Journal of Electrical and Computer Engineering Innovations
(JECEI)
Journal homepage: http://www.jecei.sru.ac.ir

Research paper

# Faster Convergence of Modified Policy Iteration for Quantitative Verification of Markov Decision Processes

***M. Mohagheghi****

*Department of Computer Science, Vali-e-Asr University of Rafsanjan, Rafsanjan, Iran.*

**Article Info**

**Article History:**
Received 02 February 2018
Reviewed 12 April 2018
Revised 12 May 2018
Accepted 13 November 2018

**Keywords:**
Expected rewards
Markov decision processes
Modified policy iteration
Probabilistic model checking
Reachability probabilities

*Corresponding Author's Email Address:
*mohagheghi@vru.ac.ir*

**Abstract**

**Background and Objectives:** Numerical iterative methods are widely used to compute reachability probabilities and expected rewards in probabilistic model checking of Markov decision processes. Several approaches have been proposed to improve the performance of these iterative methods. Reducing the total number of iterations is an important goal that is followed by several techniques in this paper.

**Methods:** In this paper, we consider MDPs with different levels of non-determinism. We show that modified policy iteration performs better than the other standard iterative methods when the degree of non-determinism increases. We propose some novel methods to improve the performance of the modified policy iteration method. Our approach is to define several criteria to avoid useless iterations and updates of the modified policy iteration method. The first criterion considers the graphical structure of the model to define the number of iterations after each policy modification. The second criterion is a dynamic one to determine when a policy should be updated. The third proposed approach in this work is to use several priority heaps to select states for value updates.

**Results:** Our proposed methods are implemented in the PRISM model checker and applied on several standard case studies. The results of experiments show that the running times of our approaches are less than the running time of the standard and previous methods for most case studies. The main reason for these results is that the total numbers of iterations and updates are reduced by our approaches, which results in an improvement in the performance of the method. While the running times are reduced by our approaches, the precision of computations are kept in most cases.

**Conclusion:** The proposed techniques are able to reduce the number of iterations and accelerate the convergence to the optimal policies. The also prioritize the commutations to reduce the total number of updates.

©2019 JECEI. All rights reserved.

## Introduction

Probabilistic model checking has been widely used in recent years for modeling and analyzing systems with stochastic behaviors. Some real-world case-studies that have used this technique include communication and security protocols [1], [2], power management [3], biological processes [4], game theory [5], and randomized algorithms [6]. A model checker is a software tool that automatically verifies a system specification against desired properties [1]. Markov decision processes (MDPs) are well-known formalism for modeling systems with probabilistic and nondeterministic behaviors [7], [8]. In this case, PCTL (as

a probabilistic temporal logic) is used for specifying properties of the system. PRISM [9], PAT [10] and STORM [11] are examples of probabilistic model checkers. Many computational problems in the verification of PCTL properties against MDPs are reduced to probabilistic reachability or expected cost problems. In probabilistic reachability, the extremal (minimum or maximum) probability of reaching one of the goals states of the model is computed [12]. In expected cost problems, the extremal expectation of the accumulated costs until reaching a goal state is needed. These two classes of properties are widely used to measure the quality attributes of hardware and software computer systems [1], [13], [14] and to approximate their costs [15]. Several methods exist to compute the values of these properties. Linear programming [13] computes the exact probability values. Its scalability is limited for large models [12]. In practice, iterative numerical methods are used to approximate the desired values. Value iteration (VI) and policy iteration (PI) are two iterative methods to approximate reachability probabilities or expected rewards of the model [12].

Gauss-Seidel (GS) value iteration [12] and modified policy iteration (MPI) [16] are two improved iterative methods that use the ideas of value and policy iteration.

A main challenge of the iterative methods is their running time [7], [12], [17], and [18]. Several improved techniques have been proposed to accelerate the standard iterative algorithms. SCC-based topological iterative methods [17], [18], and learning-based approaches [19] are proposed to accelerate the standard iterative methods. They avoid redundant and useless updates of states and use an appropriate ordering for updating the value of states. Symmetric reduction [20] and partial order reduction [21] are proposed for reducing the number of states of the model. The results of the previous works show that the performance of the standard and improved methods depends on the graphical structure of the model. The proposed methods in [17], [19], and [27] use the topological order for the models without cycle. On the other hand, these methods do not achieve considerable improvement for the models with large cycles [24]. In this paper, we focus on the models with large cycles and propose several techniques to improve the performance of the standard methods for this class of MDPs. While the main numbers of previous works consider value iteration as the main standard iterative method, our experiments show that the performance of MPI is usually more than the performance of value iteration for models with large cycles. We consider MPI [16] and propose several techniques to reduce its running time. A comparison of policy iteration and value iteration has been done in [12] on a few numbers of case studies. The performance of MPI has been analyzed in [26] and a naive criterion is proposed to determine the number of iterations of the method after each policy modification. However, no justification is proposed for this criterion and its performance is not compared to the performance of the other improved methods. In this paper, several techniques are proposed to reduce the running time of the MPI method for computing reachability probabilities and expected rewards. The proposed methods are threefold:

- As the main contribution of the work, we define a static criterion for the number of iterations after each policy modification. This criterion relies on the average number of outgoing actions from each state. We experimentally analyze different values for the maximum number of iterations after each policy modification. We show that MPI outperforms other iterative methods when it uses well-bound on the number of iterations after each policy modification.
- As the second contribution of the work, we define a dynamic criterion for the number of iterations after each policy modification. This criterion considers the number of changes in the selected actions to determine the number of iterations.
- To accelerate the convergence to the solution values, we propose a bounded depth priority queue method. In this method, a priority queue is used to dynamically select states for update. To reduce the overhead of priority queue, we limit its size and partition states to several priority queues.

Our experiments show that the proposed methods accelerate the SCC-based MPI method up to 70%. The rest of this paper is organized as follows: Section 2 presents some preliminaries about the probabilistic verification of MDPs. Section 3 describes our methods for MPI iteration. Experimental results are presented in Section 4 and Section 5 concludes the paper.

## Preliminaries

In this section, we provide an overview of MDPs and the standard iterative methods for probabilistic model checking. More details are available in [1], [12], [18]. A probability distribution on a finite set X is defined as a function $f : X \mapsto [0,1]$ such that $\sum_{x \in X} f(x) = 1$. The set of all probability distributions on $X$ is denoted by $D(X)$.

### A. Definitions

**Definition 1. (MDP)**. A Markov decision process (MDP) is a tuple $M(S, s_0, Act, P, R, G)$ where

- $S$ is a finite set of states,
- $s_0 \in S$ is the initial state,
- $Act$ is a finite set of actions. For each state $s \in S$, $Act(s) \subset Act$ is used for the set of enabled actions of

$s$,

- $P: S \times Act \to D(S)$ is a probabilistic transition function. We use $P(s,\alpha,s')$ for the probability of a transition from $s$ to $s'$ by the action $\alpha$.
- $R: S \times Act \to \Re$ is a reward function
- $G \subset S$ is the set of goal states.

The size of *M* is defined as the number of its states and transitions and is shown by $|M|$. MDPs are widely used to model decision making under uncertainty in stochastic environments [1], [3]. The actions of an MDP are used to model non-deterministic choices of a system environment. The probabilistic transition function is used to model the stochastic behavior of the environment [16]. For an MDP *M* and one of its states $s \in S$, the MDP performs a transition in two steps:

1- It non-deterministically selects an enabled action $\alpha \in Act(s)$.

2- It randomly selects the destination state $s' \in S$ with probability $P(s,\alpha,s')$.

A path of *M* is a sequence of states and transitions of the form $\pi = s_0 \xrightarrow{\alpha_0} s_1 \ldots \xrightarrow{\alpha_{n-1}} s_n$ where $s_i \in S$, $\alpha_i \in Act(s_i)$ and $P(s_i,\alpha_i,s_{i+1}) > 0$ for all $0 \le i < n$. According to the selected action $\alpha$ the reward $R(s,\alpha)$ is collected by the system. We use $|\pi|$ for the number of transitions of $\pi$, $last(\pi) = s_n$ for the last state of $\pi$ and $\pi(i)$ for the (i+1)-th state of of $\pi$. We use the index $i$ in $s_i$ to show the order of the states in the path and it may be different from the order of the states in $S$. For a state $s \in S$, we use $FPath_{s,M}$ for the set of all finite paths of *M* that start from the state $s$ and define $FPath_M$ as $\bigcup_{s \in S} FPath_{s,M}$. An MDP can perform its transitions infinitely or terminate when it reaches one of the goal states G. We use $Post(s)$ and $pre(s)$ for the set of successor and predecessor states of s:

$$Post(s) = \{s' \in S \mid \exists \alpha \in Act(s), P(s,\alpha,s') > 0\} \quad (1)$$

$$pre(s) = \{s' \in S \mid s \in Post(s)\} \quad (2)$$

$Post(s,\alpha)$ is used for the set of $\alpha$-successor states of $s$:

$$Post(s,\alpha) = \{s' \in S \mid P(s,\alpha,s') > 0\} \quad (3)$$

A discrete-time Markov chain (DTMC) is an MDP without non-deterministic choices, i.e., an MDP for which every state has only one enabled action.

*B. Quantitative probabilistic properties*

Two important classes of properties that are used in the verification of Markov models are reachability properties and expected rewards. In deterministic models (such as DTMCs) a reachability probability is defined as the probability of reaching one of the goal states G of the model. Expected rewards are defined as the expectation of accumulated rewards before reaching a goal state [1]. In the case of MDPs, a reachability probability is defined as the extremal (minimal or maximal) probability of reaching a goal state. To compute the extremal reachability probabilities or expected rewards of an MDP, we need to resolve its non-deterministic choices. To do so, we should use the notion of policies. A (deterministic) policy for an MDP M is a function $\sigma: FPath_M \to Act$ that maps an enabled action $\alpha \in Act(last(\pi))$ to any finite path $\pi \in FPath_M$. We use $FPath_M^\sigma$ for the set of all finite paths of the form $s_0 \xrightarrow{\sigma(s_0)} s_1 \ldots s_{n-1} \xrightarrow{\sigma(s_{n-1})} s_n$ and we use $FPath_{s,M}^\sigma$ for those paths of $FPath_M^\sigma$ that start from *s*. We use $\Sigma_M$ for the set of all policies of M. A policy is called memoryless if it only depends on the last state of $\pi$. In general, a policy can be probabilistic and maps to a path $\pi$, a probability distribution $f(Act)$ for the selected actions. However, deterministic and memory-less policies are sufficient for computing the extremal reachability probabilities or expected rewards of MDPs [2]. For the remainder of this paper, we use the word *policy* for a memoryless and non-deterministic policy. For a policy $\sigma \in \Sigma_M$ and a state $s \in S$ we define $reach_s^\sigma(G)$ as the set of all finite paths that start from $s$, end in G and select actions according to the policy $\sigma$:

$$reach_s^\sigma(G) = \{\pi \in FPath_{s,M}^\sigma \mid last(\pi) \in G, \forall i \lhd |\pi|: \pi(i) \notin G\} \quad (4)$$

For a policy $\sigma$ and a path $\pi \in FPath_M^\sigma$, a probability measure $pr^\sigma(\pi)$ is defined as the product of the probability of transitions between the states of $\pi$:

$$pr^\sigma(\pi) = \prod_{i=0}^{n-1} P(s_i, \sigma(s_i), s_{i+1}) \quad (5)$$

The probability measure is used to formally define the extremal reachability probabilities. For the sake of simplicity, we miss the model name *M* for the remaining definitions. For any state $s \in S$, we use $pr_s^{\min}(\Diamond G)$ and $pr_s^{\max}(\Diamond G)$ for the minimal and maximal probability of reaching G from *s* over all policies $\sum_\pi$ and formally define them as:

$$pr_s^{\min}(\Diamond G) = \inf_{\sigma \in \Sigma_M} pr(reach_s^\sigma(G)) \quad (6)$$

$$pr_s^{\max}(\Diamond G) = \sup_{\sigma \in \Sigma_M} pr(reach_s^\sigma(G)) \quad (7)$$

where we use $pr(reach_s^\sigma(G))$ for the total probability of reaching G from *s* under the policy $\sigma$:

$$pr(reach_s^\sigma(G)) = \sum_{\pi \in reach_s^\sigma(G)} pr^\sigma(\pi) \quad (8)$$

The other class of properties against MDPs is defined as the expected accumulated reward before reaching a goal state. For any path $\pi \in FPath_M^{\sigma}$, we define the random variable $r_F$ as the total accumulated reward along $\pi$ until reaching a goal state G [25]:

$$r_F(\pi) = \begin{cases} \sum_{i=0}^{j-1} R(s_i, \sigma(s_i)) & if \ \ \exists j.\pi(j) \in G \text{ and } \forall i \neq j.\pi(i) \notin G \\ \infty & \text{otherwise} \end{cases} \quad (9)$$

The notation $\mathrm{E}_s^{\sigma}(r_F)$ is used for the expectation of the random variable $r_F$ under policy $\sigma$. Starting from a state $s \in S$, the maximum and minimum expected accumulated rewards before reaching a goal state are denoted by $\mathrm{E}_s^{\min}$ and $\mathrm{E}_s^{\max}$ defined as:

$$\mathrm{E}_s^{\min} = \inf_{\sigma \in Pol_M} E_s^{\sigma}(r_F) \quad (10)$$

$$\mathrm{E}_s^{\max} = \sup_{\sigma \in Pol_M} E_s^{\sigma}(r_F) \quad (11)$$

*C. Qualitative probabilistic properties*

Some graph-based pre-computations can determine the set of states for which the maximal (or minimal) reachability probability is exactly 0 or 1. These sets are defined as:

$$S_{\min}^0 = \{s \in S \mid pr_s^{\min}(\Diamond G) = 0\} \quad (12)$$

$$S_{\min}^1 = \{s \in S \mid pr_s^{\min}(\Diamond G) = 1\} \quad (13)$$

$$S_{\max}^0 = \{s \in S \mid pr_s^{\max}(\Diamond G) = 0\} \quad (14)$$

$$S_{\max}^1 = \{s \in S \mid pr_s^{\max}(\Diamond G) = 1\} \quad (15)$$

The computations of these sets are based on definitions (1)-(3) and are called qualitative analysis [1], [12] and improve the running time and accuracy of computed reachability probabilities. The computations of expected rewards are focused on $S^1$ sets because the accumulated rewards of the other states may converge to infinity [25].

*D. Iterative Numerical Methods for Quantitative Probabilistic Properties*

The standard approach to determine reachability probabilities or expected reward is to use an iterative numerical method. To review the standard and improved iterative methods, we focus on the extremal expected rewards.

Extremal reachability probabilities can be considered as a special (and simplified) case of the expected rewards [12]. Using equation (11) and considering $x_s = E_s^{\max}$ for any $s \in S_{\min}^1$, the value of $x_s$ is computed as the least solution of the Bellman equation:

$$x_s = \begin{cases} 0 & \text{if s} \in \text{G} \\ \max_{\alpha \in Act(s)} (R(s,\alpha) + \sum_{s' \in S} \delta(s,\alpha,s').x_{s'}) & \text{otherwise} \end{cases} \quad (16)$$

Value iteration and policy iteration are two standard iterative methods to approximate the values of $x_s$ for Bellman equation. For each iteration $k$, value iteration uses a vector $\overline{x^k}$ for approximated values. This vector is initialized to 0 for $k = 0$. For every iteration $k > 0$ and each state $s \in S_{\min}^1$ the values of $x_s^k$ are updated as:

$$x_s^k = \begin{cases} 0 & \text{if } s \in G \\ \max_{\alpha \in Act(s)} (R(s,\alpha) + \sum_{s' \in S} \delta(s,\alpha,s').x_{s'}^{k-1}) & \text{otherwise} \end{cases} \quad (17)$$

The iterations continue until the convergence criterion is satisfied. For a given threshold $\varepsilon$, the standard convergence criterion is to check the maximum difference of approximated values between two iterations, i.e., the iterations terminate if the condition $\max_{s \in S}(x_s^k - x_s^{k-1}) < \varepsilon$ is satisfied. In the Gauss-Seidel version of value iteration, the last updated values are used to update the value of each state. In this case, only one vector is used for computations. In policy iteration, a sequence of policies is considered and the expected values of the states under the selected policy are computed. To do so, several iterations are done until the termination criterion is satisfied. MPI [16] uses a limited number of iteration after each policy modification. This method avoids useless computations of non-optimal policies. Algorithm 1 shows the idea of modified policy iteration.

Lines 2-4 of algorithm 1 initializes the vector $\overline{x}$. The main part of the algorithm is from line 7 to line 20. The algorithm starts from a random policy (line 5) and for each policy it updates the values of $x_s$ in line 10. Note that the algorithm uses the $mx_iters$ parameter (line 8) to limit the number of iterations for each policy.

Algorithm 1: Modified policy iteration for maximal expected rewards

*Input*: An MDP M $= (S, s_0, Act, P, R, G)$, parameters: $max_iters, \varepsilon$
*Output*: Approximation of $E_s^{max}$
1. Compute $S_{\min}^1$
2. for each s ∈ S *do*
3. $x_s = 0$;
4. *end for*;
5. select an arbitrary policy $\sigma$;
6. $\delta = 1$;
7. while $\delta > \varepsilon$ do
8. for k = 1 to mx_iters do
9. for each s ∈ $S_{\min}^1$ *do*

10. $x_s = R(s,\sigma(s)) + \sum_{s' \in S}(P(s,\sigma(s),s').x_{s'})$;
11. end for;
12. end for;
13. $\delta = 0$;
14. for each s ∈ $S^1_{\min}$ do
15. $\sigma(s) = argmax_{\alpha \in Act(s)}(R(s,\sigma(s)) + \sum_{s' \in S}(P(s,\sigma(s),s').x_{s'}))$;
16. $x_{new} = max_{\alpha \in Act(s)}(R(s,\sigma(s)) + \sum_{s' \in S}(P(s,\sigma(s),s').x_{s'}))$;
17. $\delta = \max(\delta, x_{new} - x_s)$;
18. $x_s = x_{new}$;
19. end for;
20. end while;
21. return $\overline{x}$;

In line 15, it computes a new policy. The iterations continue until the maximum difference of the value of states after policy computations (line 17) drops below the predefined threshold $\varepsilon$.

## Improving Modified Policy Iteration for Expected Rewards

To reduce the running time of MPI, we propose two approaches to focus on more important iterations and updates. The first approach considers the information of the model to use a criterion for the number of iterations after each policy modifications. We call it the model-based criterion. The second approach considers the rate of changes of the computed policies as a dynamic criterion for determining the number of iterations after each policy modification.

### *A. Model-based Approach for Avoiding Useless Iterations*

The value of $mx_iters$ parameter can affect the running time of the MPI method. The default value of this parameter in the PRISM and IscasMC model checkers is 100. One can consider the value iteration method as a version of MPI with $mx_iters = 0$ and the policy iteration method as a version of MPI with $mx_iters = \infty$ [22]. To the best of our knowledge, no previous work has studied the impact of this parameter on the performance of modified policy iteration. Good options for this parameter can avoid non-important iterations and therefore accelerate the MPI method. The running time of algorithm 1 is a function of the number of multiplications of lines 10 and 16. To reduce the running time of modified policy iteration, we try to reduce the number of these multiplications. For a known MDP M, we use $\theta$ as the average number of actions per state, i.e., $\theta = \frac{|Act|}{|S|}$. The value of $\theta$ can be used to determine a good value for $mx_iters$. In general, the number of multiplication of line 16 of algorithm 1 is approximately $\theta$ times more than the number of multiplications of line 10. For large values of $\theta$ we should try to reduce the number of iterations of line 16. In this case, Algorithm 1 should use higher values for $mx_iters$ to have more accurate results after each policy modification. On the other hand, for small values of $\theta$ (values near 1), there is not a considerable difference in the number of multiplications of lines 10 and 16. In this case., the method should reduce the number of multiplications of line 10. Using smaller values for $mx_iters$, the method may converge faster to the optimal policy. As a result, we define a linear relationship between $\theta$ and $mx_iters$ as:

$$mx_iters = a.\theta + b \quad (18)$$

We use two parameters $a$ and $b$ to have better values for the $mx_iters$ parameter. We use equation 9 as a static criterion to determine the value of $mx_iters$. MPI determines the value of this parameter once according to the structure of the model. In the next section, we experimentally determine the value of the $a$ and $b$ parameters of equation (18).

### *B. A Dynamic Approach for Number of Iterations after Policy Modification*

One can use a dynamic criterion for approximating $mx_iters$ in order to accelerate the convergence of modified policy iteration to the optimal policy. This criterion uses the accuracy of each policy to determine the value of $mx_iters$. If the computed policy is far away from the optimal one, MPI should use a small number of iterations and $mx_iters$ should be small to avoid useless computations of the non-optimal actions. If the computed policy is near to the optimal one, the method can rely on it and have more iteration. However, there is not a straightforward approach to compute the optimal policy [12]. Instead, we consider the number of action changes after each policy modification as a heuristic for the optimality of a policy. For an optimal policy (if it is unique), we expect to have no change in the actions after a policy modification (line 15 of algorithm 1). Let $\Delta_{Act}$ be the number of changed actions after a policy modification. We use $\frac{\Delta_{Act}}{|S|}$ as a criterion for the optimality of a policy. Smaller values of this fraction mean smaller changes in the number of selected actions. Hence, the MPI method can perform more iteration for the computed policy. We define the dynamic heuristic as:

$$mx_iters = \min(200, \frac{|S|}{1+\Delta_{Act}}) \quad (19)$$

We use equations (18) and (19) as two heuristics for

approximating good values for the $mx_iters$ parameter. Algorithm 1 can use equation (18) before the outer loop (before line 7) or equation (19) before the first inner loop (line 8) to determine good values for $mx_iters$.

### C. Avoiding Useless Updates

Several methods have been proposed in previous works to reduce the number of updates of iterative methods in quantitative analysis of MDPs. The general idea of these approaches is to prioritize states for updates [16], [23]. Prioritized sweeping [21] and priority queue value iteration [25] show promising results in the performance of value and policy iteration methods. In these methods, the difference of values between the new update and the old one is considered as a priority criterion of a state. In each step, a state $s$ with the highest priority is selected and the states in $pre(s)$ are selected to update. A priority queue can be used to select the best states.

One drawback of the priority queue value iteration (PQVI) is the memory and time overhead of the queue operations.

The time complexity of adding a state to and removing one from a priority queue is in $o(\log_2^n)$ where $n$ is the size of the priority queue [1]. The overhead of these computations can degrade the performance of an iterative method for large MDPs. Note that in most cases, the structure of the model is sparse and the number of multiplications for updating each state is less than the number of queue operations. To alleviate this overhead, we define a bound on the size of the priority queue to limit its depth.

For example, if we bound the size of the priority queue to 31, we limit its depth to 4, which reduces its overhead.

The limited priority queue, on the other hand, may miss some important states. To keep more states in the priority queue, we partition the states of a model to $k$ classes and use a queue for each class. In each iteration, the best state of each priority queue is selected and its predecessor states are updated and added to the related priority queue.

For simplicity, we define $k = \left\lceil \frac{|S|}{31} \right\rceil$. To use MPI with multiple queues, the method modifies equation (19) as:

$$mx_iters = 31 \times \min(200, \frac{|S|}{1+\Delta_{Act}}) \quad (20)$$

We multiply the right side of equation (20) by 31 because in each iteration, the method selects only $k$ states for update. To have a better understanding of the notations and abbreviations of this paper, we list them in Table 1.

Table 1: Nomenclature

| Notation | Meaning |
|---|---|
| M | Markov decision process (MDP) |
| S | Set of states |
| Act | Set of actions |
| P | Probabilistic transition function |
| R | Reward function |
| G | Set of goal states |
| A | Enabled action |
| $\pi = s_0 \xrightarrow{\alpha_0} s_1 \ldots \xrightarrow{\alpha_{n-1}} s_n$ | A path of an MDP |
| $\pi(i)$ | (I+1)-th state of of $\pi$ |
| $FPath_{s,M}$ | The set of all finite paths of *M* |
| $Post(s)$ | The set of successor states of $s$ |
| $Post(s,\alpha)$ | The set of $\alpha$ -successor states of $s$ |
| $reach_s^{\sigma}(G)$ | The set of all finite paths that start from $s$, end in G, and select actions according to the policy $\sigma$ |
| $pr^{\sigma}(\pi)$ | The product of the probability of transitions between the states of $\pi$ |
| $pr_s^{\min}(\Diamond G)$ | The minimal probability of reaching G from *s* over all policies |
| $pr_s^{\max}(\Diamond G)$ | The maximal probability of reaching G from *s* over all policies |
| $\mathrm{E}_s^{\sigma}(r_F)$ | The expectation of the random variable $r_F$ under policy $\sigma$ |
| $\mathrm{E}_s^{\min}$ | Minimum expected accumulated rewards before reaching a goal state |
| $\mathrm{E}_s^{\max}$ | Maximum expected accumulated rewards before reaching a goal state |
| $x_s$ | Reachability or expected value of a state s |
| $\varepsilon$ | Threshold for convergence |
| $mx_iters$ | The number of iterations for each policy |
| $\theta$ | The average number of actions per state |
| $\Delta_{Act}$ | The number of changed actions after a policy modification |

## Results and Analysis

To compare the running time of the standard iterative methods for verification of quantitative properties, we consider four classes of the standard case studies of PRISM. The *Consensus* and *zeroconf* case studies are used for extremal reachability probabilities and the *wlan* and *i_jalfon* case studies are used for extremal expected rewards [17], [18].

More details about these case studies are available in [11], [12], Table 2 shows some details of these MDPs including the number of states of each model, the average number of actions per state ($\theta$) and the average number of transitions per action. We implemented our proposed methods in PRISM 4.5 which

is currently the last version of this model checker. We use sparse engine of PRISM to implement the main part of our proposed techniques. This engine is mainly developed in C and can be compiled in Linux distributions. We use a machine with core-i7 CPU and 8GB of Ram using Ubuntu 18 to run the experiments. Table 3 shows the running time of the standard iterative methods for computing extremal reachability probabilities (*Cons* and *Zeroconf* samples) and expected rewards (*wlan* and *i-jalefon* samples).

All times are in seconds. In all cases, we use the SCC-based version of the iterative method. We use the *Tarjan* algorithm [1] to find SCCs. We consider the Gauss-Seidel value iteration (GS), the standard policy iteration (PI) and modified policy iteration (MPI) methods which are available in the current version of PRISM [12]. For the last case, we consider three values for $mx_iters$ to compare the impact of this value on the performance of the method.

The results of Table 3 show that the MPI method outperforms the standard policy iteration and Gauss-Seidel value iteration for all models. The impact of the $mx_iters$ parameter is important for the running time of the *zeroconf, wlan* and *i_jalfon* models. For *zeroconf* and *wlan* models which have small values of $\theta$, smaller values of $mx_iters$ are better for MPI.

Table 2: Case study models

| Model (parameters) | Param Values | \|S\| | $\theta$ | Number of Transitions |
|---|---|---|---|---|
| Consensus (N,K) | 4,12 | 32056 | 2.7 | 109056 |
| | 4,25 | 66168 | 2.7 | 225536 |
| | 6,8 | 148548 | 4 | 745536 |
| | 6,12 | 220548 | 4 | 1108416 |
| Zeroconf (K) | 10 | 3001911 | 1.84 | 6787615 |
| | 14 | 4427159 | 1.84 | 10009768 |
| | 18 | 5477150 | 1.84 | 12374708 |
| Wlan (N,ttm) | 5,10 | 1295218 | 1.27 | 2929960 |
| | 5,500 | 2064518 | 1.54 | 4477380 |
| | 6,10 | 5007548 | 1.27 | 11475748 |
| | 6,500 | 6534878 | 1.54 | 14540698 |
| i-jalfon (N) | 16 | 65535 | 8 | 917504 |
| | 18 | 262143 | 9 | 4128768 |
| | 20 | 1048575 | 10 | 18350080 |

For the *i_jalfon* models, which have higher values of $\theta$ the performance of MPI degrades when the $mx_iters$ parameter is decreased. To determine the parameters of equation (9), we consider six classes of MDPs with different values of $\theta$. Higher values of $\theta$ mean higher values of non-determinism in a model. Each class contains ten artificially created MDPs. The graphical structure and transition probabilities are determined randomly from a predefined range. We consider MPI with different $mx_iters$ parameter in the range of 10 to 200.

Fig. 1 to Fig. 6 propose the average running time of MPI in compare with the Gauss-Seidel method. The horizontal axis of each figure shows fixed values of $mx_iters$.

Table 3: Running time of standard iterative methods and mpi with three different values for $mx_iters$

| Model (parameters) | Param Values | GS | PI | MPI (100) | MPI (50) | MPI (20) |
|---|---|---|---|---|---|---|
| Consensus (N,K) | 4,12 | 6.7 | 8.4 | **2.45** | 2.46 | 2.51 |
| | 4,25 | 55.7 | 67.5 | **17.3** | 17.4 | 17.5 |
| | 6,8 | 73.5 | 68.2 | **10.9** | 11 | 11.1 |
| | 6,12 | 243 | 218 | **34.5** | 34.7 | 35 |
| Zeroconf (K) | 10 | 14.3 | 38.5 | 31.7 | 23.8 | **10.8** |
| | 14 | 18.2 | 52 | 46.3 | 32.9 | **14.4** |
| | 18 | 29.7 | 72.9 | 65.5 | 47.6 | **23.7** |
| Wlan (N,ttm) | 5,10 | 4.17 | 11.3 | 5.42 | 4.83 | **3.97** |
| | 5,500 | 5.83 | 13.2 | 5.92 | 5.45 | **4.2** |
| | 6,10 | 27.3 | 45.4 | 34.5 | 28.9 | **21.2** |
| | 6,500 | 31.7 | 52.4 | 35.9 | 30.3 | **23** |
| i-jalfon (N) | 16 | 5.32 | 3.14 | **3.11** | 3.17 | 3.38 |
| | 18 | 33.2 | 21.7 | **20.4** | 22.4 | 24.7 |
| | 20 | 198 | 118 | **108** | 123 | 149 |

To use good values for the constants of equation (9), we use the results of these 6 classes of case studies. For each class, we consider the best values of $ms_iters$ (minimizes the running time). Next, we use an interpolation upon the value of $\boldsymbol{\theta}$ and the best value of $ms_iters$. Fig. 7 shows a linear relation that is computed as a result of the interpolation where the vertical axis shows the fixed number for $mx_iters$ and the horizontal axis shows different values of $\boldsymbol{\theta}$. According to this interpolation, we set the parameters of equation (9) as $a = 35$ and $b = -12$.

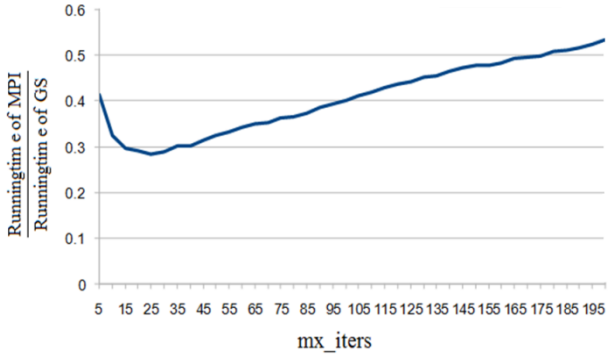


Fig. 1: $\frac{\text{Runningtime of MPI}}{\text{Runningtime of GS}}$ with different values of $mx_iters$ for $\theta = 5$. The best performances are where $mx_iters$ is in [20-30].

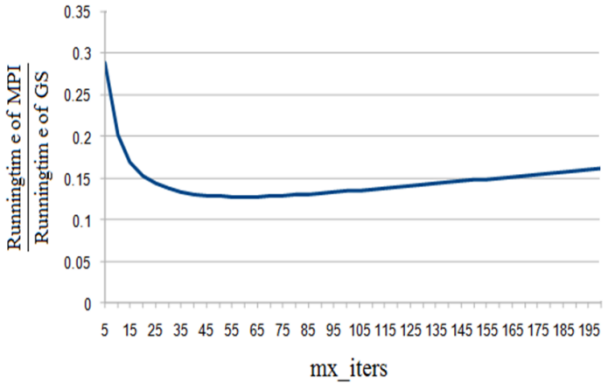


Fig.2: $\frac{\text{Runningtime of MPI}}{\text{Runningtime of GS}}$ with different values of $mx_iters$ for $\theta = 10$. The best performances are where $mx_iters$ is in [50-70].

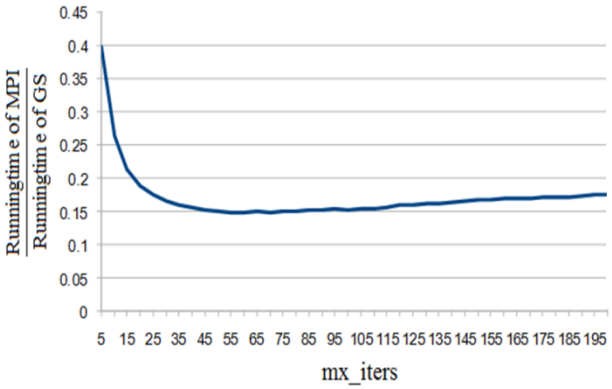


Fig. 3: $\frac{\text{Runningtime of MPI}}{\text{Runningtime of GS}}$ with different values of $mx_iters$ for $\theta = 15$. The best performances are where $mx_iters$ is in [50-90].

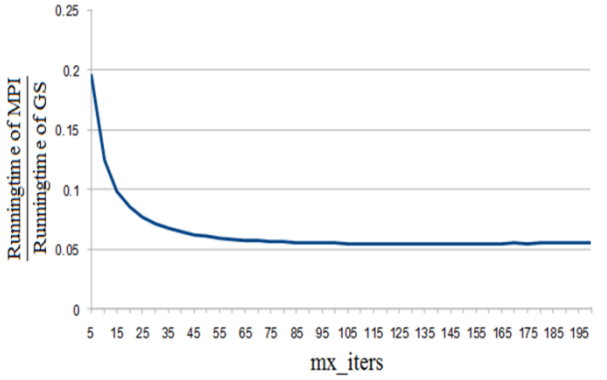


Fig. 4: $\frac{\text{Runningtime of MPI}}{\text{Runningtime of GS}}$ with different values of $mx_iters$ for $\theta = 20$. The best performances are where $mx_iters$ is in [120-160].

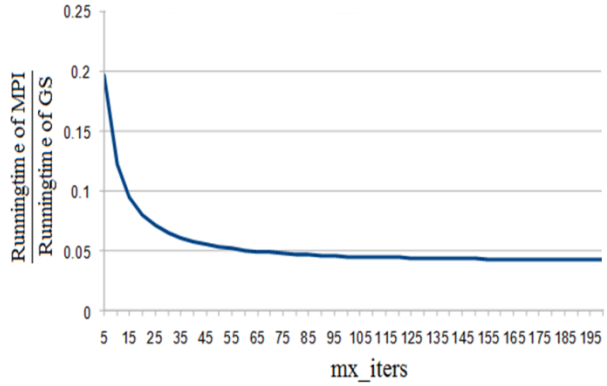


Fig. 5: $\frac{\text{Runningtime of MPI}}{\text{Runningtime of GS}}$ with different values of $mx_iters$ for $\theta = 25$. The best performances are where $mx_iters$ is in [150-190].

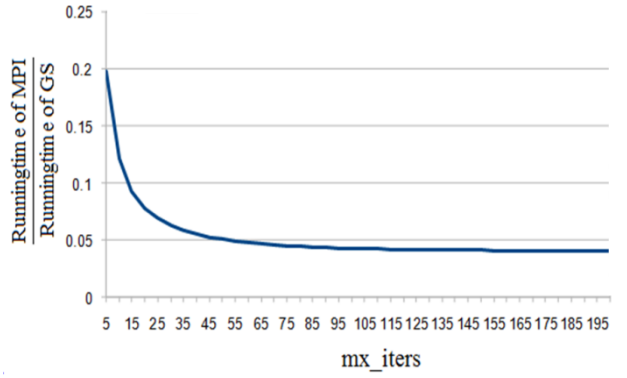


Fig. 6: $\frac{\text{Runningtime of MPI}}{\text{Runningtime of GS}}$ with different values of $mx_iters$ for $\theta = 50$. The best performances are where $mx_iters$ is in [160-200].

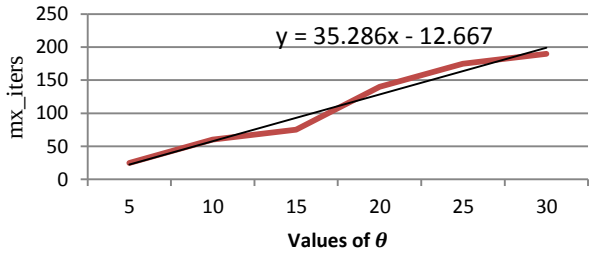


Fig. 7: Interpolation for equation (9). For each value of $\theta$ we consider the mid-point of the best values.

In Table 4 we compare the running time of the MPI method with our proposed improvements. We consider MPI with $mx_iters = 100$ as the de fault value in PRISM, MPI with equation 9 and equation 11. We also consider the priority queue MPI in two modes. The first mode uses a priority queue with the size of or related SCC (called unlimited priority queue MPI) while the second mode limits the size of priority queue to 31 (called multiple PQ-MPI). The best running times are in bold.

The results of Table 4 show that in most cases our proposed methods for the value of $mx_iters$ outperform the performance of MPI. The results are more interesting for the *zeroconf* and *wlan* models. Foe *consensus* models, the initial policies are new to the optimal one and the default value for $mx_iters$ is a good option. For *i-jalfon* models, the value of $\theta$ is more than other classes and $mx_iters = 100$ is also good for this class. The results for priority queue MPI shows that our proposed multiple priority queue methods outperform the unlimited priority queue version of MPI. The improvements are more for the cases with larger SCCs where our method reduces the overhead of priority queue.

## Conclusion

In this paper, we proposed three techniques to reduce the running time of MPI for computing extremal properties of MDPs. The proposed methods are considered as extensions to the SCC-based iterative methods. Our experiments show that MPI with good values of the $mx_iters$ parameter outperforms all other proposed methods. Our improvement of the priority queue modified policy iteration also outperforms the unlimited priority queue version of the method. For future works, we plan to develop the proposed methods for other classes of properties, such as LTL or mean-payoff.

Table 4: Running time of the proposed improved methods

| Model (parameters) | Param values | MPI (100) | MPI (eq-9) | MPI (eq-11) | Unlim PQ-MPI | Mult PQ-MPI |
|---|---|---|---|---|---|---|
| Consensus (N,K) | 4,12 | 2.45 | 2.43 | 2.67 | 2.68 | **2.14** |
| | 4,25 | 17.3 | 17.2 | 17.9 | 18.7 | **15.8** |
| | 6,8 | 10.9 | 10.8 | 11.5 | 12.2 | **9.4** |
| | 6,12 | 34.5 | 34.3 | 36.3 | 39.4 | **28.2** |
| Zeroconf (K) | 10 | 31.7 | 13.4 | 14.4 | 15.7 | **8.6** |
| | 14 | 46.3 | 18.5 | 19.7 | 21.2 | **11.6** |
| | 18 | 65.5 | 27.4 | 26.1 | 27.6 | **16.4** |
| Wlan (N,ttm) | 5,10 | 5.42 | 4 | 4.42 | 4.23 | **3.66** |
| | 5,500 | 5.92 | 4.1 | 4.52 | 4.52 | **4.2** |
| | 6,10 | 34.5 | 22.6 | 23.7 | 20.2 | **19.2** |
| | 6,500 | 35.9 | 23.2 | 24.3 | 21.6 | **19** |
| i-jalfon (N) | 16 | 3.11 | 3.03 | 3.17 | 3.22 | **2.58** |
| | 18 | 20.4 | 18.9 | 20.2 | 21.5 | **14.6** |
| | 20 | 108 | 97.2 | 104 | 109 | **73.2** |

## Author Contributions

All parts of this work including the proposed methods, implementations and experimental results have been proposed and developed by Mohammadsadegh Mohagheghi.

## Acknowledgment

The author acknowledges the HPC center of Vali-e-Asr University for providing the opportunity of running the experiments on their nodes.

## Conflict of Interest

The author declares that there is no conflict of interests regarding the publication of this manuscript. In addition, the ethical issues, including plagiarism, informed consent, misconduct, data fabrication and/or falsification, double publication and/or submission, and redundancy have been completely observed by the authors.

## References

[1] C. Baier, J.P. Katoen, Principles of Model Checking, MIT press, 2008.

[2] Y. Feng, Y. Nengkun, M. Ying, "Model checking quantum Markov chains," Journal of Computer and System Sciences, 79(7): 1181-1198, 2013.

[3] G. Norman, D. Parker, M. Kwiatkowska, S. Shukla, R. Gupta, "Using probabilistic model checking for dynamic power management," Formal Aspects of Computing, 17(2): 160-176, 2005.

[4] M. Kwiatkowska, C. Thachuk, "Probabilistic model checking for biology," Software Systems Safety, 36: 165-189, 2014.

[5] T. Chen, F. Vojtch, M. Kwiatkowska, D. Parker, A. Simaitis. "Automatic verification of competitive stochastic systems," Formal Methods in System Design, 43(1): 61-92, 2013.

[6] M. Kwiatkowska, G. Norman, D. Parker, "Probabilistic verification of Hermans self-stabilisation algorithm," Formal Aspects of Computing, 24(4): 661-670, 2012.

[7] M. Kwiatkowska, G. Norman, D. Parker, "Probabilistic model checking: advances and applications," in Formal System Verification): 73-121, Springer, Cham, 2018.

[8] N. Kamaleson, "Model reduction techniques for probabilistic verification of Markov chains," Ph.D. Diss., University of Birmingham, 2018.

[9] M. Kwiatkowska, G. Norman, D. Parker, "PRISM 4.0: verification of probabilistic real-time systems," presented at the International conference on computer aided verification, Berlin, Heidelberg, Germany, 2011.

[10] J. Sun, Y. Liu, J.S Dong, J. Pang, "PAT: Towards flexible verification under fairness," presented at the International Conference on Computer Aided Verification, Berlin, Heidelberg, Germany, 2009.

[11] C. Dehnert, S. Junges, J.P. Katoen, M. Volk, "A storm is coming: A modern probabilistic model checker," presented at the International Conference on Computer Aided Verification, Berlin, Heidelberg, Germany, 2017.

[12] V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, "Automated verification techniques for probabilistic systems," presented at the International School on Formal Methods for the Design of Computer, Communication and Software Systems, Berlin, Heidelberg, Germany, 2011.

[13] A. Bianco, L. De Alfaro, "Model checking of probabilistic and nondeterministic systems," presented at the International Conference on Foundations of Software Technology and Theoretical Computer Science, Berlin, Heidelberg, Germany, 1995.

[14] H. Rashidi, M. Sadeghzadeh Hemayati, "Software quality models: A comprehensive review and analysis," Journal of Electrical and Computer Engineering Innovations, 6(1): 59-76, 2018.

[15] A. Saberi Nejad, R. Tavoli "A method for estimating the cost of software using principle components analysis and data mining," Journal of Electrical and Computer Engineering Innovations, 6(1): 33-42, 2018.

[16] M. L. Puterman, C. S. Moon, "Modified policy iteration algorithms for discounted Markov decision problems," Management Science, 24(11): 1127-1137, 1978.

[17] M. Kwiatkowska, D. Parker, H. Qu, "Incremental quantitative verification for Markov decision processes," in Proc. IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN): 359-370, 2011.

[18] C. Baier, J. Klein, L. Leuschner, D. Parker, S. Wunderlich, "Ensuring the reliability of your model checker: Interval iteration for markov decision processes," in Proc. International Conference on Computer Aided Verification: 160-180, 2017.

[19] T. Brzdil, K. Chatterjee, M. Chmelk, V. Forejt, J. Ketnsk, M. Kwiatkowska, D. Parker, M. Ujma, "Verification of Markov decision processes using learning algorithms," in Proc. International Symposium on Automated Technology for Verification and Analysis): 98-114, 2014.

[20] M. Kwiatkowska, G. Norman, D. Parker, "Symmetry reduction for probabilistic model checking," in Proc. International Conference on Computer Aided Verification: 234-248, 2006.

[21] M. Groesser, C. Baier, "Partial order reduction for Markov decision processes: A survey," in Proc. International Symposium on Formal Methods for Components and Objects: 408-427, 2005.

[22] O. Shlakhter, "Acceleration of iterative methods for Markov decision processes," Ph.D. Diss., 2010.

[23] D. Wingate and D. Kevin, "Prioritization methods for accelerating MDP solvers," Journal of Machine Learning Research, 6(5): 851-881, 2005.

[24] M. Mohagheghi, J. Karimpour, A. Isazadeh. "Prioritizing methods to accelerate probabilistic model checking of discrete-time Markov models," The Computer Journal, 63(1): 105-122, 2020.

[25] J. Klein, C. Baier, P. Chrszon, M. Daum, "Advances in probabilistic model checking with PRISM: Variable reordering, quintiles and weak deterministic Buchi automata," International Journal on Software Tools for Technology Transfer, 20(2): 179-194. 2018.

[26] L. Feng, "On learning assumptions for compositional verification of probabilistic systems," Ph.D. Thesis, University of Oxford, 2013.

[27] F. Ciesinski, C. Baier, M. Größer, J. Klein, "Reduction techniques for model checking Markov decision processes," in Proc. IEEE 2008 Fifth International Conference on Quantitative Evaluation of Systems: 45-54, 2008.

## Biographies



**Mohammadsadegh Mohagheghi** (M'1888, F'17) received his Ph.D. in Computer Science from University of Tabriz in 2019, M.Sc. in Computer Science from Sharif University of Technology in 2008, and B.Sc. in Software Engineering from Shahid Beheshti University in 2006. He is currently a faculty member of Computer Science in Vali-e-Asr University of Rafsanjan, Iran. His main research interests include formal verification of stochastic and real-time systems, probabilistic model checking and machine learning.

**Copyrights**
©2019 The author(s). This is an open access article distributed under the terms of the Creative Commons Attribution (CC BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, as long as the original authors and source are cited. No permission is required from the authors or the publishers.

**How to cite this paper:**
M. Mohagheghi, "Faster convergence of modified policy iteration for quantitative verification of Markov decision processes," Journal of Electrical and Computer Engineering Innovations, 7(1): 113-122, 2019.

**DOI:** 10.22061/JECEI.2020.5395.216

**URL:** http://jecei.sru.ac.ir/article_1197.html