Machine Learning and Disk-based Methods for Qualitative Verification of Markov Decision Processes

Mohammadsadegh Mohagheghi[1][0000−0001−8059−3691] and Khayyam Salehi[2][0000−0002−3379−798X]

[1] Department of Computer Science, Vali-e-Asr University of Rafsanjan, Rafsanjan, Iran
[2] Department of Computer Science, Shahrekord University, Shahrekord, Iran
mohagheghi@vru.ac.ir

Abstract. State explosion is a well-known challenge of model checking. In this paper, we propose several approaches to improve the standard techniques for verifying the qualitative reachability properties of Markov decision processes. For the first approach, we use two heuristics to reduce the total number of iterations of the standard iterative methods. The second approach uses the secondary hard disks for storing the information of a model and uses the main memory for a back-ward technique on the standard methods. The third approach uses a machine learning technique to classify the state space of a model to the related classes. While this approach does not need any memory overhead, its running time is much less than the running time of the standard approaches and can be used to cope with the state explosion problem.

Keywords: Probabilistic model checking, Qualitative reachability, Markov decision processs, Machine learning.

1 Introduction

Probabilistic model checking is a formal verification technique, used to verify the correctness of computer systems with some probabilistic aspects [1]. Markov decision processes (MDPs) are a well-known formalism for modeling systems that have both non-deterministic and probabilistic behaviors [11, 9, 18]. The main challenge of model checking is the state space explosion problem. In this problem, the number of states of the model grows exponentially in the number of modules of the underlying system [1, 6]. Several approaches are proposed to cope with this problem. Symbolic model checking is a well-established approach that uses a compact data structure to store the information of a model. Binary decision diagram (BDD) is one of the most successful compact data structures that is widely used in successful model checking tools [6]. Multi-terminal BDD (MTBDD) has been developed and widely used for verification of probabilistic systems [17].

However, iterative computations required in probabilistic model checking are computationally expensive where BDD-based data structures are used. As an alternative, sparse matrix representation is used as an explicit approach that stores the information of state-values and transition probabilities of the model. A hybrid approach uses MTBDD for storing the information of transitions and an explicit array for state-values [17].

In this paper, we focus on the verification of qualitative reachability properties of MDP models and propose several techniques to improve the performance of standard iterative algorithms for this class of properties. A qualitative property states that the extremal (minimal or maximal) probability of reaching a set of states is 0 or 1 [1, 9]. Some graph-based computations are needed to compute the related set of states. While BDD-based implementation for computing qualitative reachability properties in MDPs is available in several states of the art model checkers [10, 12], their running time is a main drawback of the BDD-based techniques. To alleviate this drawback, we propose several improvements for computing qualitative reachability properties of MDPs. The forward and backward approaches for the standard iterative algorithms for computing qualitative reachability properties are compared in [16]. While the backward approach is much faster than the forward one, its memory overhead is the main obstacle for this approach. As the first contribution of the paper, we propose several improvements to the forward approach. The general idea of these approaches is to reduce the number of iteration of the standard algorithms. We use the sparse explicit data structure for implementations of these improvements and consider those models that are small enough to load overall information in the main memory. Furthermore, we propose two approaches in the case of larger models that can not be loaded in the main memory. As the second contribution of the paper, we use a disk-based approach to store the transition probability information of a model and load its graphical information to the main memory. In this case, we apply the backward approach to reduce the running time. As the third contribution, we apply machine learning approaches to compute the set of states for which the extremal reachability probability is 0 or 1. For each class of MDP models, we consider several small models for training a classifier and use it to classify the states of a large model to the related sets. Although this technique does not guarantee to find the exact sets, our experiments show promising results with nearly zero error. The main benefit of this approach is its scalability for very large models.

1.1 Related Works

The first algorithms for qualitatively verifying the reachability properties of MDP models were proposed in [7]. The time complexity of these algorithms is linear or quadratic in the size of the model for the minimal or maximal reachability probabilities, respectively [1, 7]. BDD-based and sparse explicit implementations of these algorithms have been proposed and studied in [17]. The worst-case time complexity for qualitative verification of maximal reachability probability is improved in [4]. Decomposing an MDP to its strongly connected

components (SCCs) and computing reachability probabilities of the states of each SCC has been proposed in [13]. In this case, qualitative verification is performed implicitly for each SCC. Although this technique presents promising results in the running time, its focus is on the computations of quantitative reachability properties and does not guarantee the correctness for the results in the case of qualitative verification. The backward and forward approaches for the standard algorithms for qualitative verification of MDPs are studied and compared in [15]. The backward approach for the sparse explicit data structure is supported by the STORM model checker [8]. A linear-time heuristic is proposed in [16] to approximate the qualitative reachability properties in the case of maximal probabilities. In Section 3, we use this heuristic to reduce the total number of iterations for the forward approach. Disk-based techniques have been used in [14] for verification of quantitative properties of very large DTMC and MDP models. In [3] a learning-based approach is used for statistical model checking of MDPs. This approach reduces the visited states of a model and is implemented in the explicit engine of PRISM. In [20], machine learning is employed to approximate the probabilities of an unknown MDP model. In [19] machine learning is used to extrapolate the optimal actions of MDP models. Although the proposed approach in [19] proposes some promising results, it is limited to only a case study and its generality for other models is not explained.

2 Preliminaries

In this section, the main concepts and definitions of probabilistic model checking and qualitative verification of reachability properties are reviewed. More details about the probabilistic model checking and its variant are available in [1, 9, 6]

2.1 Probability Distribution and Markov Decision Process

A probability distribution on a finite set X is defined as a function $Pr : X \to [0, 1]$ where $\sum_{x \in X} Pr(x) = 1$. We use $D(x)$ to denote the set of all probability distributions on X.

Definition 1. Markov Decision Processes. A Markov Decision Process (MDP) is a tuple $M = (S, s_0, Act, P, G)$ where:

- S is a finite set of states.
- $s_0 \in S$ is the initial state.
- Act is a finite set of actions. For every state $s \in S$, one or more elements of Act are defined as enabled actions and are shown by $Act(s)$.
- $P : S \times Act \to D(s)$ is a probabilistic transition function. We use (s, α, s') for a transition from the source state s to the destination s' by the action α and $P(s, \alpha, s')$ for the probability of this transitions. For each state $s \in S$ and enabled action $\alpha \in Act(s)$ a probability distribution is defined.
- $G \subset S$ is the (non-empty) set of goal states.

The size of X is defined as the number of its states and transitions and is shown by $|M|$. MDPs are widely used in mathematics, engineering, economics, and management to model decision making under uncertainty in stochastic environments [11, 18]. The actions of MDPs are used to model non-deterministic concepts of a system and its related environment. The probabilistic transition function is used to model the stochastic and probabilistic behavior of the environment.

In formal verification, a high level language is used for modeling computer systems. Every model includes one or several modules. Some variables in a bounded domain are defined for each module. A possible valuation of the variables in a module presents a state of the module for the system. The PRISM language is used as the standard modeling language for probabilistic systems [12]. The interpreter of the model checker constructs the resulting MDP (states, actions, and transition probabilities) according to the definitions of its modules. More details about the PRISM language are available in [17, 12].

2.2 Semantic of MDP

For an MDP M and one of its states $s \in S$, the MDP performs a transition in two steps:

- It non-deterministically selects an enabled action $\alpha \in Act(s)$.
- It randomly selects the destination state $s' \in S$ with probability $P(s, \alpha, s')$.

For any state $s \in S$ and enabled action $\alpha \in Act(s)$, we use $Post(s, \alpha)$ for the set of α-successors of s, $Post(s)$ for all its possible successors, and $Pre(s)$ for its predecessor states:

$$Post(s, \alpha) \doteq \{s' \in S \mid P(s, \alpha, s') > 0\}, \tag{1}$$

$$Post(s) \doteq \cup_{\alpha \in Act(s)} Post(s, \alpha), \tag{2}$$

$$Pre(s) \doteq \{s' \in S | s \in Post(s')\}. \tag{3}$$

A path in M shows a possible run and is defined as a non-empty (finite or infinite) sequence of states and actions of the form $\pi = s_0 \overset{\alpha_0}{\rightarrow} s_1 \overset{\alpha_1}{\rightarrow} ...$ where for every $i \geq 0$, we have $s_i \in S$ and $\alpha_i \in Act(s_i)$ and $s_{i+1} \in Post(s_i, \alpha_i)$. We use $Path_s$ to denote the set of all infinite paths of M that start from a state $s \in S$. For the subset of finite paths of $Path_s$ we use $FPath_s$. We also use $\pi(i)$ to denote the $(i+1)-th$ state in the path π, i.e., $\pi(i) = s_i$. To reason about the probabilistic behavior of an MDP M, one should resolve its non-deterministic choices. To do so, a deterministic function is used that maps an action $\alpha \in Act$ to each path of M. This function is called adversary (scheduler or policy) and is defined as [9, 6] :

Definition 2. (Adversary). An adversary of an MDP M is a function $\sigma : FPath_s \rightarrow Act$ that for every finite path $\pi = s_0 \overset{\alpha_0}{\rightarrow} s_1 \overset{\alpha_1}{\rightarrow} ... \overset{\alpha_{i-1}}{\rightarrow} s_i$ selects an enabled action $\alpha_i \in Act(s_i)$. An adversary σ is called memory-less if it depends only on the last state of the path. We use $FPath_S^\sigma$ for the set of all finite paths of the form $s \overset{\sigma(s)}{\rightarrow} s_1...s_{n-1} \overset{\sigma(s_{n-1})}{\rightarrow} s_n$ that start from s.

In this paper, we only use memory-less adversaries that are sufficient for analysis of reachability properties. We use Adv_M for the set of all (memory-less) adversaries of M.

2.3 Reachability Probabilities

The main class of properties that are used in the probabilistic model checking of MDPs includes the reachability properties. The computation of these properties is usually reduced to the computation of the reachability ones. For MDPs, a reachability probability is defined as the extremal (minimal or maximal) probability of reaching a goal state. For an adversary $\sigma \in Adv_M$ and a state $s \in S$, we define $reach_s^\sigma(G)$ as the set of all finite paths, starting from $s \in S$, ending in G, and selecting actions due to the adversary σ. Formally we define:

$$reach_s^\sigma(G) = \{\pi \in FPath_s^\sigma \mid last(\pi) \in G,\ and\ \forall i < |\pi| : \pi(i) \notin G\} \quad (4)$$

where $last(\pi)$ is the last state in π. For an adversary σ and a path $\pi \in FPath_s^\sigma$, a probability measure $pr^\sigma(\pi)$ is defined as the product of probabilities of transitions between the states of π:

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

This probability measure is used to formally define the extremal reachability probabilities. For any state $s \in S$, the minimal and maximal probability of reaching G from s over Adv_M are denoted by $Pr_s^{min}(\Diamond G)$ and $Pr_s^{max}(\Diamond G)$ and are formally defined as:

$$pr_s^{min}(\Diamond G) = inf_{\sigma \in Adv_M} Pr(reach_s^\sigma(G)), \quad (6)$$

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

where $Pr(reach_s^\sigma(G))$ is used for the total probability of reaching G from s under the adversary σ and is formally defined:

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

Graph-based computations are used to detect the set of states of an MDP for which the extremal reachability probability is exactly 0 or 1. The properties that relate to the computation of these states are called qualitative reachability properties. On the other hand, the properties related to the computations of reachability probabilities for the remaining states are called quantitative reachability properties.

2.4 Qualitative Reachability Properties

The following sets are defined for the extremal qualitative reachability probabilities:
$S^0_{min} = \{s \in S | Pr^{min}_s(\Diamond G) = 0\}$, $S^1_{min} = \{s \in S | Pr^{min}_s(\Diamond G) = 1\}$,
$S^0_{max} = \{s \in S | Pr^{max}_s(\Diamond G) = 0\}$, $S^1_{max} = \{s \in S | Pr^{max}_s(\Diamond G) = 1\}$.

The computation of these sets are necessary for qualitative verification of MDPs. For quantitative reachability probabilities, there is no need to have these sets but they can improve the precision of the computed values [9]. For the case of extremal expected rewards (where the maximal or minimal expectation of accumulated rewards until reaching a goal state is considered), the computations of the S^0_{max} and S^1_{max} are needed to guarantee the convergence of the computed values. Graph-based iterative methods, which only consider the graphical structure of the underlying models are used to compute these sets. Iterative numerical methods are used for the computation of quantitative properties [1, 6]. Algorithm. 1 and Algorithm. 2 describe the standard iterative algorithms for computing the S^0_{max} and S^1_{max} [9].

Algorithm 1 The standard algorithm for computing S^0_{max}

Input: MDP $M = (S, s_0, Act, P, G)$
Output: The set $S^0_{max} = \{s \in S \mid pr^{max}_s(\Diamond G) = 0\}$

```
1: R := G;
2: repeat
3:     R' := R;
4:     R := R' ∪ {s ∈ S | post(s) ∈ R'};
5: until R ≠ R';
6: return S \ R;
```

In each iteration, the Algorithm. 1 adds a state $s \in S/R$ to R if at least one state $s' \in Post(s)$ has been added to R in the previous iteration. The algorithm continues until the case where no new state is added to R. The last set R is the fixed point of the computations and contain all states that can reach to at least one of the goal states. For each $s \in R$ we have $Pr^{max}_s(\Diamond G) > 0$. The remaining states (S/R) are those that can not reach to any goal states and correspond to the definition of S^0_{max}.

For Algorithm. 2, two nested loops are used to compute the S^1_{max} set. The outer loop starts with S and iteratively removes those states $s \in S$ for which $Pr^{max}_s(G) < 1$. For these outer loop iterations, a sequence of R_i sets are induced where $S = R_0 \subset R_1 \subset ... \subset R_n = S^1_{max}$. To compute each R_i set the inner loop starts from G (line 6 of Algorithm. 2) and iteratively adds each state $s' \in S$ to R_i if s' can reach to one of the goal states with probability one via the states of R_{i-1}. The remaining states after reaching the fixed point (the states in S/R_n) do

Algorithm 2 The standard algorithm for computing S^1_{max}

Input: MDP $M = (S, s_0, Act, P, G)$
Output: The set $S^1_{max} = \{s \in S \mid pr^{max}_s(\Diamond G) = 1\}$

1: $R := S$;
2: **repeat**
3: $R' := R$;
4: $R := G$;
5: **repeat**
6: $R'' := R$;
7: $R := R'' \cup \{s \in S \mid \exists \alpha \in Act(s).(post(s, \alpha) \subset R' \wedge post(s, \alpha) \cap R'' \neq \emptyset)\}$;
8: **until** $R \neq R''$;
9: **until** $R \neq R'$;
10: **return** R;

not belong to S^1_{max}. The correctness of the standard algorithms for qualitative reachability properties are available in [1, 7].

For an MDP M with $|S|$ states, the time complexity of algorithms 1 and 2 are in $O(|M|)$ and $O(|M| \cdot |S|)$ respectively. In these cases, we suppose that for each state $s \in R'$ the algorithms can determine any states of $Pre(s)$ in a constant time [1]. For this purpose, an standard approach is to restore the information of the model in a backward approach, i.e. for each state $s \in S$ the method should restore the list of states in $Pre(s)$.

The main drawback of the backward approach is its memory overhead which limits it to small models. On the other hand, the forward implementation of these algorithms is proposed and used in PRISM [12]. The forward approach need not any additional memory, but may increase the number of iterations and the running time of the computations. The memory consumption and running time of the forward and backward approaches for the standard algorithms of qualitative verification of MDPs are compared in [16]. In this paper, we only consider the maximal qualitative reachability properties. The standard algorithms for the minimal case can be found in [1, 9]. All proposed techniques of this paper can also be applied to the case of minimal reachability properties.

3 The Proposed Methods for Accelerating Qualitative Verification of MDPs

In this section, we propose three approaches to improve the performance of the standard algorithms for computing qualitative reachability properties in MDPs.

3.1 Reducing Iterations of the Forward Approaches

We use two heuristic techniques to reduce the number of iterations of the forward implementation of the standard algorithms for qualitative verification of

reachability properties in MDPs. Although the idea of these heuristics has been used in the previous works for other classes of properties for MDPs, to the best of our knowledge, they were not applied for computing qualitative reachability properties.

The first heuristic (called h1 for the remaining parts) is to merge separate sets of the standard algorithms to one set. One drawback of the algorithms 1 and 2 (which are proposed in [1, 9]) is that they separate the computed sets in the current iterations from the set of the previous iteration. The Algorithm 1 uses two sets R and R' while it can keep all added states to one set R. In the latter case, any added state in each iteration can be used to add (probably) some other states in the same iteration. In Algorithm 2, the R and R'' of the inner loop can be merged to one set.

The second heuristic (called h2) reduces the total number of iterations using a good state ordering for selecting the remaining states of each algorithm. Several state ordering techniques have been proposed in the previous works for quantitative reachability properties [5] or discounted accumulated rewards [21]. In both cases, the proposed methods use the $Pre(s)$ states of each state s. To avoid the memory overhead of the information of the Pre sets, we use a forward approach for state ordering. The idea of this approach is to consider the graph of the model in a forward manner and perform a breadth-first search on the graph. The visited states of the model (nodes of the related graph) are stored in a list and the iterative methods use the list in reverse order. The algorithm of this method is proposed in Algorithm. 3. This algorithm uses an array *StateOrder* to store the selected states. In each step, it selects a state from the beginning of the array and adds its non-selected successor states to the end of the array. The algorithm terminates when all states of the model have been added to the *StateOrder* array. The main advantage of our proposed state ordering is that it does not need to store any additional information of the model and as a result it does not have any memory overhead. Note that the backward approach does not need to compute the state ordering explicitly. It considers states for adding to the R sets in the right way.

The iterative methods should use the states in the *StateOrder* array in reverse order (from end to the beginning of the array). In general, this state ordering can reduce the number of iterations of the iterative methods because it accelerates the propagation of the related information between the states of the model. This state ordering is used in line 4 of Algorithm. 1 and line 7 of Algorithm. 2 where the algorithms update the R sets in each iteration.

3.2 Disk-based Backward Technique for Qualitative Verification

To compensate the memory overhead of the backward approach, the secondary storage can be used. The idea of this approach is shown in Algorithm. 4. It first uses the MTBDD-based approach to construct the model and directly store the information of the model to the hard disk. Furthermore, it computes the information of the backward graph and loads them into the main memory. This structure does not include the information of transition probabilities and is used

Algorithm 3 Forward method for state ordering

Input: MDP $M = (S, s_0, Act, P, G)$
Output: A state ordering array $Stateorder[]$

```
1: l := 0;
2: h := 1;
3: Stateorder[l] := s_0;
4: while l < h do
5:     s := Stateprder[l];
6:     for all s_i ∈ post(s) do
7:         if s_i ∈ Stateorder then
8:             Stateorder[h] := s_i;
9:             h := h + 1;
10:        end if
11:    end for
12: end while
13: return ;
```

to compute the S^0_{max} and S^1_{max} sets. Finally, the information of the remaining states $(S/(S^0_{max} \cup S^1_{max}))$ will be loaded into the main memory if they are needed for quantitative verification. The main advantage of this approach is avoiding any memory overhead while it uses the backward approach for qualitative verification. However, this approach applied only to those models for which the information of the backward graph can be loaded into the main memory.

Algorithm 4 The Disk-base Approach

Input: An MTBDD representation for an MDP $M = (S, s_0, Act, P, G)$
Output: The sets S^1_{max} and S^0_{max}

1: Store the information of M from MTBDD to the hard disk;
2: Compute the backward graph of M and load it to the main memory;
3: Apply the backward technique for computing the sets S^1_{max} and S^0_{max};
4: Free the memory of the backward graph;
5: Load the information of M for the remaining states to the main memory;

3.3 Using Machine Learning for Qualification Verification of MDPs

Machine learning plays an important role in nowadays life. Classification is a act of categorizing a given set of data into classes. In machine learning, classification is supervised learning in which it infers from the data given to it and makes new observations or classifications based on the pattern extracted from training data [2]. There are many algorithms used in classifying new data. Some types of

classification in machine learning can be summarized as Naive Bayes, Stochastic Gradient Descent, K-Nearest Neighbors, Decision Tree, Random Forest, Artificial Neural Network, Support Vector Machine, and so on.

As the main contribution of this paper, we propose a new approach which uses machine learning techniques to verify the qualitative reachability properties of MDPs. Our technique uses the fact that some parameters define the bound of some variable domains of MDP modules. The size of MDPs for a high-level definition changes according to the values of these parameters. In Fig.1 the initial definition of the *Consensus* model is proposed. This model includes four modules ($N = 4$). Each module contains two variables (*pc1* and *coin1*). A *counter* is defined as a global variable. The model has a parameter (K) which determines the range of the *counter* variable. According to the values of this parameter, we can have some models with different size.

```
// COIN FLIPPING PROTOCOL FOR POLYNOMIAL RANDOMIZED CONSENSUS [AH90]
// gxn/dxp 20/11/00

mdp
// constants
const int N=4;
const int K;
const int range = 2*(K+1)*N;
const int counter_init = (K+1)*N;
const int left = N;
const int right = 2*(K+1)*N - N;
// shared coin
global counter : [0..range] init counter_init;
module process1
        // program counter
        pc1 : [0..3];
        coin1 : [0..1];
        // flip coin
        [] (pc1=0)  -> 0.5 : (coin1'=0) & (pc1'=1) + 0.5 : (coin1'=1) & (pc1'=1);
```

Fig. 1. A high-level definition of an MDP model in PRISM environment

In general and for each class of models, we can consider small values for its parameters and have some small MDPs. A standard method can compute the qualitative reachability properties of these small models. As a result, the set S of state of a model are classified into the S^0_{max}, S^1_{max}, and $S^?_{max} = S/(S^0_{max} \cup S^1_{max})$ sets. We use the small models for training a classifier. Furthermore, it is used to classify the state space S of a large model to its S^0_{max}, S^1_{max}, and $S^?_{max}$ sets. We consider the variables and parameters of each model as its properties and its states as the samples for training. Also, for each variable with parametric domain (like the *counter* variable in Fig.1), we consider a property as the difference between its domain bound and its values. This additional property helps the classifier to improve its performance.

4 Experimental Results

To show the feasibility and applicability of the proposed approaches, we consider several test case models. These test cases are selected from the PRISM standard case studies and are widely used in the literature [9]. For the forward approach, we implement our improvements in the PRISM model checker. We use its sparse engine, which is implemented in C. We use a machine with a Core-i7 Intel processor with 8GB of RAM. In Table 1, the experiments for computing S^0_{min} and S^0_{max} sets are demonstrated. The first column of the table presents the model's name. For *Consensus*, *Zeroconf*, and *Wlan* classes, we consider a maximal reachability property and for the *Firewire* class, we consider a minimal reachability property. More information about these classes are available in [12]. The MDP size of each model is the number of its states and transitions.

We consider the running time and the number of iterations of Algorithm. 1 for the MTBDD-based method as is used in PRISM. We also demonstrate the running time of the sparse implementation of this algorithm. For our proposed heuristics, we report the running time and the number of iterations of the forward method with these heuristics. The running time in all tables are in seconds.

Table 1. Case study models, number of iterations and running times for computing S^0_{min} and S^0_{max} sets

Model	Parameter	MDP	MTBDD		sparse	forward with h1		forward with h2	
(parameters)	values	Size	time	iters	time	time	iters	time	iters
	6 , 18	604K	2.25	979	1.1	.47	511	.01	7
Consensus	8 , 18	2378K	10.9	1305	6.3	4.3	681	.1	8
N,K	10 , 10	4725K	14.8	911	7.5	7.1	477	.3	9
	10	9787K	6	140	3.7	1.6	98	2.14	12
Zeroconf	14	14.4M	7.5	156	3.9	2.14	110	4.9	12
K	18	17.8M	10.1	172	8.6	122	4.9	12	4.4
e	4,2500	7052K	18.4	207	.7	.26	76	.8	65
Wlan	5, 1000	8905K	10.4	399	1.24	.44	140	1.39	129
(K,ttm)	6, 200	18264K	17	783	2.2	.73	268	2	257
firewire	12,600	28.35M	30	373	13.6	12.4	368	3.1	5
(delay,	30,400	115M	161	409	52.7	48	404	9.7	4
deadline)	30,600	227.7M	345	409	110.3	100.5	404	25.3	4

In Table. 2, we show the running time for the standard iterative method and our heuristics for computing the S^1_{min} and S^1_{max} sets. For Disk-based technique, the running times include the time for writing a model in the secondary disk, constructing the backward graph, computing qualitative reachability properties using the backward graph, and finally loading the information of the remaining part of each model (for the $S^?$ sets) to the main memory. For the case of S^0 sets, all running times are less than 1 second and we do not separate their running times.

Table 2. Case study models, number of iterations and running times for computing S^1_{min} and S^1_{max} sets

Model	Parameter	MTBDD		sparse	forward with h1		forward with h2		Disk-based
(parameters)	values	time	iters	time	time	iters	time	iters	time
	6, 18	323	94.6K	141	85	56.5K	2.6	1118	7.2
Consensus	8, 18	1513	169K	1934	1359	101K	35.3	1788	17.7
(N,K)	10, 10	1738	88K	1767	1551	52.5K	113	1486	24
	10	121	2357	104	30.1	1200	66.1	213	4.9
Zeroconf	14	184.3	2525	74.9	36.6	1385	52.4	209	5.1
(K)	18	128.4	2665	97.9	58.6	1481	107.8	192	6.5
s	4 , 2500	58.3	414	9.2	3.3	226	3.1	65	2.2
Wlan	5, 1000	30.8	798	17.2	5.5	418	4.1	206	2.7
(K,ttm)	6, 200	39.1	1566	29.2	9.1	802	8	256	2.9
firewire	12,600	33	261	11.2	9.7	255	.96	4	6.3
(delay,	30,400	74.2	169	14.3	11.4	156	2.1	3	17.9
deadline)	30,600	189	253	54.6	51.5	244	5.2	4	31.4

The results of Table 1 and 2 show that our proposed heuristics for the forward implementation of the standard algorithms improves their performance for computing qualitative reachability properties of MDPs. The best results are for the Consensus and *firewire* cases. For the *wlan* and *zeroconf* models, our heuristics reduce the number of iterations by one or two orders of magnitude. In most cases, the Disk-based method outperforms the others. Note that this approach is more useful for large models, where the information of a model cannot be loaded to the main memory.

To apply machine learning for computing qualitative reachability properties of MDPs, we use Scikit-learn package of Python. For each class of models, we consider two or three small models for training and use decision tree classifier of Python. For each training MDP samples, PRISM is used to separate the state to the S^1_{max}, S^0_{max} and $S^?_{max}$ classes. We use the *exportstates* PRISM command to store the state-property information of states to related files. The inputs to the training phase are the state-separed and state-property files. To test the precision of this technique, we consider several test cases for each class of models. The results of this technique are proposed in Table. 3.

To compare the running time of the proposed methods for computing the S^1_{max} sets we consider several models of the *Consensus* example with N=6 and different values for the K parameter in range [20,100]. The results are proposed in Fig. 2 where the horizontal axix shows the value of K and the vertical axis shows the running time in logarithmic mode. The results show that our proposed heuristics reduce the running times by at leat two orders of magnitude.

5 Conclusion and future work

In this paper, we propose several techniques to improve the performance of the iterative computations for the qualitative reachability properties of MDPs. We

Table 3. Experimental results of the proposed machine learning approach for computing S^0_{max} and S^1_{max} sets

Model (parameters)	Param value	Sample parameters	Reading samples time	Training time	Test parameter	test time	Precision iters	False detects
Consensus	6	K=3,4	.9	.1	K = 50	82	100%	2
N	8	K=3,4	2.4	.27	K = 35	17.1	100%	2
Zeroconf	-	K=2,3	6.2	.62	K = 14	2.14	99.8%	26,070
Wlan	5	ttm=50,100	43	5.9	900	18.7	100%	0
(K)	6	ttm=50,100	89.5	11.3	600	26.5	100%	0
firewire	12	ddl=200,250	7.2	2.2	ddl=800	57	100%	0
(delay)	30	ddl=200,250	48.4	11	ddl=600	145.8	100%	0

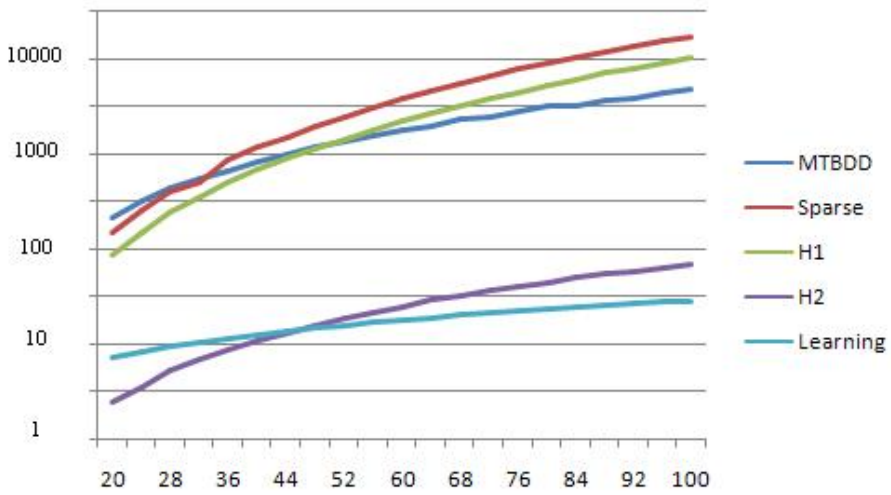

Fig. 2. The running times for computing S^1_{max} sets for the *Consensus* case studies.

first propose two heuristics for the forward implementation of the standard algorithms. The running times of our experiments show promising results for these heuristics. Because of the memory limitation of the explicit methods, we propose a disk-based method to save memory for graph-based computations. As the main contribution, we propose a method using machine learning to classify the state space of a model to the related classes. The experiments show promising results for this technique. For future works, we plan to extend our machine learning method to other problems of probabilistic model checking. As an example, this technique can be used to approximate the optimal policy of a large MDP model, which can be used for improving the current techniques for statistical model checking.

References

1. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
2. Bishop, C.M.: Pattern recognition and machine learning. springer (2006)
3. Brázdil, T., Chatterjee, K., Chmelik, M., Forejt, V., Křetínskỳ, J., Kwiatkowska, M., Parker, D., Ujma, M.: Verification of markov decision processes using learning algorithms. In: International Symposium on Automated Technology for Verification and Analysis. pp. 98–114. Springer (2014)
4. Chatterjee, K., Henzinger, M.: Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification. In: Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms. pp. 1318–1336. SIAM (2011)
5. Ciesinski, F., Baier, C., Größer, M., Klein, J.: Reduction techniques for model checking markov decision processes. In: 2008 Fifth International Conference on Quantitative Evaluation of Systems. pp. 45–54. IEEE (2008)
6. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R.: Handbook of model checking, vol. 10. Springer (2018)
7. De Alfaro, L.: Formal verification of probabilistic systems. Ph.D. thesis, Stanford University (1997)
8. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A storm is coming: A modern probabilistic model checker. In: International Conference on Computer Aided Verification. pp. 592–600. Springer (2017)
9. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: International School on Formal Methods for the Design of Computer, Communication and Software Systems. pp. 53–113. Springer (2011)
10. Hahn, E.M., Li, Y., Schewe, S., Turrini, A., Zhang, L.: IscasMC: a web-based probabilistic model checker. In: International Symposium on Formal Methods. pp. 312–317. Springer (2014)
11. Katoen, J.P.: The probabilistic model checking landscape. In: Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science. pp. 31–45 (2016)
12. Kwiatkowska, M., Norman, G., Parker, D.: Prism 4.0: Verification of probabilistic real-time systems. In: International conference on computer aided verification. pp. 585–591. Springer Berlin Heidelberg (2011)

13. Kwiatkowska, M., Parker, D., Qu, H.: Incremental quantitative verification for markov decision processes. In: 2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN). pp. 359–370. IEEE (2011)
14. Mehmood, R.: Disk-based techniques for efficient solution of large Markov chains. Ph.D. thesis, University of Birmingham Birmingham (2004)
15. Mohagheghi, M.: Improving pre-computation for verification of markov decision processes. International Journal of New Technologies in Science and Engineering **5**(5) (2018)
16. Mohagheghi, M., Salehi, K.: Improving graph-based methods for computing qualitative properties of markov decision processes. Indonesian Journal of Electrical Engineering and Computer Science **17**(3), 1571–1578 (2020)
17. Parker, D.A.: Implementation of symbolic model checking for probabilistic systems. Ph.D. thesis, University of Birmingham (2003)
18. Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons (2014)
19. Rataj, A., Woźna-Szcześniak, B.: Extrapolation of an optimal policy using statistical probabilistic model checking. Fundamenta Informaticae **157**(4), 443–461 (2018)
20. Tappler, M., Aichernig, B.K., Bacci, G., Eichlseder, M., Larsen, K.G.: L^*-based learning of markov decision processes. In: International Symposium on Formal Methods. pp. 651–669. Springer (2019)
21. Wingate, D., Seppi, K.D.: Prioritization methods for accelerating mdp solvers. Journal of Machine Learning Research **6**(May), 851–881 (2005)