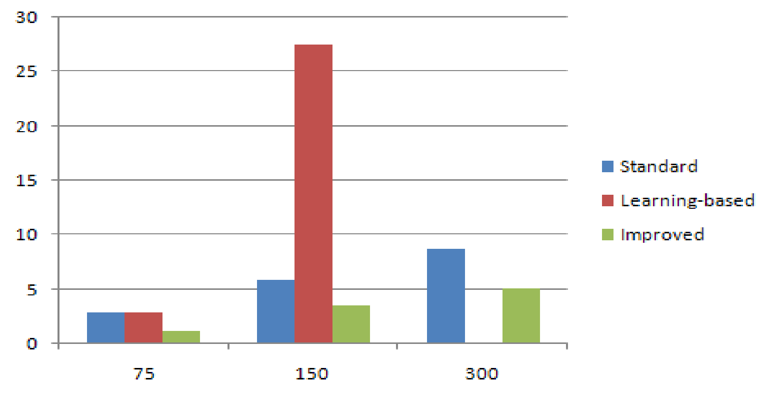

Figure 5. Running times for leader_7

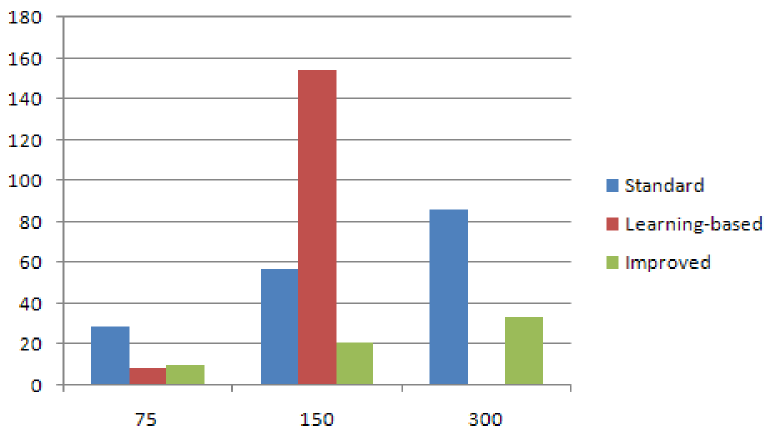

Figure 6. Running times for leader_8

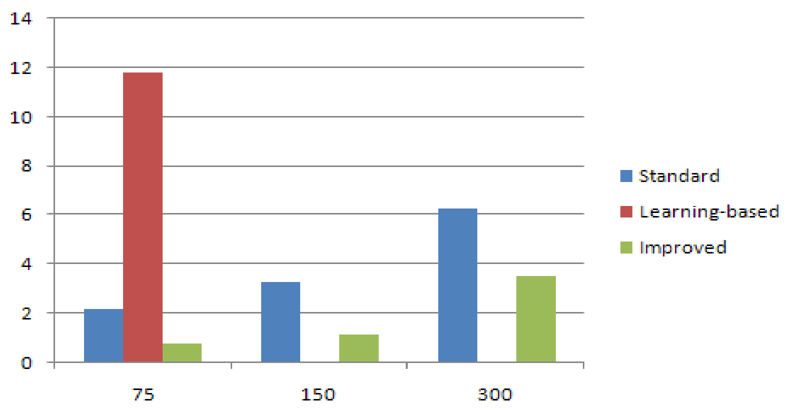

Figure 7. Running times for phil_lss_40

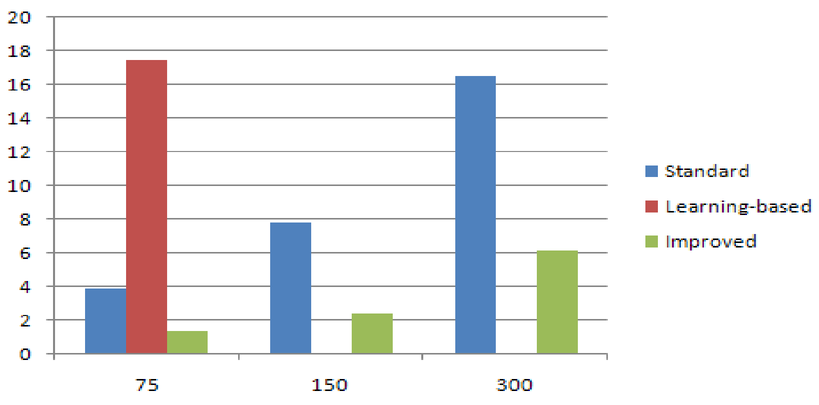

Fig 8. Running times for phil_lss_60

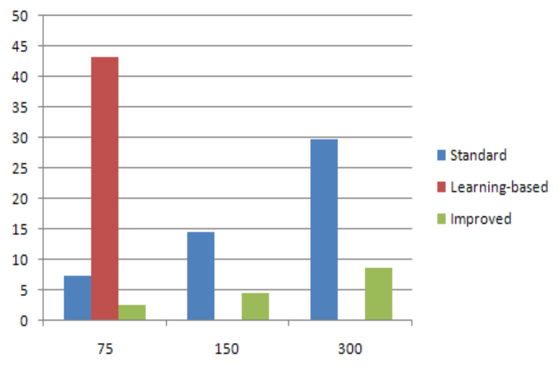

Figure 9. Running times for phil_lss_80

For some cases, the running time of the learning based method is at least ten times more than the running time of the other methods. To have better presentation, the running times of learning based method is avoided for these cases. The results show that in most cases the proposed improved approaches outperform the standard and learning based methods. Learning based method is faster for small values of the bound. However, it is time consuming for large bounds.

5. Conclusion

In this paper, two new approaches are proposed to improve the performance of iterative methods for computing bounded reachability probabilities in MDPs. The first approach avoids useless multiplications for Dirac transitions. This approach relies on the fact that a main part of transitions of the studied cases are Dirac transitions. The second proposed approach marks useful updates for the next iteration and avoid useless updates. Experimental results show promising improvements in the performance of the iterative computations for bounded reachabilities. In most cases, the running time reduces to less than the running time of the standard method. For future works, the proposed techniques can be extended for cost-bounded or multi-objective properties. The possibility of using these techniques for multi-core model checking is another direction for future studies.

References

[1] C. Baier, L. de Alfaro and V. Forejt. “Probabilistic Model Checking”, *Dependable Software Systems Engineering,* Vol. 45, pp. 1-23, 2016.

[2] C. Baier and J. Katoen. “Principles of model checking” MIT Press, USA. 2008.

[3] C. Baier, J. Klein, L. Leuschner, D. Parker and S. Wunderlich, “Ensuring the reliability of your model checker: Interval iteration for Markov Decision Processes”, *International Conference on Computer Aided Verification,* Springer Cham, Vol. 27, pp. 160-180. 2017.

[4] T. Brazdil, K. Chatterjee, M. Chmelik, V. Forejt, J. Kretinsky, M. Kwiatkowska, D. Parker and M. Ujma. “Verification of Markov decision processes using learning algorithms”, *International Symposium on Automated Technology for Verification and Analysis,* Vol. 12, pp. 98-114. 2014

[5] F. Ciesinski, C. Baier, M. Gromer and J. Klein. “Reduction techniques for model checking Markov decision processes”, *Fifth International Conference on Quantitative Evaluation of Systems,* pp. 45-54. 2008.

[6] L. De Alfaro. “Formal verification of probabilistic systems.” PhD thesis. Stanford University, US, 1997.

[7] C. Dehnert, S. Junges, J. Katoen and M. Volk. “A storm is coming: A modern probabilistic model checker”, *International Conference on Computer Aided Verification,* pp. 592-600. Springer. 2017.

[8] V. Forejt, M. Kwiatkowska, G. Norman and D. Parker. “Automated Verification Techniques for Probabilistic Systems”, *InSFM,* Vol. 11, pp. 53-113. 2011.

[9] L. Gui, J. Sun, S. Song, Y. Liu and J.S. Dong. “SCC-based improved reachability analysis for Markov decision processes”, *In: International Conference on Formal Engineering Methods,* pp. 171-186. Springer.

[10] E.M. Hahn, Y. Li, S. Schewe, A. Turrini and L. Zhang. “iscas M c: a web-based probabilistic model checker”, *International Symposium on Formal Methods,* Springer. pp. 312-317. 2014.

[11] A. Hartmanns. “On the analysis of stochastic timed systems”, PhD thesis. Saarland University, Germany. 2015.

[12] A. Hartmanns and H. Hermanns. “The modest toolset: an integrated environment for quantitative modeling and verification”. *International Conference on Tools and Algorithms for the Construction and Analysis of Systems,Springer.* pp. 593-598. 2014.

[13] J. Katoen. “The probabilistic model checking landscape”. *In Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science,* Vol. 5, pp. 31-45. 2013.

[14] M. Kattenbelt, M. Kwiatkowska, G. Norman and D. Parker. “Abstraction refinement for probabilistic software”, In: *International Workshop on Verification, Model Checking, and Abstract Interpretation*, Springer. pp. 182-197. 2009.

[15] J. Klein, C. Baier, P. Chrszon, M. Daum, C. Dubslaf, S. Kluppelholz, S. Marcker and D. Muller. “Advances in probabilistic model checking with PRISM: variable reordering, quantiles and weak deterministic Buchi automata”. *International Journal on Software Tools for Technology Transfer,* Vol. 20(2), pp. 179-194. 2018.

[16] M. Kwiatkowska, G. Norman and D. Parker. ”symmetry reduction for probabilistic model checking”, *International Conference on Computer Aided Verification, Springer.* pp. 234-248. 2006.

[17] M. Kwiatkowska, G. Norman and D. Parker. “The PRISM benchmark suite”, *9th International Conference on Quantitative Evaluation of SysTems,* IEEE CS press. pp. 203-204. 2010.

[18] M. Kwiatkowska, D. Parker and H. Qu. “Incremental quantitative verification for Markov decision processes”, *Dependable Systems & Networks (DSN), IEEE/IFIP 41st International Conference,* IEEE, Vol. 41, pp. 359-370. 2011.

[19] M. L. Puterman. “Markov decision processes: Discrete stochastic dynamic programming”, In: *Journal of the Operational Research Society,* Vol. 46(6), pp. 792-792. 1994.

[20] T. Quatmann and J. Katoen. “Sound value iteration”, *International Conference on Computer Aided Verification,* Springer, Vol. 27, pp. 643-661. 2018.

[21] M. Ujma. “On verification and controller synthesis for probabilistic systems at runtime”, Ph.D. thesis, University of Oxford. 2015.