

Learning Methods for Statistical Model Checking of DTMC

Mohammadsadegh MOHAGHEGHI¹, Khayyam SALEHI^{2,*}

¹ *Department of Computer Science, Vali-e-Asr University of Rafsanjan, Rafsanjan, Iran*

² *Department of Computer Science, Faculty of Mathematical Sciences, Shahrekord University, Shahrekord, Iran*

e-mail: mohagheghi@vru.ac.ir, kh.salehi@sku.ac.ir

Received: January 2025; accepted: February 2026

Abstract. Statistical model checking offers an alternative to traditional model checking for large stochastic systems, addressing state space explosion and approximating quantitative properties. This paper proposes machine learning approaches using decision trees to approximate zero-reachability states, offering both computational efficiency and interpretability. Statistical analysis is used as an alternative approach to establish simulation run length bounds to control computation errors. Experimental results across standard Markov models demonstrate that our decision structures maintain high correctness (99% in most cases), reduce runtime, and have minimal memory overhead. Even when some methods show limitations, alternative approaches within our framework yield effective results.

Key words: statistical model checking, machine learning, decision tree classifiers, probabilistic systems, reachability probability.

1. Introduction

Computer system failures can have severe consequences, potentially risking human lives. While testing is useful, it cannot exhaustively verify system correctness (Clarke *et al.*, 2018). Formal methods offer a more comprehensive approach, using mathematical techniques to prove system property under all circumstances. Two main types of formal methods are theorem proving, which requires expert knowledge, and model checking, which automatically verifies system behaviour against desired properties. This paper focuses on model checking.

Systems can be modelled using various formalisms, including transition systems. The probabilistic behaviour of systems is essential. To do so, transition systems can be extended to support the probabilistic behaviour. Markovian processes, particularly Markov decision processes and discrete-time/continuous-time Markov chains, are well-established for this purpose. Properties are typically specified using temporal logics such as Linear Temporal Logic (LTL) or Computational Tree Logic (CTL). For probabilistic systems,

*Corresponding author.

Probabilistic CTL (PCTL) is commonly used (Baier and Katoen, 2008). The analysis of probabilistic system behaviour is termed Probabilistic Model Checking (PMC).

Solving (probabilistic) model checking often requires exploring all reachable states, leading to state space explosion as model complexity increases exponentially with variables. This can exceed memory capacity for large systems (Legay and Viswanathan, 2015). While numerical computations offer a solution for probabilistic systems, they can exacerbate the problem due to lengthy iterative computations (Mohagheghi *et al.*, 2020). Statistical Model Checking (SMC), utilizing simulation and sampling, emerges as a promising approach to address these challenges (Younes *et al.*, 2006).

Statistical Model Checking (SMC) offers an alternative approach to verify probabilistic systems. It uses finite simulation runs and statistical techniques to estimate if samples satisfy or violate specifications. This approach relies on the Monte Carlo simulations to provide probably approximately correct results, which are sufficient in most cases of system verification (Legay *et al.*, 2019; Ashok *et al.*, 2019). It can verify both black-box and infinite-state systems (Aichernig and Tappler, 2017), as well as white-box systems (Ashok *et al.*, 2020), by simulating entire or partial system executions. SMC extends to more complex properties, including hyper-properties (Wang *et al.*, 2021). For overcomplicated or undecidable problems, SMC often emerges as one of the most viable solutions.

Statistical Model Checking has been extended to systems with probabilistic behaviours, modelled by Markovian structures (Legay and Viswanathan, 2015; Brázdil *et al.*, 2014; Henriques *et al.*, 2012; Gros *et al.*, 2020). As a memory-efficient method, SMC approximates underlying properties of given states. Its applications cover various domains, including safety critical and cyber-physical systems (Clarke and Zuliani, 2011; Cleaveland *et al.*, 2022; Qin *et al.*, 2022; Kim and Kim, 2012), computer networks (D’argenio *et al.*, 2022), software design (Bao *et al.*, 2017; Karna *et al.*, 2018), system of systems (Song *et al.*, 2017), process mining (Casaluce *et al.*, 2022), transportation systems (Bertrand *et al.*, 2019) and biological applications (David *et al.*, 2015; Zuliani, 2015).

SMC simulates a set of runs and checks whether each of them reaches a goal state or not. To terminate each simulation run, SMC method should reach a goal state, or detect Bottom Strongly Connected Components (BSCCs) without goal states. While standard algorithms can compute BSCCs in linear time (Baier and Katoen, 2008), they require complete model information, conflicting with SMC’s goal of addressing state explosion. On-the-fly BSCC detection approaches have been proposed in Brázdil *et al.* (2014), Daca *et al.* (2017), but these may require longer runs to enhance result confidence, potentially increasing overall SMC runtime.

Contributions. Efficient SMC requires a low-overhead and high-confidence approach to detect states with no path to goal states (denoted as S^0). This paper proposes new machine learning-based methods to approximate S^0 , maintaining SMC’s memory efficiency by constructing a classifier (as a black-box) for verification. The first approach considers a limited set of simulation runs and applies the available on-the-fly methods to detect whether each run reaches a goal state or a state in S^0 . It then labels the traversed states as 0 or 1 for the training step and constructs a decision tree classifier based on them. To diversify the sample set, the second approach considers a set of randomly selected states

and applies a statistical test to approximate which states more probably belong to S^0 . In the third approach, we apply a statistical analysis to provide an approximation for the upper-bound of required successive steps until reaching a goal state. This approach brings low overhead to the computations and can mathematically limit the probability of false predictions. Briefly, our contributions are:

- applying decision trees to detect when each simulation run reaches a BSCC and terminate the run,
- proposing several techniques for early detection of runs that will eventually reach BSCCs, in order to reduce the running time,
- employing binary decision diagrams to store the information of detected states in a compact way, in order to reduce detection errors,
- providing a bound on the number of simulation runs using statistical analysis, and
- demonstrating the applicability and feasibility of the proposed approaches in several case studies.

Paper outline. This paper is structured as follows: Section 2 covers preliminaries, Section 3 reviews SMC methods, and Section 4 presents our our main contributions, including approaches for detecting S^0 with examples to describe the application of each approach. Section 5 describes the experimental validation. Section 6 and 7 discuss related work and threats to validity, respectively. Section 8 concludes the paper and outlines some future work.

2. Preliminaries

For a finite set X , a probability distribution is defined as a mapping $P : X \rightarrow [0, 1]$, such that $\sum_{x \in X} P(x) = 1$. We use $D(X)$ to denote the set of all probability distributions on X . Transition systems are used in formal verification for modelling computer systems at the design level. For modelling systems with non-deterministic and stochastic aspects, Markovian processes are used (Baier *et al.*, 2019).

2.1. Stochastic Systems

To model Stochastic systems, it requires presenting the formalism to support the probabilistic transition. Discrete-Time Markov Chain (DTMC) is formally defined as follows:

DEFINITION 1 (DTMC). A DTMC \mathcal{M} is tuple (S, P, s_0, G, AP) where:

- S is a set of states,
- $P : S \times S \mapsto [0, 1]$ is a transition probability function, s.t. $\forall s \in S, \sum_{s' \in S} P(s, s') = 1$,
- $s_0 \in S$ is an initial state,
- $G \subseteq S$ is the set of goal states,
- AP is a set of atomic propositions in each state.

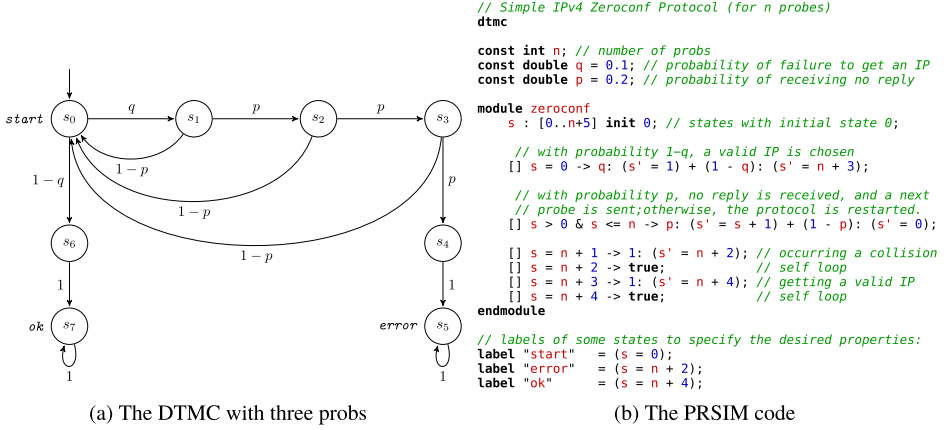


Fig. 1. The Zeroconf protocol.

In this paper, states $s \in S$ are not merely abstract indices but are defined by a valuation of a set of model variables $V = \{v_1, v_2, \dots, v_k\}$. Thus, each state s can be represented as a vector $\mathbf{v}_s = (\text{val}(v_1), \text{val}(v_2), \dots, \text{val}(v_k))$, which serves as the feature vector for our learning algorithms. We define a feature mapping function $F : S \rightarrow \mathbb{R}^k$ that maps each state to its corresponding valuation.

For any state $s \in S$, we use $\text{post}(s)$ for the set of all immediate successor state of s and define it as $\text{post}(s) = \{s' \in S \mid P(s, s') > 0\}$. A path in a Markov chain is a (finite or infinite) sequence of states with a positive transition probability function. That is, $\pi = s_0 s_1 s_2 s_3 \dots$ is a path in DTMC such that $\forall i \geq 0: s_i \in S, P(s_i, s_{i+1}) > 0$. We use $\pi[i]$ to denote the $(i + 1)$ -th state of the path, i.e. $\pi[0]$ for the first state of π , and so on. The probability of a finite path is calculated by multiplying the corresponding transition probability function, e.g. $Pr(\pi) = P(s_0, s_1) \times P(s_1, s_2) \times P(s_2, s_3) \times \dots$. For a set of infinite paths, a probability measure is defined on the corresponding set of cylinder set (Baier and Katoen, 2008).

EXAMPLE 1. Consider a simple IPv4 *Zeroconf* communication protocol (Kwiatkowska *et al.*, 2006) for automatic IP address configuration. It is designed to provide mutual communication. The protocol, represented by the DTMC in Fig. 1a, demonstrates the process of acquiring a valid IP address. Starting from the initial state s_0 , a device randomly selects an IP address. The probability q represents the chance that this address is already in use by another device. If a conflict occurs, the protocol enters state s_1 , from which it can either send a probe (with probability p), moving to s_2 , or restart the process (with probability $1 - p$), returning to s_0 ($P(s_1, s_0) = 1 - p$). This probing sequence can repeat up to three times (states s_2 and s_3) before either successfully acquiring an address (transitioning to s_6 and then s_7) or failing after all attempts (moving to s_4 and then s_5). A typical candidates for features in each vector of values in a state are the programme counter and integer values controlling the number of probes to be sent in the protocol. *start*, *error*, and *ok* are atomic propositions valid in states s_0 , s_5 , and s_7 , respectively.

Path $\pi = s_0s_1s_0s_6s_7 \dots$ represents a specific sequence of events occurring, such as encountering a conflict, restarting once, and then successfully acquiring an address. It starts in initial state s_0 with atomic proposition `start` and meets the atomic proposition `ok`. Its probability equals to $Pr(\pi) = P(s_0, s_1) \times P(s_1, s_0) \times P(s_0, s_6) \times P(s_6, s_7) \dots = q \times (1 - p) \times (1 - q) \times 1 = q(1 - q)(1 - p)$.

A *strongly connected component* (SCC) of a DTMC is a non-empty set $C \subseteq S$ where for each pair of states s and $s' \in C$ there is a path from s to s' . A set $C \subseteq S$ is a *bottom strongly connected component* (BSCC) if C is a maximal SCC and for each $s \in C$ and $s' \in S \setminus C$ we have $P(s, s') = 0$. For the DTMC of Example 1, $\{s_5\}$ and $\{s_7\}$ are only the two non-trivial BSCCs. More details about probabilistic model checking, reachability probabilities, and the theory behind them are available in Baier and Katoen (2008), Baier *et al.* (2019).

2.2. Logic

The properties of a Markovian process can be formulated in a Probabilistic Computation Tree Logic (PCTL).

DEFINITION 2 (PCTL). The syntax is formally defined in the following sense:

$$\Phi ::= \phi \mid \Phi_1 \vee \Phi_2 \mid !\Phi \mid \mathbf{P}_J(\psi),$$

where \vee and $!$ are logical operators, ϕ is an atomic proposition or Boolean constant, $J \subseteq [0, 1]$, \mathbf{P} is a probability operator, and ψ is a path formula that imposes a property on the set of paths defined as $\psi ::= X \Phi \mid \Phi_1 U \Phi_2 \mid \Phi_1 U^{\leq t} \Phi_2$.

The semantics of a PCTL formula are defined as follows. Given a DTMC $M = (S, P, s_0, G, AP)$, state $s \in S$, and satisfaction relation \models :

- $s \models \phi$ iff ϕ is valid in s ,
- $s \models \Phi_1 \vee \Phi_2$ iff $s \models \Phi_1$ or $s \models \Phi_2$,
- $s \models !\Phi$ iff ($s \not\models \Phi$),
- $s \models \mathbf{P}_J(\psi)$ iff the sum of probabilities in paths starting at s and satisfying ψ is in J .

The next formula $X \Phi$ holds for a path if Φ is satisfied in the next state of the path, and until formula $\Phi_1 U \Phi_2$ holds for a path iff Φ_2 holds in some state s in the path and Φ_1 holds in all states before s in the path. Path formula $\Phi_1 U^{\leq t} \Phi_2$ denotes the step-bounded of $\Phi_1 U \Phi_2$; that is, it asserts that Φ_2 will hold within at most t steps. (A full description is available in Baier and Katoen (2008).) Another PCTL formula frequently used in the following is $\mathbf{P}_{=a}(\psi)$, which is equivalent to $\mathbf{P}_J(\psi)$ where $J = [a, a]$.

It is worth noting that the above properties are all assertions, that is, we would expect a true or false answer to them. In some probabilistic model checkers such as PRISM (Kwiatkowska *et al.*, 2011), quantitative properties can be directly specified that compute the exact probability instead of verifying whether the probability is in bound J or not. Thus, $\mathbf{P}_{=?}(\psi)$ computes the exact value of sum of probabilities in paths starting at s_0 and satisfying ψ .

EXAMPLE 2. Consider a PCTL formula $\mathbf{P}_{[0.8,1]}[true \ U \ ok]$ and a Markov chain depicted in Example 1. Assume a path $\pi = s_0s_1s_0s_6s_7 \dots$. The property $true \ U \ ok$ holds in path π , i.e. $\pi \models true \ U \ ok$. The reason is that atomic proposition ok is valid in state s_7 , that is $s_7 \models ok$ and $\forall i, s_i \models true$. The PCTL formula is another representation of a reachability probability, as well as the probability of reaching a state with atomic proposition ok (s_7) from s_0 , with values $p = 0.2$ and $q = 0.1$, is 0.99911. That is, $s_0 \models \mathbf{P}_{[0.8,1]}[true \ U \ ok]$.

Qualitative verification computes states where the probability of reaching G is exactly 0 or 1 (Baier *et al.* (2019)). S^1 denotes states with probability 1 of reaching goal states, while S^0 represents those with probability 0. Another important set of states is $S^? = S \setminus (S^0 \cup S^1)$. Formally, for DTMCs, we define $S^0 = \{s \in S \mid s \models P_{=0}[true \ U \ G]\}$, $S^1 = \{s \in S \mid s \models P_{=1}[true \ U \ G]\}$.

Quantitative reachability probabilities in DTMCs and Markov Decision Processes (MDPs) are computed using numerical iterative methods. MDPs generalize DTMCs by incorporating non-deterministic choices between states (Baier and Katoen, 2008). Qualitative verification employs iterative graph-based algorithms (Mohagheghi and Salehi, 2020). While bounded properties offer finer verification in SMC (Budde *et al.*, 2020a; Hahn and Hartmanns, 2016; Arora and Rao, 2020), we focus on unbounded properties.

2.3. The PRISM Modelling Language

High-level modelling languages offer a more efficient alternative to detailed probabilistic models like DTMCs for describing stochastic systems. The PRISM language (Kwiatkowska *et al.*, 2011), supported by PRISM, STORM (Dehnert *et al.*, 2017), and ePMC (Calinescu *et al.*, 2019), is a prominent example. PRISM models consist of modules with bounded integer variables and probabilistic guarded commands that define system evolution. These high-level programmes are translated into low-level probabilistic transition systems, where states are represented by valid variable valuations.

EXAMPLE 3. A PRISM programme for the *Zeroconf* protocol in Example 1 is proposed in Fig. 1b. This programme contains one module that includes a variable s . The upper-bound of s depends on the constant n that is determined by the user. The states of the corresponding DTMC model are defined as value s where each value shows a valid valuation for the programme variable. `start`, `error`, and `ok` are three labels to specify the desired properties.

2.4. Decision Tree Classifier

Decision tree (DT) classifiers are established and well-known supervised learning methods that predict target values using decision rules derived from training data features (Murphy, 2022). Visualized as top-down binary trees, DTs use Boolean predicates at nodes and true/false edges, with decisions at leaf nodes. For a given input, one traverses the tree, evaluating predicates to reach a decision. In this paper, we employ DTs to approximate the set of states that cannot reach a goal state.

2.5. Binary Decision Diagrams

Binary decision diagrams (BDDs), introduced by Bryant (1986), are powerful data structures encoding Boolean functions, widely used in formal verification. As rooted and directed acyclic graphs (DAGs), BDDs compactly represent sets or relations. They consist of leaves labelled 0 or 1, and nodes representing variables, with edges denoting Boolean values true or false. In this paper, BDDs capture set S^0 efficiently, enabling reuse and avoiding recomputation.

3. Statistical Model Checking Approach

For a Markovian process \mathcal{M} and PCTL formula Φ , verifying the satisfiability of Φ against \mathcal{M} using standard numerical methods can be highly challenging, especially for large instances of \mathcal{M} . As an alternative, statistical model checking addresses this problem by using simulation techniques to answer two key questions (Legay and Viswanathan, 2015; Agha and Palmkog, 2018):

1. Qualitative question: whether the probability of the Markovian process \mathcal{M} can satisfy Φ greater or equal to threshold θ or not, formally $Pr(\mathcal{M} \models \Phi) \stackrel{?}{\geq} \theta$,
2. Quantitative question: what is the probability of stochastic systems satisfying Φ , formally $Pr(\mathcal{M} \models \Phi) = ?$.

For both quantitative and qualitative approaches, SMC relies on a set of simulation runs, where each run generates a finite path. If the PCTL formula Φ is satisfied by a simulation run, the outcome is 1; otherwise, it is 0. Let X_i be a discrete random variable following a Bernoulli distribution, representing the outcome of the i -th simulation run. The value of X_i is 1 with probability p and 0 with probability $1 - p$. SMC simulates the system, checks whether it satisfies Φ , and returns 1 or 0 based on the result, where $Pr(X_i = 1) = p$ and $Pr(X_i = 0) = 1 - p$.

Hypothesis testing is typically used to address qualitative questions, where the goal is to test hypothesis $H : p \geq \theta$ against $K : p < \theta$ to determine if $p \geq \theta$ for $p = Pr(\Phi)$. Two types of errors can occur: Type-I (false positive) and Type-II (false negative). Based on these errors, the parameters α and β are defined. If the probability of incorrectly accepting hypothesis K when H is true, denoted by $Pr(K | H)$, is at most α , formally:

$$\begin{aligned} Pr(K | H) &\leq \alpha && \text{(Type-I error),} \\ Pr(H | K) &\leq \beta && \text{(Type-II error).} \end{aligned}$$

It is impossible to guarantee that both of the above are held for low-value α and β . A proper solution to this problem is to consider threshold δ to relax the hypotheses, that is, $H_0 : p \geq \theta + \delta$ and $H_1 : p \leq \theta - \delta$. The region $(\theta - \delta, \theta + \delta)$ is called indifference; that is, the probability p is so close to θ .

Statistical algorithms use the hypothesis testing approach as they need some simulations to check the model statistically. Thus, the number of simulations is important.

Consider a system satisfying property Φ with probability p . After simulating the system, the probability that estimation \hat{p} occurs with error ϵ is less than or equal to δ , formally $Pr(|\hat{p} - p| \geq \epsilon) \leq \delta$. Consequently, the number of simulations is bounded by Chernoff bound (Legay and Viswanathan, 2015) and is equal to

$$n = \left\lceil \frac{\ln 2 - \ln \delta}{2\epsilon^2} \right\rceil. \quad (1)$$

3.1. Challenges

Despite SMC's advantage in attacking state space explosion, simulation termination remains a key challenge. For bounded reachability properties $\mathbf{P}_J[\text{true } U^{\leq t} G]$, termination occurs naturally when simulation length exceeds t . However, unbounded properties pose a greater challenge, as runs may continue infinitely without reaching a goal state, particularly when stuck in a BSCC without any goal states. These states belong to S^0 , but their detection isn't straightforward. For unbounded formulas $\psi = \phi_1 U \phi_2$, approximating $\mathbf{P}_J[\psi]$ first requires approximating S^0 , where $s \in S^0 \iff s \models \mathbf{P}_{=0}[\psi]$.

4. The Proposed Method

A key SMC challenge is determining simulation termination points. For unbounded reachability, termination occurs upon reaching either a goal state or a BSCC without goal states. BSCC detection can be achieved through explicit graph-based methods (Baier and Katoen, 2008) or on-the-fly statistical analysis (Daca *et al.*, 2017). While statistical approaches align with SMC's confidence interval methodology, they can be time-intensive as they run independently without storing previous BSCC detection information.

EXAMPLE 4. For clarification, consider Fig. 2a, which illustrates a large BSCC B that doesn't contain any goal states G . From an initial state s_0 , a simulation run may either

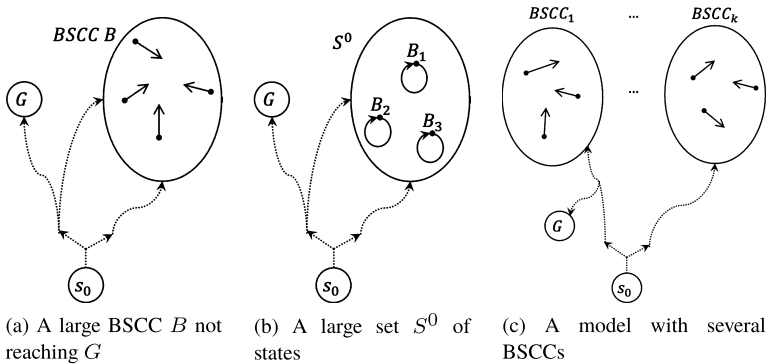


Fig. 2. Some challenges.

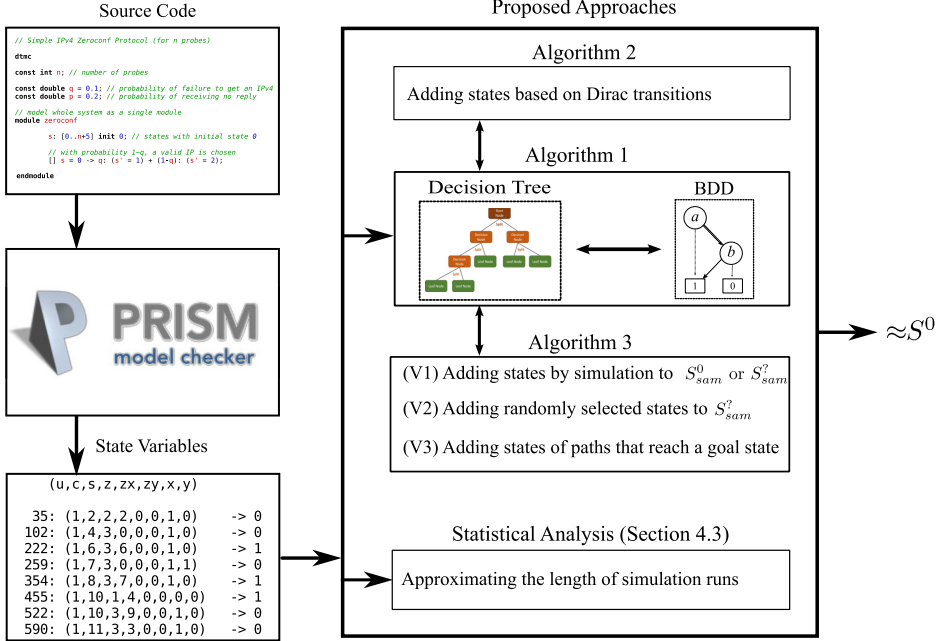


Fig. 3. Overview of the proposed approach.

reach the goal state of G or enter BSCC B . Once a run enters B , it cannot reach G due to B 's strongly connected nature and will continue circulating among B 's states indefinitely. Statistical BSCC detection requires sufficiently long runs to visit all states within B multiple times with high confidence. This requirement for longer runs to ensure accurate BSCC detection significantly impacts the overall runtime of the SMC method.

We propose machine learning and statistical approaches to eliminate redundant BSCC detection computations. Fig. 3 shows our framework where PRISM model checker constructs the induced DTMC of the PRISM code to generate state valuations. Furthermore, the proposed approaches are applied to approximate the set S^0 .

Algorithm 1 employs decision trees and BDDs to store detected BSCC states, enabling early BSCC detection and simulation termination in subsequent runs. This approach is expressed in detail in Section 4.1.

When S^0 contains many non-BSCC states, processing time increases significantly. For example, small BSCCs (B_1 , B_2 , B_3) in Fig. 2b within a large S^0 set often contain states that almost surely reach these BSCCs. Current SMC methods inefficiently process these non-BSCC S^0 states. To address this, we enhance our approach to detect more S^0 states during decision tree construction. We first identify states reaching BSCCs through Dirac transitions with probability 1 (Mohagheghi and Chaboki, 2020), then use statistical methods to find states that almost surely reach BSCCs (Algorithm 2 box).

Decision trees can classify states as belonging to S^0 , but their error rates must be controlled. To improve performance, we use a statistical approach that includes selected

states from $S \setminus S^0$ in the training set (Algorithm 3 box). Details of Algorithms 2 and 3 are provided in Section 4.2. To address a key SMC limitation, we statistically analyse which simulation runs cannot reach goal states (Statistical Analysis of Length). This analysis is detailed in Section 4.3.

4.1. Using Decision Trees for BSCC Detection

This approach applies machine learning classifiers to efficiently decide when to end SMC simulations (Algorithm 1). The algorithm takes a DTMC model D with parameters δ and ϵ for the SMC confidence interval. Based on these parameters, it computes the number of runs n required to satisfy the Chernoff bound (Eq. (1)), and uses n' runs for training the decision tree classifier DT . As a simple heuristic, we set $n' = \frac{n}{10}$ (Line 3 of Algorithm 1).

Algorithm 1: Decision Tree Classifier for S^0

Input: A DTMC D and the feature-based information of its states, δ and ϵ as the parameters for confidence interval.

Output: A decision tree DT as classifier for detecting states in S^0 .

- 1 Initialize S_{sam}^0 and $S_{sam}^?$ to empty sets;
 - 2 Compute n for the number of simulation runs according to Chernoff bound (Eq. (1));
 - 3 Let $n' = \frac{n}{10}$ for the number of simulations in the training step;
 - 4 **for** $i := 1$ **to** n' **do**
 - 5 Compute a simulation run π starting from s_0 until reaching a goal state or a BSCC B ;
 - 6 **if** *the last state of π is a goal state* **then**
 - 7 Add all states in π to $S_{sam}^?$;
 - 8 **else**
 - 9 Add the states that are in B to S_{sam}^0 ;
 - 10 Use $S_{sam}^?$ and S_{sam}^0 to construct a decision tree DT ;
 - 11 **return** DT ;
-

The algorithm runs n' simulations following the standard SMC method, e.g. in Legay and Viswanathan (2015), Legay *et al.* (2010). These simulations start from s_0 and stop at goal states or detected BSCCs with maximum uncertainty ϵ (Daca *et al.*, 2017). Depending on each run π , if π reaches a goal state, its states are added to $S_{sam}^?$ as they cannot be in S^0 . If π reaches a BSCC without any goal states, the whole detected BSCC states are added to S_{sam}^0 , as they cannot reach a goal.

After n' runs, the algorithm has two training sets S_{sam}^0 and $S_{sam}^?$. It then uses machine learning to construct a decision tree classifier DT that can classify any state as belonging to S^0 or not, based on state's variable valuations as data features. The SMC method then

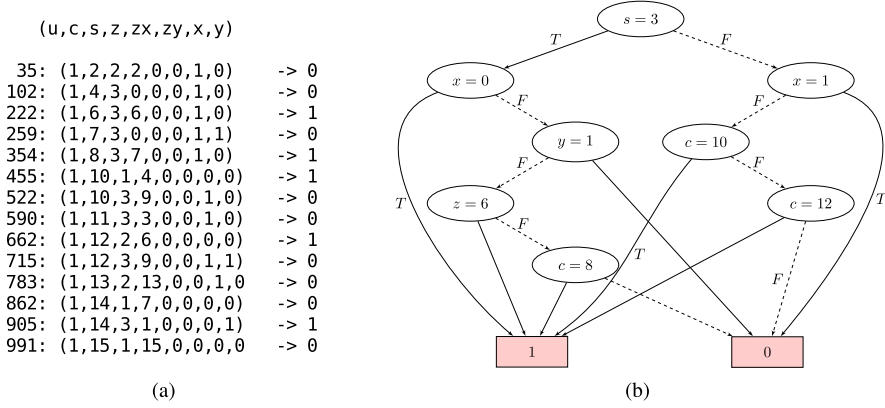


Fig. 4. a) Variable valuations for some states and their labels for the training step, b) The corresponding BDD.

applies DT to classify states during the remaining 90% of simulation runs. Starting from s_0 , DT can be applied on a state $s \in S$ considered as a successor state during a run. If it classifies s as in S^0 , the run is terminated early, reporting a 0 outcome. The training set size n' can be reduced if sufficient to build an accurate classifier.

EXAMPLE 5. In Fig. 4a, we propose some results of running 10000 simulations on the constructed DTMC of the compiled PRISM code of *NAND* multiplexing (Norman *et al.*, 2005), where parameters N and K of *NAND* multiplexing are set to 20 and 3, respectively. The following figure illustrates the state numbers (e.g. 35), their variable valuations, and their corresponding labels (including 8 features used in the training process), 0 if the state is in S_{sam}^0 and 1 if it is in $S_{sam}^?$.

The key factors affecting the performance of our method are memory consumption, runtime, and the correctness of SMC using decision trees in detecting states of S^0 . We aim to achieve a suitable trade-off among these factors in our approaches.

Memory Consumption. The SMC method stores all states from each simulation run, leading to a worst-case space complexity linear in the model size. In practice, only a small portion of states are typically encountered. The size of the training sets S_{sam}^0 and $S_{sam}^?$ also affects memory consumption. To address potential state explosion, we limit the size of these sets and terminate the training if the number of states exceeds the limit. Notably in machine learning, appropriate classifiers work on a limited number of training samples and generalize the results to predict the label of unseen data (Pasupa and Sunhem, 2016; Tadjudin and Landgrebe, 1996).

Using BDDs to precisely maintain S_{sam}^0 states is another solution. BDDs mark states as S^0 members only when confirmed as BSCC states, while providing efficient and compact data storage for S_{sam}^0 and $S_{sam}^?$ in large DTMC models with multiple BSCCs.

EXAMPLE 6. Consider the DTMC model in Fig. 2c with k reachable BSCCs, where memory constraints limit S_{sam}^0 to store only 10 BSCCs. For $k \gg 10$, Algorithm 1 can train

on only a subset of BSCCs, potentially failing to correctly classify states in uncovered BSCCs depending on model structure and training set properties.

Instead of using a single DT for all BSCCs, we can incrementally construct a BDD D to represent BSCC members. Initially, D stores states of the first detected BSCC. For subsequent runs, D checks for BSCC entrance while parallel on-the-fly detection continues. When new BSCCs are found, their states are added to D , enabling immediate detection in future runs.

EXAMPLE 7. The corresponding BDD of Fig. 4a is demonstrated in Fig. 4b. Variables u , zx , and zy remain constant across states, thus ignoring in the BDD. The root of the BDD is conditioned on $s = 3$, if the condition is true, the left branch with solid line (labelled by T) is traversed; otherwise, the right branch with dashed line is followed. By evaluating the values of various parameters, the corresponding labels are derived at the terminal nodes labelled 1 and 0.

Running Time. The main objectives are to reduce SMC runtime with low memory overhead, while maintaining computation correctness. The decision trees act as black-boxes to avoid redundant work. The runtime of on-the-fly BSCC detection depends on the DTMC topology and minimum transition probabilities (Brázdil *et al.*, 2014; Daca *et al.*, 2017). For models without cycles or with small BSCCs, early BSCC detection is possible. But for models with large BSCCs, long runs are needed to detect them confidently.

EXAMPLE 8. Consider Fig. 2a with BSCC B of 1000 states, where the minimum non-zero transition probability $p_{min} = 0.1$ and the average run to reach B is 300 steps. To detect B with 0.99 confidence at error bound $\delta = 0.01$, the standard approach in Daca *et al.* (2017) requires a path π long enough to meet each state of B at least n times, where $1 - (1 - p_{min})^n < 0.01$. We should have $n \geq 43$, i.e. meeting each B state 43 times, resulting in a path length of at least 43 300 steps. In contrast, our approach can detect the BSCC after only about 300 steps using a decision tree classifier, reducing the run length by 144 times.

Applying a decision tree to simulation states incurs some overhead, potentially increasing computation time. To mitigate this, our approach can call the decision tree only every 10 or 20 times over states along a path. Using BDDs to store BSCCs accelerates SMC through early BSCC detection. With efficient set union operations, Algorithm 1 requires only one long run per BSCC, potentially reducing the required runs below n' .

Correctness of Computation. Correctness is measured by the error rate in reachability probability computations. Using classifiers like decision trees for determining S^0 membership introduces two types of errors: false positives (label an $S \setminus S^0$ state as in S^0) and false negatives (label an S^0 state as not in S^0). While false negatives merely cause redundant SMC computations and can be mitigated using parallel conservative methods (e.g. Daca

et al., 2017), false positives critically affect correctness by causing incorrect simulation termination, thus impacting reachability probability calculations.

Although decision trees can achieve perfect classification on S_{sam}^0 and $S_{sam}^?$, they may misclassify new states. To reduce false positives, we can expand $S_{sam}^?$ by randomly adding states not in S_{sam}^0 , leading to diversify $S_{sam}^?$. While this may increase false negatives by biasing the tree towards $S^?$, it reduces false positives.

4.2. Enhancing the Decision Tree with More Sample States

Decision trees can generalize from a small training set, which is beneficial when only a portion of S^0 is in BSCCs (as in Fig. 2b). To terminate runs reaching S^0 as soon as possible and reduce overall running time, we should consider more S^0 states than just those in BSCCs for the training set. To this end, we propose two techniques that enable the decision tree to detect a larger portion of S^0 . In cases where explicitly representing the S_{sam}^0 set is infeasible, BDDs can be used to store it.

Adding States that Almost Surely Lead to S_{sam}^0 . We can add predecessor states to S_{sam}^0 if their successors are all in S_{sam}^0 , as these states almost surely lead to S^0 . Formally for $s \in S_{sam}^0$, we add $s' \in pre(s)$ to S_{sam}^0 if every $s'' \in post(s')$ satisfies either $s'' = s'$ or $s'' \in S_{sam}^0$. Algorithm 2 uses this approach. Starting from the path's final state in S_{sam}^0 , the algorithm processes states backwards. For each state, if all its successors belong to S_{sam}^0 , the algorithm adds it to S_{sam}^0 . Otherwise, it terminates, as the state's membership in S_{sam}^0 cannot be determined. This algorithm can be called by Algorithm 1 after detecting a new BSCC, to expand S_{sam}^0 .

Using BDDs, hash tables, heap, or similar data structures may introduce overhead for this operation. An alternative is to consider only Dirac transitions (probability is 1). For state $s = \pi[i - 1]$, the algorithm adds s to S_{sam}^0 if it has a Dirac transition; otherwise, it terminates. This simplified check reduces overhead but may identify fewer states for S_{sam}^0 .

Algorithm 2: Adding states to S_{sam}^0 based on Dirac transitions

Input: A DTMC D , a path π , and S_{sam}^0 .

Output: S_{sam}^0 with more states added.

```

1 for  $i := \text{length}(\pi)$  downto 1 do
2    $s := \pi[i - 1]$ ;
3   if  $post(s) \in S_{sam}^0$  and  $s \notin S_{sam}^0$  then
4     Add  $s$  to  $S_{sam}^0$ ;
5   else
6     break;
7 return  $S_{sam}^0$ ;

```

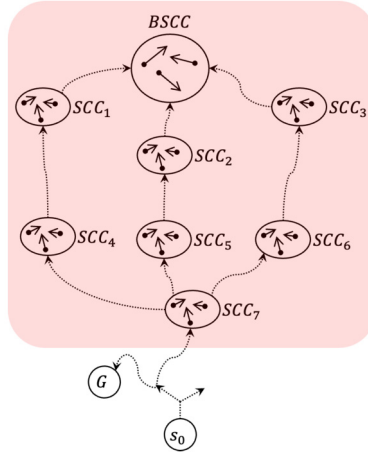


Fig. 5. A model with a large S^0 containing small BSCCs and several SCCs.

Statistical Analysis to Detect S^0 states. A large S^0 with small BSCCs may contain multiple SCCs that eventually reach a BSCC. Figure 5 illustrates this structure, where S^0 (shaded area) contains several nontrivial SCCs with paths to BSCCs. These nontrivial SCCs within S^0 can make simulation runs longer.

While simulation-based approaches exist for approximating SCCs (Brázdil *et al.*, 2014), they cannot reliably identify S^0 due to potential goal state reachability. Our solution enhances S^0 detection beyond BSCC states through statistical sampling. For each randomly selected state, we run multiple simulations. If any simulation reaches a goal state, the state is excluded from S^0 . Otherwise, we can statistically infer its membership in S^0 with high confidence.

Algorithm 3 implements this approach. It first determines the required simulation count l based on confidence parameters δ and ϵ . Through k iterations, it selects states not in S_{sam}^0 and runs simulations until either reaching G or confirming BSCC containment without goal states. States reaching G are added to $S_{sam}^?$, while those confirmed unreachable are added to S_{sam}^0 along with their successor states.

Algorithm 3 determines the number of simulation runs l needed to ensure, with high confidence, that the probability of reaching G from s is below threshold ϵ . While this condition likely holds for many states reachable from s that are added to S_{sam}^0 , some states with non-zero probability of reaching G might be incorrectly included, affecting correctness. To mitigate this, we can modify Algorithm 3 by limiting either the number of successor states or the search depth (e.g. to 4 steps) when adding states to S_{sam}^0 .

A limitation of Algorithm 3 is its tendency to select many states for S_{sam}^0 but few for $S_{sam}^?$, potentially increasing false positives. We propose two extensions: 1) Algorithm 3 (V2) randomly adds states to $S_{sam}^?$ (between the sizes of current $S_{sam}^?$ and S_{sam}^0), while 2) Algorithm 3 (V3) runs single simulation from random states, adding successful goal-reaching paths to $S_{sam}^?$. Unlike the original algorithm, V3 discards states after one failed simulation run.

Algorithm 3: Adding states to S_{sam}^0 and $S_{sam}^?$ by simulation

Input: A DTMC D , k the number of states to run experiments on, δ and ϵ as the parameters for confidence interval.

Output: S_{sam}^0 , $S_{sam}^?$.

```

1 Let  $l$  as the number of simulation runs to satisfy the conditions of confidence
  interval;
2 for  $i := 1$  to  $k$  do
3   Select a state  $s \in S \setminus S_{sam}^0$  randomly;
4    $flag := false$ ;
5   for  $j := 1$  to  $l$  do
6     Run a path  $\pi$  starting from  $s$  and terminating in a goal state or in a BSCC
       without goal state;
7     if  $\pi$  terminates in a goal state then
8        $flag := true$ ;
9       break;
10    if  $flag := false$  then
11       $Q := \{s\}$ ;
12      while  $Q$  is not empty do
13        Remove a state  $t$  from  $Q$ ;
14        Add  $t$  to  $S_{sam}^0$ ;
15        forall  $s' \in post(t)$  do
16          if  $s' \notin S_{sam}^0$  then
17            Add  $s'$  to  $S_{sam}^0$ ;
18      else
19        Add all states of  $\pi$  to  $S_{sam}^?$ ;
20 return  $S_{sam}^0$ ,  $S_{sam}^?$ ;

```

EXAMPLE 9. Fig. 6 shows the decision tree from Algorithm 3 (V3) for *egl* (Norman and Shmatikov, 2006) with $N = 6$ and $L = 8$. The tree has 21 nodes. In each node, the decision condition, as well as the number of samples in each class (S^0 and $S^?$) is shown. Notably, for this case of *egl*, a total of 84 variables are in each state. Each variable serves as a feature during the model training process. For example, at the root of the tree (*node* #0), the decision condition is $x[44] \leq 7.5$, i.e. feature 44 is evaluated for states. In this node, $value = [151, 206]$ means that 206 $S^?$ and 151 S^0 states are used to classify. That is, if $x[44] \leq 7.5$ is met (true), a left branch to *node* #1 is considered. If the condition is not met (false), the right branch is *node* #2. Node colours indicate sample distribution: green for S^0 -majority, pastel orange for $S^?$ -majority. Tree traversal determines S^0 states at leaf nodes: *node* #10, *node* #11, *node* #14, *node* #19, and *node* #20 indicate S^0 , while *node* #1, *node* #4, *node* #6, *node* #9, *node* #16, and *node* #18 show $S^?$.

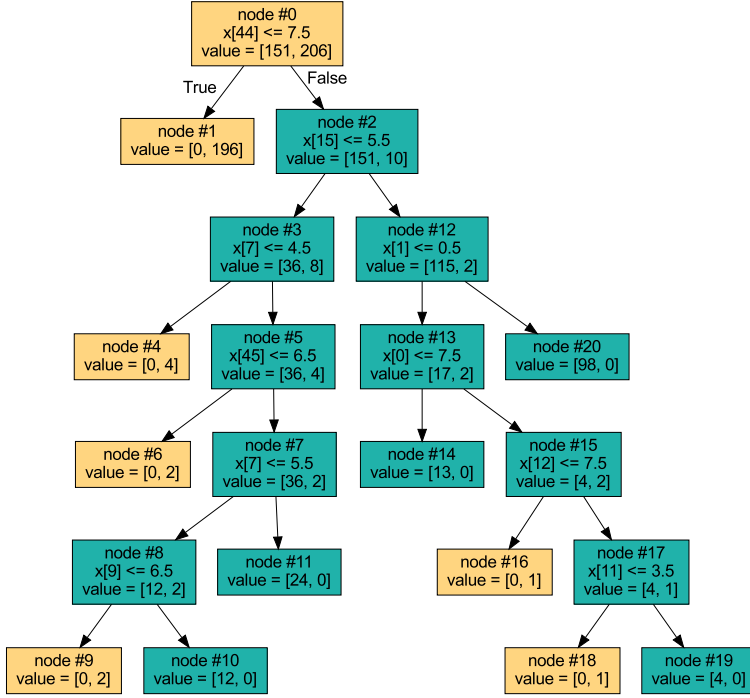


Fig. 6. The decision tree classifier obtained by Algorithm 3 (V3) for the case of *egl*.

To determine the minimum number of simulation runs to meet the conditions of the confidence interval for the membership of a state in S^0 , we recall X_i in Section 3 as the Bernoulli distribution with parameter p , that is, $Pr(X_i = 1) = p$, $0 \leq p \leq 1$. After running n simulations, we observe the outcome $\underline{x} = (x_1, x_2, \dots, x_n)$. The probability of p given the evidence \underline{x} is denoted by $Pr(p|\underline{x})$ (the posterior probability). We use the following lemma to obtain a bound for n .

Lemma 1. *Let ϵ and δ be real numbers, where $0 < \delta, \epsilon < 1$. To have $Pr(p \leq \delta|\underline{x}) > 1 - \epsilon$, we require at least n simulation runs that never reach G where $n \geq \frac{\log \epsilon}{\log(1-\delta)}$.*

Proof. Let X_s defined as $X_1 + X_2 + \dots + X_n$. Therefore, X_s has the Binomial distribution of parameter n and p . Due to the lack of any information of p , we consider a non-informative uniform distribution $p \sim Uniform(0, 1)$. Thus, the posterior distribution of p given \underline{x} has distribution $Beta(\alpha, \beta)$, where $\alpha = \sum_{i=1}^n x_i + 1$ and $\beta = n - \sum_{i=1}^n x_i + 1$. Consider the case where all of n simulation runs reach a BSCC without any goal states. In this case, $\sum_{i=1}^n x_i = 0$. Therefore, the posterior has distribution $Beta(1, n + 1)$. Recall that $Beta(\alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1} (1-p)^{\beta-1}$. The posterior density function of $p|\underline{x}$ equals to $(n+1)(1-p)^n$. Thus, we have

$$Pr(p \leq \delta|\underline{x}) = \int_0^{\delta} (n+1)(1-p)^n dp = 1 - (1-\delta)^{n+1}.$$

Considering $Pr(p \leq \delta | \underline{x}) > 1 - \epsilon$, we have $1 - (1 - \delta)^{n+1} > 1 - \epsilon$. To do so, we should have $(1 - \delta)^{n+1} < \epsilon$ and hence $(n + 1) \cdot \log(1 - \delta) < \log \epsilon$. Because $\log(1 - \delta) < 0$, it follows that $n + 1 > \frac{\log \epsilon}{\log(1 - \delta)}$ or $n \geq \frac{\log \epsilon}{\log(1 - \delta)}$. \square

If any simulation run reaches a goal state, s cannot be in S^0 . Lemma 1 determines the required number of simulation runs to establish an upper-bound for reachability probability with a specified confidence level. When no simulation reaches a goal state, even if s is incorrectly added to S_{sam}^0 , we can be confident that the probability of reaching G is below threshold δ . This ensures that such misclassification maintains the overall correctness of the SMC approach.

4.3. Statistical Approximation of Simulation Run Length

Our approach aims to reduce simulation run lengths by efficiently identifying paths that cannot reach goal states. We propose a statistical method that computes the mean and standard deviation of successful goal-reaching paths to establish an upper bound (ub) for path length. The SMC method can then terminate runs exceeding ub , marking them as non-reaching outcomes. Using parameters ϵ and δ for confidence intervals, we calculate the required number of simulation runs n to satisfy the Chernoff-Hoffding bound. This enhancement consists of two steps:

Training Step. Using $n' = \frac{n}{10}$ simulations as a heuristic, we compute mean μ and standard deviation σ of goal-reaching path lengths. Chebyshev's inequality helps establish an upper bound that ensures the probability of reaching a goal state beyond this bound is below threshold ϵ ($Pr(|X - \mu| \geq k \cdot \sigma) \leq \frac{1}{k^2}$), where X represents path length to G . Having ϵ as an upper-bound for errors in detecting a state in S^0 , we can set $k = \sqrt{\frac{1}{\epsilon}}$. Assuming $X > \mu$, we define $ub = \mu + \sqrt{\frac{1}{\epsilon}} \cdot \sigma$ as the path length upper bound.

Remaining SMC Runs. Once ub is established, SMC terminates any simulation exceeding this bound, as these paths have probability of at least $1 - \epsilon$ of never reaching a goal state. This eliminates the need for BSCC detection. This statistical approach differs from our other approaches by reducing computational overhead through direct path length analysis rather than invoking constructed decision trees or BDDs. Furthermore, it provides mathematical control over false negative rates, which is not possible with decision trees.

5. Evaluation

In this section, we present experimental results to validate our proposed approaches. We evaluate the ML metrics (including precision, recall, specificity, negative predictive value, as well as precision-recall and receiver operating characteristic curves), the correctness of state classification and computational efficiency across diverse DTMC models. Furthermore, we compare the proposed approaches with the state-of-the-art.

5.1. Experimental Setup

We evaluate our approaches using four standard DTMC model classes: *Nand*, *brp*, *egl*, and *crowds*, as documented in Kwiatkowska *et al.* (2012), Hartmanns *et al.* (2019), Budde *et al.* (2020b). The reachability properties under investigation are:

- $P_{=?} [F \text{ ! knowA } \& \text{ knowB}]$ for *egl* cases,
- $P_{=?} [F \text{ observe0 } > 1]$ for *crowds* cases,
- $P_{=?} [F \text{ s } = 4 \ \& \ \text{z/N} < 0.1]$ for *Nand* cases,
- $P_{=?} [F \text{ s } = 5]$ for *brp* cases.

We implemented our algorithms in PRISM model checker, constructing models with varying parameter values. For fair comparison, we disabled pre-computation of S^0 and S^1 sets. To provide the reproducibility of our results and to maintain a balance between detection performance and computational overhead, we configured the DT classifiers with the following hyperparameters: Max Depth: limited to values between 4 and 10 (depending on model complexity) to avoid overfitting to specific simulation paths; Criterion: we used Gini impurity to measure the quality of splits, as it is computationally efficient for large state spaces; Random State: a fixed seed 0 was utilized for the training process to provide the reproducibility of the results across multiple executions. The other hyperparameters are set to their default values.

To evaluate the performance of the DT classifier, we employed a series of standard machine learning metrics. For this analysis, 10% of the whole set of states was allocated for training the model, while the remaining 90% was reserved for testing its predictive accuracy. Precision measures the reliability of a positive prediction of S^0 . It answers: of all states of the model labelled as S^0 , how many were actually S^0 ? Recall (true positive rate) measures the model’s ability to find all positive instances, i.e. of all S^0 states that exist, how many did the model successfully find? Specificity measures the model’s ability to correctly identify negative instances, of all the non- S^0 ($S^?$) states, how many were correctly labelled as such? Negative Predictive Value (NPV) measures the reliability of a negative prediction. It answers: of all states the model labelled as $S^?$, how many were truly $S^?$? False Positive Rate (FPR) answers a different question, i.e. of all the cases that were actually $S^?$, how many did the model incorrectly flag as S^0 ?. Finally, we used the Receiver Operating Characteristic (ROC) curve to visualize the classifier’s performance across various threshold settings. By plotting recall against FPR, the ROC curve illustrates the model’s fundamental ability to distinguish between classes.

5.2. Experimental Results

To evaluate the S^0 detection, the *egl* model with parameters $N = 5$ and $L = 6$, as well as Algorithm 3 (V3) is considered. In the *egl* model with the predefined parameters, each state includes 84 feature values, used for training the DT. With a Precision of 97.5%, we can deduce that when the model identifies something as S^0 , it is almost certainly correct. This high level of certainty is backed by a Specificity of 95.1%, meaning it rarely confuses

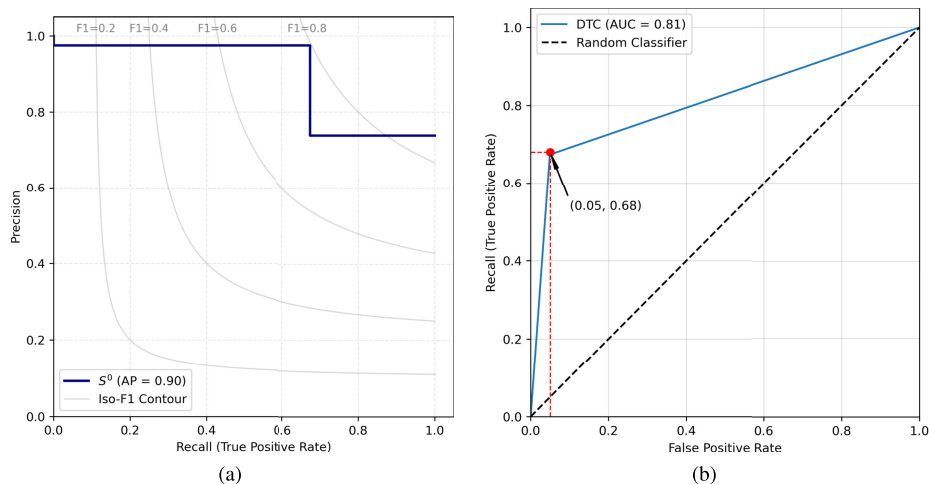


Fig. 7. Quantitative performance evaluation of the decision tree classifier for detecting S^0 in the case of model *egl* with $N = 5$ and $L = 6$. (a) Precision-recall curve showing the trade-off between precision and recall, overlaid with Iso-F1 contours. (b) ROC curve plotting the recall against the FPR. The model attains an AUC of 0.81, demonstrating discriminatory power significantly superior to the random classifier baseline.

the background with our target. The trade-off for this high accuracy is that the model is somewhat conservative; it only catches about 67.4% of all actual S^0 instances. In contrast, the model takes a wide-net approach for S^2 . It achieves a high recall of 95%, ensuring that very few actual S^2 instances are missed. However, this inclusiveness comes at a cost to precision, a fact further reflected by the 50.9% NPV. This tells us that while the model finds almost every S^2 case, about half of those predictions are actually false alarms. Ultimately, with F1-scores of 80% and 66%, the data shows a classifier that is fine-tuned to be an expert at identifying the target class while serving as a sensitive, albeit less precise, detector for the rest. These findings are consistent with the discrete operating points observed in the precision-recall and ROC curves illustrated in Fig. 7.

Figure 7 illustrates a quantitative evaluation of the DT for identifying S^0 states by demonstrating precision-recall and ROC curves. Our evaluation reveals that the classifier is both reliable and highly discerning, achieving an average precision (AP) of 0.90 and an area under the curve (AUC) of 0.81. What stands out most is the model's sweet spot at approximately 68% recall. At this point, the classifier maintains nearly perfect precision while keeping false alarms to a remarkably low 5%. While we can push the model to catch every single positive instance (100% recall), it does so at a reasonable cost, with precision dipping to 0.74. Throughout this range, the model remains remarkably stable, consistently holding an F1-score of about 80%. Essentially, these curves describe a classifier that is good at making high-confidence predictions. These results demonstrate that the constructed DT provides good separation, suggesting that our proposed DT-based methods are promising.

Table 1 presents our experimental results showing for each case study: model name and parameter values, total number of states ($|S|$), non-goal-reaching states ($|S^0|$) as re-

Table 1
Correctness of the proposed approach for the selected DTMC models.

Model name	Parameter values	$ S $	$ S^0 $	Algorithm 1		Algorithm 2		Algorithm 3 (V1)		Algorithm 3 (V2)		Algorithm 3 (V3)	
				Correctness	$ \approx S^0 $	Correctness	$ \approx S^0 $	Correctness	$ \approx S^0 $	Correctness	$ \approx S^0 $	Correctness	$ \approx S^0 $
<i>egl</i>	$N = 5, L = 6$	115 710	85 376	> 99.9%	17	> 99.9%	85 605	98.4%	87 209	> 99.9%	86 919	98.4%	97 457
	$N = 5, L = 8$	156 670	115 136	> 99.9%	18	> 99.9%	114 869	98.4%	116 039	> 99.9%	114 570	> 99.9%	112 263
	$N = 6, L = 6$	552 958	415 456	> 99.9%	20	> 99.9%	414 580	74.6%	413 950	99.7%	423 114	> 99.9%	457 582
	$N = 6, L = 8$	749 566	560 608	> 99.9%	22	> 99.9%	559 792	48.3%	671 104	> 99.9%	560 744	99.6%	566 450
<i>crowds</i>	$TR = 5, CS = 15$	592 060	472 368	> 99.9%	15 504	> 99.9%	385 988	91.9%	592 060	> 99.9%	9 442	99.8%	384 238
	$TR = 5, CS = 20$	2 061 951	1 745 849	> 99.9%	53 130	> 99.9%	1 446 459	92.4%	2 007 425	99.8%	24 539	> 99.9%	220 151
	$TR = 6, CS = 10$	253 525	227 271	> 99.9%	8 008	> 99.9%	180 488	85.5%	320 750	> 99.9%	4 433	> 99.9%	72 889
	$TR = 6, CS = 15$	2 464 168	1 844 704	> 99.9%	54 264	> 99.9%	1 498 636	87.2%	2 365 053	> 99.9%	21 204	> 99.9%	364 698
<i>Nand</i>	$N = 20, K = 4$	171 542	29 276	> 99.9%	21	> 99.9%	40	94.9%	46 098	99.9%	7 164	99.8%	41 718
	$N = 30, K = 2$	681 362	128 079	> 99.9%	31	> 99.9%	56	51.8%	490 409	> 99.9%	14 802	85.7%	158 627
	$N = 30, K = 3$	1 020 152	128 079	> 99.9%	31	> 99.9%	56	43.9%	588 956	93.9%	20 628	99.8%	109 731
<i>brp</i>	$N = 1024, Max = 2$	43 013	4 105	97.4%	2 052	97.4%	4 120	99.1%	7 560	99.4%	126	99.4%	352
	$N = 2048, Max = 2$	89 021	8 201	> 99.9%	12 291	94.8%	8 216	> 99.9%	6 144	> 99.9%	270	94.8%	8 205

ported by PRISM. Moreover, the number of approximated S^0 states ($|\approx S^0|$) by our learning methods, as well as the correctness in computing reachability probability for the initial state are reported. The correctness metric for reachability probabilities is defined as $\frac{\tilde{x}_{s_0}}{x_{s_0}}$, where $x_{s_0} = P_{\Rightarrow}(\Psi)$ represents the actual probability and \tilde{x}_{s_0} denotes the computed value using decision trees for S^0 and $S^?$ labelling. We evaluate three versions of Algorithm 3: V1: original algorithm without S_{sam}^0 or $S_{sam}^?$ additions, V2: adds randomly selected states to $S_{sam}^?$, and V3: adds goal-reaching path states.

Algorithms 1 and 2 yield more conservative results for the first three model classes. For these cases and PCTL properties, the models contain several small BSCCs where the constructed decision trees properly detect the states within them. The main limitation of these approaches (Algorithms 1 and 2) is its tendency to overfit to the states in BSCCs, resulting in poor generalization to other parts of S^0 . While this limitation affects the simulation run length, it maintains high precision in BSCC detection and run termination. For the last class of case studies (*brp*), the computational correctness varies across models depending on the parameter values. The first two algorithms suffer from overfitting, incorrectly labelling some unseen states in $S \setminus S^0$ as belonging to S^0 .

The experimental results demonstrate that in most cases, balancing the sizes of S_{sam}^0 and $S_{sam}^?$ generally improves the computational correctness of Algorithm 3. While the first version of this algorithm (denoted as (V1) in Table 1) performs poorly and achieves correctness less than 95% in nine cases, there is only one case for version 2 and two cases for version 3 that the correctness is less than 95%. Notably, the main purpose of presenting the second and the third versions of Algorithm 3 is to cover its weaknesses in providing some results that are far from the exact value.

In addition to computational correctness, we report the number of detected states for S^0 (denoted by $|\approx S^0|$) for each approach. For *egl* and *crowds* models, the values of $|\approx S^0|$ are not far away from $|S^0|$. For the *crowds* models, Algorithm 2 demonstrates the best performance in detecting a substantial portion of $|S^0|$ while maintaining computational correctness. For these models and other model classes, Algorithm 3 (V1) labels the widest range of states as $|S^0|$ and achieves the highest $|\approx S^0|$ among all proposed approaches.

In the case of *Nand* models, Algorithms 1 and 2 overfit to the training set and detect only a small portion of states in S^0 . For these models, the first and second versions of Algorithm 3 label an excessive number of states as S^0 , which degrades computational correctness. Similarly, in *egl* models, Algorithm 1 exhibits overfitting.

To evaluate the impact of the statistical approach for terminating simulation runs, we apply the method proposed in Section 4.3 to the selected case study models. The results are presented in Table 2. For each model, we execute 10% of the required simulations for the training step. The average and standard deviation of the lengths for each model are reported in the table. Based on the relation proposed in Section 4.3 and to bound the errors to less than 0.01, we determine the computed upper bounds for the length of remaining simulation runs. In three classes of models (*egl*, *crowds*, and *brp*), the computed upper bounds for terminating simulation runs are completely perfect and correctly terminate runs that have reached S^0 . For the *Nand* models, the correctness exceeds 99.6%, indicating near-perfect run termination.

Table 2
Correctness of the proposed statistical approach for bounding simulation runs with $\epsilon = 0.01$.

Model name	Parameter values	μ	σ	ub	Correctness
<i>egl</i>	$N = 5, L = 6$	94.5	34.5	440	> 99.9%
	$N = 5, L = 8$	123.5	44.5	568.5	> 99.9%
	$N = 6, L = 6$	117.9	41.3	531.5	> 99.9%
	$N = 6, L = 8$	152	53.5	687	> 99.9%
<i>crowds</i>	$TR = 5, CS = 15$	74.4	22.3	297	> 99.9%
	$TR = 5, CS = 20$	78.2	27.1	349	> 99.9%
	$TR = 6, CS = 10$	86.5	22.6	313	> 99.9%
	$TR = 6, CS = 15$	87.3	26.4	351	> 99.9%
<i>Nand</i>	$N = 20, K = 4$	721	0.05	721.5	99.6%
	$N = 30, K = 2$	601	0.04	601.4	99.6%
	$N = 30, K = 3$	841	0.08	841.8	99.7%
<i>brp</i>	$N = 1024, Max = 2$	6219	155	7769	> 99.9%
	$N = 2048, Max = 2$	12429	454	16968	> 99.9%

Table 3
The stability analysis of the experimental results for the significance level of 0.01.

		Levene's test for equality of variances	T-test for equality of means		
		Sig.	Sig. (2-tailed)	99% Confidence interval of the difference	
				Lower	Upper
Correctness	Equal variances assumed	0.598	0.520	-0.004095	0.002533
	Equal variances not assumed	-	0.520	-0.004096	0.002534

5.3. Stability Analysis

Statistical model checking reliability requires multiple runs to evaluate result stability, defined as the consistency of results across multiple experimental runs. This evaluation employs t -test and Levene's test (Milton and Arnold, 2002), where Levene's test examines variance equality, and t -test verifies mean equality between independent result groups.

In the evaluation process, the proposed algorithms were independently executed 30 times and the probability of reachability properties was computed for a single instance, i.e. in the case of *egl*, $N = 6$, and $L = 8$. Algorithm 3 (V3) and the instance were selected as a representative example of all possible parameters and instances. The experimental runs were subsequently partitioned into two distinct groups, each comprising 15 independent runs. Table 3 represents the stability analysis of the experimental results.

The Levene's test significance value (Sig. = 0.598) exceeds 0.01, confirming equal variance assumption. The t -test significance (0.520) and the difference interval containing 0 indicate no statistically significant differences between group means at 99% confidence, confirming result stability in the sense that the number of runs is sufficient.

Table 4

Runtime, memory consumption, and the number of detected S^0 states for the selected DTMC models of our approaches in comparison with state-of-the-art.

Model name	Parameter values	Algorithm 2		Algorithm 3 (V3)		SimAdaptive (Daca <i>et al.</i> , 2017)		
		Time	Memory	Time	Memory	$ \approx S^0 $	Time	Memory
<i>egl</i>	$N = 5, L = 6$	7.6	8.4 MB	8.3	539 KB	523	27.6	32KB
	$N = 5, L = 8$	13	9.7 MB	28.9	752 KB	522	38.4	43 KB
	$N = 6, L = 6$	9.5	12 MB	19.6	728 KB	1428	36.8	39.6 KB
	$N = 6, L = 8$	15.5	17 MB	29.4	921 KB	1391	49	50.7 KB
<i>crowds</i>	$TR = 5, CS = 15$	1.5	653 KB	1.6	67 KB	603	2	9.3 KB
	$TR = 5, CS = 20$	1.5	643 KB	1.7	57 KB	835	2	9.4 KB
	$TR = 6, CS = 10$	2.5	955 KB	2.9	114 KB	528	3.3	11 KB
	$TR = 6, CS = 15$	2.6	6.5 MB	3	788 KB	845	3.3	11.2 KB
<i>Nand</i>	$N = 20, K = 4$	10.9	489 KB	10.5	293 KB	18	11.1	23 KB
	$N = 30, K = 2$	10	135 KB	6.5	157 KB	18	10.3	19 KB
	$N = 30, K = 3$	13.7	335 KB	3.5	244 KB	19	14	26 KB
<i>brp</i>	$N = 1024, Max = 2$	42.8	472 KB	32	424 KB	136	43.4	446 KB
	$N = 2048, Max = 2$	184	867 KB	37	23 KB	257	188	905 KB

5.4. Comparison with the State-of-the-Art Methods

Based on their superior performance and robustness within the DTMC models for the SMC task, we selected Algorithm 2 and Algorithm 3 (V3) as the basis for our comparison. In the literature, SimTermination and SimAnalysis developed by Younes *et al.* (2010), SimAdaptive by Daca *et al.* (2017), and the method developed by Brázdil *et al.* (2015) represent the state-of-the-art. Since the approach by Daca *et al.* has been shown to dominate these existing methods (Daca *et al.*, 2017), we evaluate it against our proposed algorithms. The results, obtained using a machine equipped with a 1.8 GHz Intel(R) Core(TM) i7-10510U processor and 12 GB of RAM running Ubuntu 24.04, are presented in Table 4. In the table, the running time in seconds, memory consumption for both the training step and the active simulation phase of SMC process, and the number of detected S^0 states are reported.

As demonstrated in Table 4, in most cases, our proposed approaches run significantly faster than the state-of-the-art SimAdaptive method while maintaining the computational correctness of the results. For example, in the case of *egl* for $N = 5$ and $L = 6$, Algorithm 2 and Algorithm 3(V3) consume 7.6 and 8.3 seconds, respectively, while SimAdaptive takes three to four times longer. For the *Nand* and *brp* cases, Algorithm 3(V3) works faster than Algorithm 2. Compared to SimAdaptive, our proposed framework labels a significantly larger portion of states as belonging to S^0 without dramatically degrading the precision of reachability probabilities. This ability to identify a wider range of S^0 states leads to the early termination of simulation runs that would otherwise enter BSCCs. In contrast, SimAdaptive detects only a small part of S^0 and continues simulations on many states that actually have zero probability of reaching a goal state. For example, in the case of *egl* for $N = 5$ and $L = 8$, Algorithm 2 and Algorithm 3(V3) detect 114 869 and 112 263 states of 115 136 states in S^0 , while SimAdaptive detects 522 states. Regarding

resource efficiency, the memory consumption of our compared approaches remains highly manageable, with usage in most cases staying below 1MB.

6. Related Work

Statistical model checking for stochastic systems with bounded properties was pioneered by several researchers: Younes and Simmons (2002), Sen *et al.* (2004), Younes *et al.* (2006), David *et al.* (2011), Henriques *et al.* (2012). Sen *et al.* (2005) extended this to unbounded formulas using Monte Carlo simulation and hypothesis testing, implementing their approach in VESTA. Daca *et al.* (2017) later developed on-the-fly algorithms for SMC to detect BSCCs for unbounded properties. Our work enhances this approach by incorporating decision trees to eliminate redundant computations, as demonstrated in Example 8.

The intersection of machine learning and formal methods has gained significant attention (Larsen *et al.*, 2022). In the verification of stochastic systems, several approaches have emerged in Brázdil *et al.* (2014), Bortolussi *et al.* (2015), Mohagheghi and Salehi (2024). Brázdil *et al.* (2014) introduced machine learning for probabilistic reachability in MDPs, though their method differs from our statistical S^0 detection algorithm (Algorithm 3). Bortolussi *et al.* (2015) focused on bounded properties (bounded until) for stochastic models, unlike our unbounded approach. For MDP policy optimization, Brázdil *et al.* (2015) used decision trees for near-optimal policies, while Gros *et al.* (2020), Gros *et al.* (2022) employed deep learning. Wang *et al.* (2018) proposed an alternative approach using evolutionary algorithms for learning probabilistic models, comparing their efficiency with traditional SMC methods.

7. Threats to Validity

In examining the limitations of our approaches, we analyse various elements that may influence the validity of the results. Research validity encompasses two main categories, external and internal validity. The former refers to the capability of research results to be applied across different contexts and scenarios.

Regarding the external validity, while we selected four widely-used and standard DTMC models (*Nand*, *brp*, *egl*, and *crowds*), our results might not generalize to all types of DTMCs. The performance of existing approaches, including our proposed ones, degrades when processing dense models, particularly those containing a very large BSCC. Traditional SMC methods often treat the system as a black-box, requiring only the ability to observe the satisfaction of properties. Our approach, however, falls into the category of white-box or grey-box SMC. In models described via high-level languages like PRISM or STORM, the state variables (counters, flags, etc.) automatically serve as features. No manual feature engineering is required beyond using the existing state valuation. Consequently, this framework is specifically designed for white-box or grey-box verification where the model structure or state variables are accessible during simulation. In some models, the available features may be insufficient for reliable classification of S^0 states.

This can be detected by evaluating classifier performance on smaller instances of the same DTMC class. When classification is ineffective, we rely on a BDD-based approach (e.g. Algorithm 1) that focuses on accurately labelling confirmed S^0 states for reuse in SMC, rather than generalizing to unseen states.

Internal validity addresses the methodological robustness of experimental procedures. In the context of our investigation, several potential challenges to internal validity have been identified, including measurement correctness limitations, algorithmic stability concerns, and the appropriateness of statistical analytical methods. The specific factors that may compromise the internal validity of this study encompass:

- The randomness in state selection for $S_{sam}^?$ (Algorithm 3 (V2)) might affect the reproducibility of results. To mitigate this, we conducted multiple runs with different random seeds. A T-test is conducted to investigate the stability of the results.
- The decision tree's classification performance might vary depending on training data quality and feature selection. To address this threat, a uniformly random selection has been utilized to train decision trees, avoiding potential bias in the training data selection.
- The risk of overfitting is another internal validity in this research. To mitigate the risk of overfitting, which could lead to incorrect simulation termination, we not only limit the tree depth but also utilize a uniformly random selection for training data. Furthermore, we use Algorithm 3 to diversify the training set and reduce false positives by incorporating statistical sampling. Also, we mitigate overhead by calling the classifier only every 10–20 steps rather than at every state transition.

8. Conclusion and Future Work

We presented novel approaches for approximating qualitative reachability probabilities in DTMCs, particularly valuable for statistical model checking of quantitative properties. Our three-stage methodology involves: (1) using decision tree classification to identify states within bottom strongly connected components that cannot reach goal states, (2) employing BDDs to efficiently store unreachable states, and (3) conducting statistical analysis through model simulation. Experimental validation across four benchmark case studies demonstrated promising results.

Future research directions can include

- applying the proposed approaches to particular scenarios, e.g. security protocols in Salehi *et al.* (2019),
- extending these approaches to other Markovian models, such as MDPs, though this presents challenges in optimal policy approximation,
- investigating alternative ML models, such as light-weight ensemble methods (e.g. Random Forests with limited estimators) or pruning techniques to further enhance generalization without sacrificing the runtime efficiency required for large-scale DTMCs,
- developing hybrid approaches combining multiple ML techniques, e.g. with Markov clustering,

- using the structure of Markovian models to fine-tune hyperparameters utilized in the approaches,
- using knowledge across similar models to improve learning.

Conflict of Interest

None.

Acknowledgements

The authors appreciate Dr. Kambiz Ahmadi for his valuable comments.

References

- Agha, G., Palmiskog, K. (2018). A survey of statistical model checking. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 28(1), 1–39.
- Aichernig, B.K., Tappler, M. (2017). Probabilistic black-box reachability checking. In: *Runtime Verification: 17th International Conference, RV 2017, Seattle, WA, USA, September 13–16, 2017, Proceedings 17*. Springer, pp. 50–67.
- Arora, S., Rao, M.P. (2020). The Bouquet algorithm for model checking unbounded until properties. In: *2020 International Symposium on Theoretical Aspects of Software Engineering (TASE)*. IEEE, pp. 113–120.
- Ashok, P., Křetínský, J., Weininger, M. (2019). PAC statistical model checking for Markov decision processes and stochastic games. In: *Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15–18, 2019, Proceedings, Part I 31*. Springer, pp. 497–519.
- Ashok, P., Daca, P., Křetínský, J., Weininger, M. (2020). Statistical model checking: black or white? In: *Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles: 9th International Symposium on Leveraging Applications of Formal Methods, ISOFA 2020, Rhodes, Greece, October 20–30, 2020, Proceedings, Part I 9*. Springer, pp. 331–349.
- Baier, C., Katoen, J.-P. (2008). *Principles of Model Checking*. MIT Press.
- Baier, C., Hermanns, H., Katoen, J.-P. (2019). The 10,000 facets of MDP model checking. In: *Computing and Software Science: State of the Art and Perspectives*, pp. 420–451.
- Bao, Y., Chen, M., Zhu, Q., Wei, T., Mallet, F., Zhou, T. (2017). Quantitative performance evaluation of uncertainty-aware hybrid AADL designs using statistical model checking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(12), 1989–2002.
- Bertrand, N., Bordais, B., Hérouët, L., Mari, T., Parreaux, J., Sankur, O. (2019). Performance evaluation of metro regulations using probabilistic model-checking. In: *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification: Third International Conference, RSSRail 2019, Lille, France, June 4–6, 2019, Proceedings 3*. Springer, pp. 59–76.
- Bortolussi, L., Milios, D., Sanguinetti, G. (2015). Machine learning methods in statistical model checking and system design—tutorial. In: *Runtime Verification: 6th International Conference, RV 2015, Vienna, Austria, September 22–25, 2015, Proceedings*. Springer, pp. 323–341.
- Brázdil, T., Chatterjee, K., Chmelík, M., Forejt, V., Křetínský, J., Kwiatkowska, M., Parker, D., Ujma, M. (2014). Verification of Markov decision processes using learning algorithms. In: *Automated Technology for Verification and Analysis: 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3–7, 2014, Proceedings 12*. Springer, pp. 98–114.
- Brázdil, T., Chatterjee, K., Chmelík, M., Fellner, A., Křetínský, J. (2015). Counterexample explanation by learning small strategies in Markov decision processes. In: *Computer Aided Verification: 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18–24, 2015, Proceedings, Part I 27*. Springer, pp. 158–177.

- Bryant, R.E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 100(8), 677–691.
- Budde, C.E., D’Argenio, P.R., Hartmanns, A., Sedwards, S. (2020a). An efficient statistical model checker for nondeterminism and rare events. *International Journal on Software Tools for Technology Transfer*, 22(6), 759–780.
- Budde, C.E., Hartmanns, A., Klauck, M., Křetínský, J., Parker, D., Quatmann, T., Turrini, A., Zhang, Z. (2020b). On correctness, precision, and performance in quantitative verification: QComp 2020 competition report. In: *International Symposium on Leveraging Applications of Formal Methods*. Springer, pp. 216–241.
- Calinescu, R., Paterson, C., Johnson, K. (2019). Efficient parametric model checking using domain knowledge. *IEEE Transactions on Software Engineering*, 47(6), 1114–1133.
- Casaluce, R., Burattin, A., Chiaromonte, F., Vandin, A. (2022). Process mining meets statistical model checking: towards a novel approach to model validation and enhancement. In: *International Conference on Business Process Management*. Springer, pp. 243–256.
- Clarke, E.M., Zuliani, P. (2011). Statistical model checking for cyber-physical systems. In: *Automated Technology for Verification and Analysis: 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11–14, 2011. Proceedings 9*. Springer, pp. 1–12.
- Clarke, E.M., Henzinger, T.A., Veith, H. (2018). Introduction to model checking. In: *Handbook of Model Checking*, pp. 1–26.
- Cleaveland, M., Ruchkin, I., Sokolsky, O., Lee, I. (2022). Monotonic safety for scalable and data-efficient probabilistic safety analysis. In: *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (IC-CPS)*. IEEE, pp. 92–103.
- Daca, P., Henzinger, T.A., Křetínský, J., Petrov, T. (2017). Faster statistical model checking for unbounded temporal properties. *ACM Transactions on Computational Logic (TOCL)*, 18(2), 1–25.
- David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., Van Vliet, J., Wang, Z. (2011). Statistical model checking for networks of priced timed automata. In: *Formal Modeling and Analysis of Timed Systems: 9th International Conference, FORMATS 2011, Aalborg, Denmark, September 21–23, 2011. Proceedings 9*. Springer, pp. 80–96.
- David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., Sedwards, S. (2015). Statistical model checking for biological systems. *International Journal on Software Tools for Technology Transfer*, 17, 351–367.
- Dehnert, C., Junges, S., Katoen, J.-P., Volk, M. (2017). A storm is coming: a modern probabilistic model checker. In: *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24–28, 2017. Proceedings, Part II 30*. Springer, pp. 592–600.
- D’Argenio, P.R., Fraire, J.A., Hartmanns, A., Raverta, F. (2022). Comparing statistical and analytical routing approaches for delay-tolerant networks. In: *Quantitative Evaluation of Systems: 19th International Conference, QEST 2022, Warsaw, Poland, September 12–16, 2022. Proceedings*. Springer, pp. 337–355.
- Gros, T.P., Hermanns, H., Hoffmann, J., Klauck, M., Steinmetz, M. (2020). Deep statistical model checking. In: *Formal Techniques for Distributed Objects, Components, and Systems: 40th IFIP WG 6.1 International Conference, FORTE 2020, Held as Part of the 15th International Federated Conference on Distributed Computing Techniques, DisCoTec 2020, Valletta, Malta, June 15–19, 2020. Proceedings 40*. Springer, pp. 96–114.
- Gros, T.P., Hermanns, H., Hoffmann, J., Klauck, M., Steinmetz, M. (2022). Analyzing neural network behavior through deep statistical model checking. *International Journal on Software Tools for Technology Transfer*, 25, 407–426.
- Hahn, E.M., Hartmanns, A. (2016). A comparison of time- and reward-bounded probabilistic model checking techniques. In: *Dependable Software Engineering: Theories, Tools, and Applications: Second International Symposium, SETTA 2016, Beijing, China, November 9–11, 2016. Proceedings 2*. Springer, pp. 85–100.
- Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E. (2019). The quantitative verification benchmark set. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, pp. 344–350.
- Henriques, D., Martins, J.G., Zuliani, P., Platzer, A., Clarke, E.M. (2012). Statistical model checking for Markov decision processes. In: *2012 Ninth international conference on quantitative evaluation of systems*, IEEE, pp. 84–93.
- Karna, A.K., Chen, Y., Yu, H., Zhong, H., Zhao, J. (2018). The role of model checking in software engineering. *Frontiers of Computer Science*, 12, 642–668.
- Kim, Y., Kim, M. (2012). Hybrid statistical model checking technique for reliable safety critical systems. In: *2012 IEEE 23rd International Symposium on Software Reliability Engineering*, IEEE, pp. 51–60.

- Kwiatkowska, M., Norman, G., Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In: *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14–20, 2011. Proceedings 23*. Springer. pp. 585–591.
- Kwiatkowska, M., Norman, G., Parker, D. (2012). The PRISM benchmark suite. In: *9th International Conference on Quantitative Evaluation of SysTems*. IEEE CS Press, pp. 203–204.
- Kwiatkowska, M., Norman, G., Parker, D., Sproston, J. (2006). Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design*, 29(1), 33–78.
- Larsen, K., Legay, A., Nolte, G., Schlüter, M., Stoelinga, M., Steffen, B. (2022). Formal methods meet machine learning (F3ML). In: *International Symposium on Leveraging Applications of Formal Methods*. Springer. pp. 393–405.
- Legay, A., Viswanathan, M. (2015). Statistical model checking: challenges and perspectives. *International Journal on Software Tools for Technology Transfer*, 17, 369–376.
- Legay, A., Delahaye, B., Bensalem, S. (2010). Statistical model checking: an overview. In: *Runtime Verification*, Vol. 10. Springer Berlin Heidelberg, pp. 122–135.
- Legay, A., Lukina, A., Traonouez, L.M., Yang, J., Smolka, S.A., Grosu, R. (2019). Statistical model checking. In: *Computing and Software Science: State of the Art and Perspectives*. Springer. pp. 478–504.
- Milton, J.S., Arnold, J.C. (2002). *Introduction to Probability and Statistics: Principles and Applications for Engineering and the Computing Sciences*. McGraw-Hill Higher Education.
- Mohagheghi, M., Chaboki, B. (2020). Dirac-based reduction techniques for quantitative analysis of discrete-time markov models. In: *Topics in Theoretical Computer Science*. Springer, pp. 1–16.
- Mohagheghi, M., Salehi, K. (2020). Improving graph-based methods for computing qualitative properties of markov decision processes. *Indonesian Journal of Electrical Engineering and Computer Science*, 17(3), 1571–1577.
- Mohagheghi, M., Salehi, K. (2024). Improving probabilistic bisimulation for MDPs using machine learning. *Mathematics Interdisciplinary Research*, 9(2), 151–169.
- Mohagheghi, M., Karimpour, J., Isazadeh, A. (2020). Prioritizing methods to accelerate probabilistic model checking of discrete-time Markov models. *The Computer Journal*, 63(1), 105–122.
- Murphy, K.P. (2022). *Probabilistic machine learning: an introduction*. MIT press.
- Norman, G., Shmatikov, V. (2006). Analysis of probabilistic contract signing. *Journal of Computer Security*, 14(6), 561–589.
- Norman, G., Parker, D., Kwiatkowska, M., Shukla, S. (2005). Evaluating the reliability of NAND multiplexing with PRISM. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(10), 1629–1637.
- Pasupa, K., Sunhem, W. (2016). A comparison between shallow and deep architecture classifiers on small dataset. In: *2016 8th International Conference on Information Technology and Electrical Engineering (ICIT-TEE)*. IEEE, pp. 1–6.
- Qin, X., Xian, Y., Zutshi, A., Fan, C., Deshmukh, J.V. (2022). Statistical verification of cyber-physical systems using surrogate models and conformal inference. In: *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCP)*. IEEE, pp. 116–126.
- Salehi, K., Karimpour, J., Izadkhah, H., Isazadeh, A. (2019). Channel capacity of concurrent probabilistic programs. *Entropy*, 21(9), 885.
- Sen, K., Viswanathan, M., Agha, G. (2004). Statistical model checking of black-box probabilistic systems. In: *Computer Aided Verification: 16th International Conference, CAV 2004, Boston, MA, USA, July 13–17, 2004. Proceedings 16*. Springer, pp. 202–215.
- Sen, K., Viswanathan, M., Agha, G. (2005). On statistical model checking of stochastic systems. In: *Computer Aided Verification: 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6–10, 2005. Proceedings 17*. Springer, pp. 266–280.
- Song, J., Baek, Y.-M., Jin, M., Jee, E., Bae, D.-H. (2017). Sos gap slicer: Slicing sos goal and prism models for change-responsive verification of sos. In: *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, IEEE, pp. 546–551.
- Tadjudin, S., Landgrebe, D.A. (1996). A decision tree classifier design for high-dimensional data with limited training samples. In: *IGARSS'96. 1996 International Geoscience and Remote Sensing Symposium*, Vol. 1, IEEE, pp. 790–792.
- Wang, J., Sun, J., Yuan, Q., Pang, J. (2018). Learning probabilistic models for model checking: an evolutionary approach and an empirical study. *International Journal on Software Tools for Technology Transfer*, 20(6), 689–704.

- Wang, Y., Nalluri, S., Bonakdarpour, B., Pajic, M. (2021). Statistical model checking for hyperproperties. In: *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, IEEE, pp. 1–16.
- Younes, H.L., Simmons, R.G. (2002). Probabilistic verification of discrete event systems using acceptance sampling. In: *Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings 14*. Springer, pp. 223–235.
- Younes, H.L., Clarke, E.M., Zuliani, P. (2010). Statistical verification of probabilistic properties with unbounded until. In: *Brazilian Symposium on Formal Methods*. Springer. pp. 144–160.
- Younes, H.L., Kwiatkowska, M., Norman, G., Parker, D. (2006). Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer*, 8, 216–228.
- Zuliani, P. (2015). Statistical model checking for biological applications. *International Journal on Software Tools for Technology Transfer*, 17, 527–536.

M. Mohagheghi received his PhD in computer science from University of Tabriz in 2019, his master's in computer science from Sharif University of technology in 2008 and BSc in software engineering from Shahidbeheshty 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.

K. Salehi received his PhD in computer science from University of Tabriz in 2019, MSc and BSc in computer science from Sharif University of Technology and Yazd University in 2010 and 2007, respectively. He is currently an assistant professor of computer science in Shahrekord University, Iran. His main research involves machine learning and formal methods, as well as scientific computation.