IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 23, 2019, accepted June 18, 2019, date of publication June 24, 2019, date of current version July 12, 2019.

*Digital Object Identifier 10.1109/ACCESS.2019.2924639*

# A Probabilistic Assume-Guarantee Reasoning Framework Based on Genetic Algorithm

**YAN MA[1,3], ZINING CAO[1,4,5], AND YANG LIU[2,3]**

[1]College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
[2]College of Information Engineering, Nanjing University of Finance and Economics, Nanjing 210023, China
[3]School of Computing, National University of Singapore, Singapore 117417
[4]Key Laboratory of Safety-Critical Software (Nanjing University of Aeronautics and Astronautics), Ministry of Industry and Information Technology, Nanjing 210023, China
[5]MIIT Key Laboratory of Pattern Analysis and Machine Intelligence (Nanjing University of Aeronautics and Astronautics), Nanjing 210023, China

Corresponding author: Yang Liu (yliu@nufe.edu.cn)

This work was supported in part by the Singapore–U.K. Cyber Security of EPSRC under Grant EP/N020170/1, in part by the National Natural Science Foundation of China under Grant 61303022 and Grant 61572253, in part by the National Key Fundamental Research and Development Plans (973 Plans) under Grant 2014CB744903, in part by the Aviation Science Foundation of China under Grant 20150652008 and Grant 20185152035, in part by the Fundamental Research Funds for the Central Universities under Grant NZ2013306, in part by the National Science and Technology Support Program under Grant BAH29F01, in part by the Collaborative Innovation Center of Novel Software Technology and Industrialization, in part by the Natural Science Research Key Project of Jiangsu Higher Education Institutions under Grant 17KJA520002, and in part by the Nanjing Scientific and Technological Innovation Project for Outstanding Overseas Returnees.

**ABSTRACT** Probabilistic assume-guarantee reasoning is a theoretically feasible way to alleviate the state space explosion problem in stochastic model checking. The key to probabilistic assume-guarantee reasoning is how to generate the assumption. At present, the main way to automatically generate assumption is the L* (or symbolic L*) learning algorithm. An important limitation of it is that too many intermediate results are produced and need to be stored. To overcome this, we propose a novel assumption generation method by a genetic algorithm and present a probabilistic assume-guarantee reasoning framework for a Markov decision process (MDP). The genetic algorithm is a randomized algorithm essentially, and there are no intermediate results that need to be stored in the process of assumption generation, except the encoding of the problem domain and the training set. It can obviously reduce the space complexity of the probabilistic assume-guarantee reasoning framework. In order to improve the efficiency further, we combine the probabilistic assume-guarantee reasoning framework with interface alphabet refinement orthogonally. Moreover, we employ the diagnostic submodel as a counterexample for the guidance of augmenting training set. We implement a prototype tool for the probabilistic assume-guarantee reasoning framework and report the encouraging results.

**INDEX TERMS** Probabilistic assume-guarantee reasoning, genetic algorithm, interface alphabet, counterexample, stochastic model checking.

## I. INTRODUCTION

Nowadays, more and more real-life critical systems are subject to various phenomena of non-deterministic stochastic nature. Stochastic aspects are essential for, among others [1], [2]: 1) modeling unpredictable and unreliable system behavior (e.g., processor failure or message loss); 2) randomized algorithms (e.g., leader election or consensus algorithms); 3) model-based performance evaluation (i.e., forecasting system dependability and performance). Automatic formal verification of stochastic systems by model checking is called stochastic model checking or probabilistic model checking [2], [3]. The process of stochastic model checking combines classical model checking techniques (e.g., graph algorithms) and numerical methods for calculating probabilities (e.g., linear equation solving or linear programming) [4], so, it faces more severe state explosion problem, compared with classical model checking. This means that applying stochastic model checking to verify large scale systems remains challengeable, as pointed out by Turing Award winner Clarke at Turing Lecture [5].

The associate editor coordinating the review of this manuscript and approving it for publication was Mohammad Anwar Hossain.

### A. PROBLEM STATEMENT

As a prominent compositional verification method, assume-guarantee reasoning [6], [7] is a ''divide and conquer''

 This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/ 

paradigm. It replaces verification on the global system with localized components verification: each component is separately verified under corresponding contextual assumptions. Theoretically speaking, it is a feasible technique to alleviate state space explosion problem of stochastic model checking. Adapting assume-guarantee reasoning into stochastic model checking, i.e., probabilistic assume-guarantee reasoning, can be stated formally as follows. For a stochastic system $M$ of two components $M_0$ and $M_1$, a quantitative property specification $P$ to be verified, the probabilistic assume-guarantee reasoning process is captured by the asymmetric rule:

$$\frac{M_0 \models A \quad A \parallel M_1 \models P}{M_0 \parallel M_1 \models P}$$

where $A$ is an assumption on $M_1$'s environment, i.e., $M_0$. The verification of a property $P$ on a two-component system $M_0 \parallel M_1$ can be concluded by two steps: (i) checking that, under the assumption that property $A$ holds, $M_1$ is guaranteed to satisfy $P$, denoted $A \parallel M_1 \models P$; (ii) checking that $M_0$ satisfies the assumption $A$, denoted $M_0 \preccurlyeq A$.

We argue that probabilistic assume-guarantee mainly faces the following three issues, and named it as PAG (probabilistic assume-guarantee) problem: (a) How to generate the assumption. Assumption $A$ should contain enough information to guarantee the quantitative property $P$ under consideration, and be small enough for efficient analysis; (b) How to check the assume-guarantee tuple $A \parallel M_1 \models P$. The time and space cost of it and generating assumption should be significantly lower than that of stochastic model checking the entire system $M_0 \parallel M_1$. Otherwise, the probabilistic assume-guarantee reasoning is meaningless. (c) How to construct the counterexample. Providing counterexample is the important advantage of stochastic model checking, when the quantitative property is not satisfied on the stochastic system. In addition to being diagnostic information, it can also afford guidance for assumption refinement.

The core and key issue of solving PAG problem is how to generate the assumption. If a suitable assumption is obtained, the original system satisfies the property; otherwise, the system does not satisfy the property or space cost is too large to fit within the physical limits of the computer memory. In this sense, the process of generating assumption is just the process of probability assume-guarantee reasoning.

### B. RELATED WORK

At present, there is not too much related work about probabilistic assume-guarantee reasoning. According to the method of assumption generation, we divide the existed work into four categories: (1) probabilistic assume-guarantee reasoning by manual interaction (PAG-MI), (2) probabilistic assume-guarantee reasoning by L* learning algorithm (PAG-L*), (3) probabilistic assume-guarantee reasoning by symbolic L* learning algorithm (PAG-SL*), (4) probabilistic assume-guarantee reasoning by abstraction-refinement (PAG-AR).

#### 1) PAG-MI

Based on compositional analysis theory of stochastic system [8], Kwiatkowska *et al.* [9] introduced assume-guarantee reasoning into stochastic model checking for the first time, and presented the first probabilistic assume-guarantee reasoning framework for probabilistic safety properties over probabilistic automaton. It solved PAG problem in the following way: (a) required non-trivial manual effort to derive assumptions from Segala probabilistic automata; (b) used traditional stochastic model checking technique to checking the assume-guarantee tuple; (c) did not involve the counterexample.

This is the rudiment of probabilistic assume-guarantee reasoning framework. Assumption generation by manual interaction limits its practical usage.

#### 2) PAG-L*

To overcome the limitation of reference [9], Feng *et al.* [10] and Feng [11] used the L* [12], [13] learning algorithm to generate assumption, and proposed the first fully-automated probabilistic assume-guarantee verification framework for probabilistic safety properties over probabilistic automaton. It solved PAG problem in the following way: (a) generated assumption from probabilistic automata by L* learning algorithm; (b) used multi-objective stochastic model checking technique to check the assume-guarantee tuple; (c) adopted the method of reference [14] to construct the counterexample.

#### 3) PAG-SL*

He *et al.* [15] and Bouchekir and Boukala [16] used the symbolic data structure MTBDD (multiterminal binary decision diagram) to optimize the assumption generation, and presented the assume-guarantee reasoning by symbolic L* learning algorithm for probabilistic safety properties over MDP, respectively. They solved PAG problem in the following way: (a) generated assumption from MDP by a symbolic MTBDD-L* learning algorithm; (b) used traditional stochastic model checking technique to checking the assume-guarantee tuple; (c) adopted the method of reference [14] to construct the counterexample.

#### 4) PAG-AR

Komuravelli *et al.* [17] proposed an assume-guarantee framework for checking strong simulation between a probabilistic system and a quantitative specification, in which both the system and specification are expressed as nondeterministic labeled probabilistic transition systems (LPTSes). It solved PAG problem in the following way: (a) generated assumption by abstract-refinement; (b) used a strong simulation relationship to check the assume-guarantee triple; (c) adopted the method of reference [18] to construct the counterexample.

This framework relies on partitioning the explicit state space and strong simulation to construct assumptions, which usually lead to the assumptions with larger sate space. Moreover, it is just a theoretical framework and there is not the implemented tool.

### C. OUR CONSTRIBUTIONS

At present, the solutions for PAG problem, i.e., PAG-SL* and PAG-L*, can automatically generate the sound and complete assumption, but the learning process must store the whole computation history in order to continuous queries. In other words, all the intermediate results should be stored as the subproblem solutions in the process of assumption generation. This leads to probabilistic assume-guarantee reasoning framework needs too many memory spaces [19]. Memory consumption is the major obstacle for PAG-SL* and PAG-L* frameworks. They will not get the assumption from the large system component, if the space cost of generating assumption process is too large to fit within the physical limits of the computer memory.

For dealing with it, we use the genetic algorithm [20]–[22] to generate assumption for the first time, and propose a fully-automated genetic algorithm-based probabilistic assume-guarantee reasoning (PAG-GA) framework for probabilistic safe property over MDP. Genetic algorithm is essentially a randomized algorithm, and its correctness is guaranteed by satisfying all the constraints specified in the training set. So, in this framework, there are not intermediate results need to be recorded. Only the encode of the problem domain and the training set need to be recorded. This largely reduces the space complexity with respect to assumption generation. In order to improve the time efficiency, we first combine probabilistic assume-guarantee reasoning framework with an orthogonal technique, interface alphabet refinement [23], in this work. We construct the assumption with a small subset of the interface alphabet, and add actions to the necessary alphabet until the required property is shown to be hold or violated in the system, which are decided by counterexample analysis. Moreover, we employ the diagnostic submodel [24] as counterexample for guidance of training set augment. It will be useful for reducing time consumption of assumption generation. We solve PAG problem in the following way: (a) generate assumption from MDP interface alphabet by a genetic algorithm; (b) use multi-objective stochastic model checking technique to checking the assume-guarantee tuple; (c) adopt the method of reference [24] to construct the counterexample.

### D. ORGANIZATION OF THE PAPER

The rest of the paper is organized as follows. The basic definitions and terminology used throughout the paper is given in section 2. Section 3 presents how to generate assumption by optimized genetic algorithm. A specific description of our framework is in section 4. The experimental results are presented in section 5. Conclusion is in final section.

## II. PRELIMINARIES

### A. MDP

In our framework, Markov decision process (MDP) [2] is used as the stochastic system model. We use $Dist(S)$ to denote the set of all discrete probability distributions over a set $S$.

*Definition 1 (MDP):* A finite Markov decision process (MDP) over a set of propositions AP, is formally a tuple $M = (S, \bar{s}, \alpha_M, T, L)$, where

$S$ is a (finite or countable) set of states;

$\bar{s}$ is the initial state; $\alpha_M$ is a set of actions;

$T : S \to 2^{\alpha_M \times \mathrm{Dist}(S)}$ is the transition function.

$L : S \to 2^{AP}$ is a labeling function.

Let $supp(\mu) = \{s \in S | \mu(s) > 0\}$, an infinite path in an MDP $M$ is an infinite sequence $\pi = (s_0, a_0, \mu_0), (s_1, a_1, \mu_1), \ldots$ such that $s_0 \in \bar{s}$, $(a_i, \mu_i) \in T(s_i)$ and $s_{i+1} \in supp(\mu_i)$ for all $i \geq 0$. A finite path in $M$ is a finite prefix $\pi = (s_0, a_0, \mu_0), (s_1, a_1, \mu_1), \ldots s_n$ of an infinite path in $M$ with last state by $last(\pi) = s_n$. $Path_M$ denote the set of all infinite or finite paths over $M$.

*Definition 2 (Interface Alphabet):* Let $M_0$ and $M_1$ be component MDPs, and $G$ be a safety property. The interface alphabet $\alpha_A$ of $M_0$ is defined as: $\alpha_A = (\alpha_{M_0} \cup \alpha_G) \cap \alpha_{M_1}$, where $\alpha_G$ is alphabet in G.

*Definition 3 (Parallel Composition):* Let $M_0 = (S_0, \bar{s}_0, \alpha_{M_0}, T_0, L_0)$ and $M_1 = (S_1, \bar{s}_1, \alpha_{M_1}, T_1, L_1)$ be component MDPs. The parallel composition of $M_0$ and $M_1$, denoted $M_0||M_1$, is given by MDP $M_0||M_1 = (S_0 \times S_1, (\bar{s}_0, \bar{s}_1), \alpha_{M_0} \cup \alpha_{M_1}, T_{M_0||M_1}, L_0(s_0) \cup L_1(s_1))$ where $T_{M_0||M_1}$ is defined such that $(s_0, s_1) \xrightarrow{a} \mu_0 \times \mu_1$ iff one of the following holds:

$---s_0 \xrightarrow{a} \mu_0, s_1 \xrightarrow{a} \mu_1$ and $a \in \alpha_{M_0} \cap \alpha_{M_1}$

$---s_0 \xrightarrow{a} \mu_0, \mu_1 = \eta_{s_1}$ and $a \in (\alpha_{M_0} \backslash \alpha_{M_1}) \cup \{\tau\}$

$---s_1 \xrightarrow{a} \mu_1, \mu_0 = \eta_{s_0}$ and $a \in (\alpha_{M_1} \backslash \alpha_{M_0}) \cup \{\tau\}$

*Definition 4 (Projection):* For a path $\pi$, its *trace* $tr(\pi)$ is the sequence $< a_0, a_1, \ldots >$ of its action labels, after removal of any'' internal'' action $\tau$. For a trace t, we denote by $t \restriction_\alpha$ the projection of $t$ onto a subset $\alpha$ of its alphabet.

*Definition 5 (Alphabet Extension):* Given an MDP $M$ and an alphabet $\alpha$. The alphabet extension of $M$ with alphabet $\alpha$, denoted $M[\alpha]$, is obtained from $M$ by adding an a-labeled self-loop to every state for each $a \in \alpha \backslash \alpha_M$, where $\alpha_M$ is alphabet in $M$.

An adversary $\sigma$ of an MDP $M$ is a function from paths to pairs of actions and distributions, it is a resolution of nondeterminism. An adversary $\sigma$ is called simple if it only looks at the last state in a path, i.e., if it is a function $Adv_M : S \to (\alpha_M \times Dist(s)) \cup Dirac$. For a state $s \in S$ and an adversary $\sigma$, $Pr_s^\sigma$ and $Pr_M^\sigma$ denote the probability measurement over $Path_M$ and MDP $M$, respectively.

### B. COUNTEREXAMPLE

Model checking a probabilistic safety property $< G >_{\geq pG}$ (or $< G >_{\leq pG}$) on an MDP $M$, rather than checking a reachability or persistence property, requires computation of the minimum (or maximum) probability $Pr_M^{min}(G)$ (or $Pr_M^{max}(G)$). This can be reduced to computing the probability of accepting runs in the product MDP $M \otimes G^{err}$ by standard automata-based techniques for stochastic model checking, where $G^{err}$ is a DFA (deterministic finite automaton) that represents bad prefixes of $G$.

Given an MDP $M$ and regular safety with lower-bound properties $< G >_{\geq pG}$, if $M \not\models < G >_{\geq pG}$ in state $s$, then the probability sum of all paths $Path_{M^\sigma} \subseteq (M \otimes G^{err})^\sigma \restriction_{\alpha M}$ which are from $s$ and satisfy the formulas $G$ exceeds $1 - pG$. Generating a counterexample $(\sigma, c)$ for $M < G >_{\geq pG}$ can be reduced to the problem of computing the maximum probability of reaching an accepting state in $M \otimes G^{err}$ (the detailed information can be seen in [1]). I.e.,

$$M \not\models < G >_{\geq pG} \Longleftrightarrow \sup_{\sigma \in Adv_M} . Pr^{\sigma}_{M \otimes G^{err}} \{\pi \in Path^{\sigma}_{M \otimes G^{err}} | \pi \in c \,\&\, \pi \models M \otimes G^{err}\} > 1 - pG.$$

The counterexample generation for regular safety property with upper-bound probability ($M \not\models < G >_{\leq pG}$) can be reduced to lower-bound probability properties.

### C. DFA (DETERMINISTIC FINITE-STATE AUTOMATON)

Definition 6 (Deterministic finite-state automaton, DFA). A DFA [25] is a quintuple $\mathcal{M} = (Q, \alpha_{\mathcal{M}}, \delta, q_{init}, F)$, where

$Q$ is a finite nonempty set of states;
$\alpha_{\mathcal{M}}$ is a finite nonemepty set, called alphabet;
$\delta : Q \times \alpha_{\mathcal{M}} \to Q$ is the state transition function;
$q_{init} \in Q$ is the initial state;
$F \subseteq Q$ is a set of accepting states.

A run of $\mathcal{M}$ on a finite state $\omega = \omega_1 \ldots \omega_n \in \alpha^*_{\mathcal{M}}$ is a sequence $q_0 \xrightarrow{\omega_1} q_1 \xrightarrow{\omega_2} \ldots q_n$, where $q_0$ is the initial state of $\mathcal{M}$ and $q_{i+1} = \delta(q_i, \omega_{i+1})$ for $i \in \{0, 1, \ldots n-1\}$. If $q_n \in F$, it is called accepted. The language accepted by $\mathcal{M}$ is defined as $\mathcal{L}(\mathcal{M}) = \{\omega \in \alpha^*_{\mathcal{M}} | \text{there is an accepted run of } \mathcal{M} \text{ on } \omega\}$.

A language $\mathcal{L}$ is prefix-closed if for every word $\omega \in L$, all prefixes of $\omega$ are in $\mathcal{L}$. A DFA $\mathcal{M}$ is prefix-closed if its language $\mathcal{L}(\mathcal{M})$ is prefix-closed. It follows that a prefix-closed DFA has only one nonfinal state with transitions only to itself.

### D. GENETIC ALGORITHM

A genetic algorithm (GA) [20]–[22] is a heuristic search mechanism, which is inspired by the process of natural selection, i.e., survival of the fittest. It belongs to the larger class of evolutionary algorithms (EA). Genetic algorithm begins with a set of a random population in which each individual is encoded into a string or chromosome. Then evaluate the fitness of each individual in the current population, and select the fittest individual as the parents to reproduce the next generation by crossover and mutation. The fittest parents and the new offspring will form a new population. The algorithm terminates if either a fitness value has been satisfied, or a maximum number of generations has been produced. The standard genetic algorithm is shown in Algorithm 1.

**Algorithm 1** Standard Genetic Algorithm

Generate the initial population;
Compute fitness;
Repeat
  Selection; Crossover; Mutation; Compute fitness;
Until fitness value has been satisfied, or a maximum number of
generations have been produced.

## III. GENERATING ASSUMPTION BY GENETIC ALGORITHM

For dealing with too many intermediate results needs to be stored, we propose an optimized genetic algorithm for generating assumption in this section. It generates an assumption represented by a DFA from a component model MDP for probabilistic safety properties, and iteratively improves the accuracy guided by spurious counterexample. Firstly, a training set is initialized by W method [25] to generate a rough assumption. Then, it is an iterative process: refining the assumption or report a counterexample of the property over the stochastic system, according to the results of stochastic model checking assume-guarantee tuple.

### A. REPRESENTING ASSUMPTION

The assumption in our framework is defined as a DFA $\mathcal{M} = (Q, \alpha_{\mathcal{M}}, \delta, s_{init}, F)$. We should devise a suitable representation scheme in genetic algorithm for the states, actions, transition function, an initial state and final states of DFA, respectively.

**--Representation of states**

We encode elements of $Q$ as the Boolean value. Each element $e \in Q$ is assigned a unique vector of Boolean values $(x_1, x_2, \ldots x_n)$, such that $x_i \in \{0, 1\}$, $n$ meets $2^{n-1} < |Q| \leq 2^n$, $|Q|$ is the number of elements in $Q$. $|Q|$ is not always exact power of 2, there will be some vectors are ignored, which do not correspond to any element of $Q$. We define a characteristic function $f_Q : \{0, 1\}^n \to \{0, 1\}$ over $Q$, which maps $e$ represented by $(x_1, x_2, \ldots x_n)$, onto 1 if $e \in Q$, and maps it onto 0, otherwise.

**--Representation of actions**

Since actions are also a finite set, they can be encoded the same as the representation of states of DFA [26].

**--Representation of transition function**

Transition function can be represented as Boolean vectors by considering the characteristic function of a binary encode. The transition function: $Q \times \alpha_{\mathcal{M}} \to Q$ is a subset of $Q \times \alpha_{\mathcal{M}} \times Q$. Let states $q$ and $q'$ be represented as $(x_1, x_2, \ldots x_n)$ and $(x_1^{'}, x_2^{'}, \ldots x_n^{'})$, respectively. And a letter" $a$" is represented as $(v_1, v_2, \ldots v_m)$, where $a \in \alpha_{\mathcal{M}}$. Thus, the transition $q \xrightarrow{a} q'$ can be represented by a 3-tuple boolean vectors $((x_1, x_2, \ldots x_n), (v_1, v_2, \ldots v_m), (x_1^{'}, x_2^{'}, \ldots x_n^{'}))$, where $f_Q(x_1, x_2, \ldots x_n) \bigwedge f_{\alpha_{\mathcal{M}}}(v_1, v_2, \ldots v_m) \bigwedge f_Q\left(x_1^{'}, x_2^{'}, \ldots x_n^{'}\right) = 1$. This is easy to interpret and transform to Boolean function, and the space complexity of it is $O(|\alpha_{\mathcal{M}}| \cdot |Q|^2)$.

**--Representation of initial state**

Without loss of generality, considering there is only one initial state in a DFA, we can simply appoint $n$-bit vector to represent it. We initialize $f_Q(x_1, x_2, \ldots x_n) = 1$, which is not necessarily a correct guess.

**--Representation of final states**

There may be more than one state in the final state set, we can allocate $|Q|$-bit vector represent them,

i.e., $(b_1, b_2, \ldots b_{|Q|})$. Meanwhile, $b_1 \bigvee b_2 \bigvee \ldots \bigvee b_{|Q|}$ is true, which means several of them can be selected as final states, and none is not allowed.

### B. GENETIC OPERATORS

We use fitness proportionate selection operator, uniform crossover operator and mutation operator of genetic algorithm to manipulate process of assumption generation.

**--Fitness proportionate selection operator**

Selection operator assigns the reproductive opportunities to each individual (i.e., candidate assumption), in the population. The fitness proportionate selection operator selects individuals for mating according to the proportional to their fitnesses $\Pr(h_i)$. Assuming that there are $n$ individuals, the probability of selecting individual $h_i$ from population assumptions is: $\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{n} Fitness(h_j)}$, where $Fitness(h_i)$ is the fitness value of $h_i$. In other words, the higher the fitness value of an individual, the more possibility it will be preserved.

**--Uniform crossover operator**

The crossover operator produces two new candidate assumptions (called children) from two selected individuals (called parents, i.e., last candidate assumptions), by copying selected bits from each parent. Let $p_c$ be the probability of crossover, the number of individuals needing the crossover operation is:$p_c * np$, where $np$ is the number of individuals in the population. The uniform crossover operator creates two candidate children assumptions by crossover mask. The crossover mask is assigned as a random bit string, in which each bit is chosen at random and independent of the others. If the value of bit at position $i$ of the mask is "1", it indicates that uniform crossover operator exchange the bits of parents at corresponding position; if it is "0", uniform crossover operator does nothing.

**--Mutation operator**

Mutation operator transforms one candidate individual into a new one: it chooses a single bit or some bits at random, and produces the small random changes to the bit string, then changes values of a bit or some bits. Let $p_m$ be the probability of mutation, the number of individuals needing the mutation operation is: $p_m * np$.

### C. FITNESS FUNCTION

There are two types of training set X produced by W method [25], one is the words labeled "+" that are accepted by the assumption, and the other is the words labeled "−" that are rejected by the assumption. So, the target assumption DFA of iterative process can be distinguished as a classifier. The fitness value of each assumption $h_i$ is measured by its precise classification on the training set X:

$$Fitness(h_i) = \begin{cases} 0.0 & \exists q.f_q(q) = 0; \\ correct(h_i) & otherwise \end{cases}$$

where $correct(h_i)$ is the percent of all elements of training set correctly classified by assumption $h_i$.

### D. OPTIMIZED GENETIC ALGORITHM FOR GENERATING ASSUMPTION

The convergence of standard genetic algorithm is guaranteed by many theoretical results, which is independent of the specific application field. However, the local optimum always exists, which is the well-known premature problem. In our framework, we use the following strategies to prevent premature problems, and present an optimized genetic algorithm for generating assumption.

**--Adaptive probability of mutation**

If the parameters of mutation and crossover variate with the genetic evolution, genetic algorithm with self-organizing characteristic is apt to more robust and global optimized.

We adopt mutation strategy proposed in [21], in which the probability of mutation $p_m$ is related with Hamming distance. The Hamming distance between two individuals is the number of positions at which the corresponding bits are different, i.e., $\text{Ham} : \{0,1\}^n \times \{0,1\}^n \to Int$. In each iteration, the best assumption $h_{best}$ and the worst assumption $h_{worst}$ are stored. The probability of mutation $p_m$ is defined as:

$$p_m = p_m^{min} + (p_m^{max} - p_m^{min}) \times \frac{nh - Ham(h_{best}, h_{worst})}{nh}$$

where $p_m^{max}$ and $p_m^{min}$ are the maximum and minimum fixed probability of mutation fixed by genetic algorithm, $nh = |h_{best}| = |h_{worst}|$ is the length of assumption.

**--Inertia punishment**

In each iteration, we record the best fitness value. If a continuous sequence owns the same best fitness value, and the length of it exceeds an allowed threshold MAX_STABILITY, they indicate that the population get into local optimum, and the population evolves inertially. To punish the population's inertia, we temporarily change the probability $p_m$ of mutation with a relatively large number (approximately equals to 1). This change for the probability of mutation can be seen as the environmental disaster.

**--Probabilistic crossover**

Probabilistic crossover we adopted is derived from the standard uniform crossover [27]. For the probabilistic crossover operator, the probability of parents to be chosen for passing its variable to the offspring is proportional to its fitness value. Formally, given parents $h_1$ and $h_2$, the crossover mask Mask is:

$$Mask[i] = \begin{cases} 0, & with\ probability\ p_{mask} = \frac{Fitness(h_2)}{Fitness(h_1) + Fitness(h_2)} \\ 1, & with\ probability\ 1 - p_{mask} \end{cases}$$

Operators of adaptive probability of mutation and inertial penalty aim at keeping the diversity of population. These strategies can help the algorithm jump out of the local optimum and prevent precocity effectively. Meanwhile, probabilistic crossover can lead the algorithm to progress towards the optimum solution. The Optimized genetic algorithm for generating assumption is shown in Algorithm 2. It iteratively

generates a new population, until all the elements of training set are correctly recognized.

**Algorithm 2** Optimized Genetic Algorithm for Generating Assumption

Input:
*Fitness* fitness function; *np* the number of the population;
$p_c$ crossover rate; $p_m$ mutation rate; X training set;
Output:
An assumption $A_i = h_{best}$ with the highest fitness value
Process:
Represent assumption via binary encoding;
Generate *np* assumption randomly and form a population;
For each assumption *h*, compute*Fitness(h)*;
Max←maximum fitness value of the current population;
While(max<1.0) do
Update the chromosomes by selection, crossover and mutation
operations;
Evaluate the fitness of each assumption;
Select assumption to form a new population;
For each assumption *h*, compute fitness(*h*);
Max←maximum fitness value fitness *Fitness*($h_{best}$) of the
current population;

## IV. PROBABILISTIC ASSUME-GUARANTEE REASONING

### A. PROBABILISTIC ASSUME-GUARANTEE REASONING SEMANTICS

Probabilistic assume-guarantee reasoning in this work is the queries of the form $< A >_{\geq pA} M < G >_{\geq pG}$, where $< G >_{\geq pG}$ is probabilistic safety property, and $< A >_{\geq pA}$ is the probabilistic assumption which can be seen as probabilistic safety property. Informally, this means that "whenever $M$ is a part of a system satisfying property $A$ with probability at least $pA$, the system will guarantee property $G$ with probability at least $pG$".

*Definition 7 (Probabilistic Assume-Guarantee Reasoning Semantics):* If $< A >_{\geq pA}$ and $< G >_{\geq pG}$ are probabilistic safety properties, $M$ is an MDP and $\alpha G \subseteq \alpha A \cup \alpha M$, then:$< A >_{\geq pA} M < G >_{\geq pG}$. This is equivalent to: $\forall \sigma \in Adv_{M[\alpha A]}. Pr^{\sigma}_{M[\alpha A]}(A) \geq pA \Longrightarrow Pr^{\sigma}_{M[\alpha A]}(G) \geq pG$.

With the above definitions, we interest in the following asymmetric rule (ASMY). For MDPs $M_0$, and $M_1$, probabilistic safety properties $< A >_{\geq pA}$ and $< G >_{\geq pG}$ such that $\alpha A \subseteq \alpha M_0$ and $\alpha G \subseteq \alpha M_1 \cup \alpha A$:

$$\frac{< true > M_0 < A >_{\geq pA} \quad < A >_{\geq pA} M_1 < G >_{\geq pG}}{< true > M_0 \parallel M_1 < G >_{\geq pG}}$$

In other words, verifying $M_0 \parallel M_1 \models < G >_{\geq pG}$ can be done compositionally by stochastic model checking a safety property $< A >_{\geq pA}$ on $M_0$ and a assume-guarantee tuple on $M_1$. Given an MDP $M_1$, regular safety properties $A$, $G$ and a fixed value of $pA$, the tightest lower-bounded interval $I_G \subseteq [0, 1]$ can be computed by multi-objective stochastic model checking. It can be represented as $< A >_{\geq pA} M_1 < G >_{I_G=?}$. Similarly, the widest lower-bounded interval $I_A \subseteq [0, 1]$ can also be computed by multi-objective stochastic model checking. It can be represented as $< A >_{I_A=?} M_1 < G >_{\geq pG}$. Intuitively speaking, these inquiries enable us to compute the strongest possible guarantee under some assumption $< A >_{\geq pA}$ and the weakest possible assumption under a particular $< G >_{\geq pG}$. What we need to be aware is $I_A = \emptyset$. This means that even under the strongest possible assumption $< A >_{\geq 1}, < G >_{\geq pG}$ can not be guaranteed.

### B. COUNTEREXAMPLES FOR PROBABILISTIC SAFETY PROPERTIES

Counterexamples in PAG-L* and PAG-SL* are represented by paths [14], which often causes the number of paths is too big. Moreover, it is difficult to construct, since it is NPC problem sometimes. In reference [24], we defined the diagnostic sub-graph as counterexample for probabilistic reachability over DTMC, and extended PSO algorithm with heuristic (HPSO) to generate it.

In this framework, we generalize the diagnostic submodel [24] to express the counterexample of probabilistic safety properties over MDP. The counterexample represented by diagnostic submodel has rich information, and is compact about states and transitions.

*Definition 8 (Diagnostic Submodel):* Let $M$ be an MDP, $\sigma$ a deterministic simple adversary, a DTMC $M^{\sigma} = (S, s_{init}, \alpha_M, T)$ can be gotten under $\sigma$ . If $c$ is a set of paths of $M^{\sigma}$, the diagnostic submodel $M^{\sigma,c} = (S', s'_{init}, \alpha'_M, T')$ is part of $M^{\sigma}$ with $S' \subseteq S, s'_{init} \subseteq s_{init}, \alpha'_M \subseteq \alpha_M$. For distribution $\mu \in Dist(S)$ we define the subdistribution $\mu^c(s') = \mu(s)$ if the state $s'$ appears in some path in $c$, and $\mu^c(s') = 0$, otherwise. For each $s \xrightarrow{a} \mu$ in $T$, $T'$ contains $s \xrightarrow{a} \mu$ if and only if $\mu^c$ is nonempty.

Note that, the diagnostic submodel $M^{\sigma,c}$ is a part of $M$ according the definition of $T'$. We can straightforward conclude that $M$ strongly probabilistic simulates [28] $M^{\sigma,c}$, i.e., $M^{\sigma,c} \preccurlyeq M$. Strongly probabilistic simulation preserves safety properties, we can get $M^{\sigma,c} \models < G >_{\geq pG}$ if $M \models < G >_{\geq pG}$. Meanwhile, strongly probabilistic simulation is compositional, we can get: (1) $M^{\sigma,c} || M' \models < G >_{\geq pG}$ if there is an MDP $M'$ and $M || M' \models < G >_{\geq pG}$; (2) $M^{\sigma,c} || M' \not\models < G >_{\geq pG}$ if there is an MDP $M'$ and $M || M' \not\models < G >_{\geq pG}$.

### C. PROBABILISTIC ASSUME-GUARANTEE REASONING FRAMEWORK

There are two auxiliary sources in the process of generating an assumption: (1) Membership oracle answers whether a word is accepted by $M_0$, which is performed by multi-objective stochastic model checking; (2) Equivalence oracle

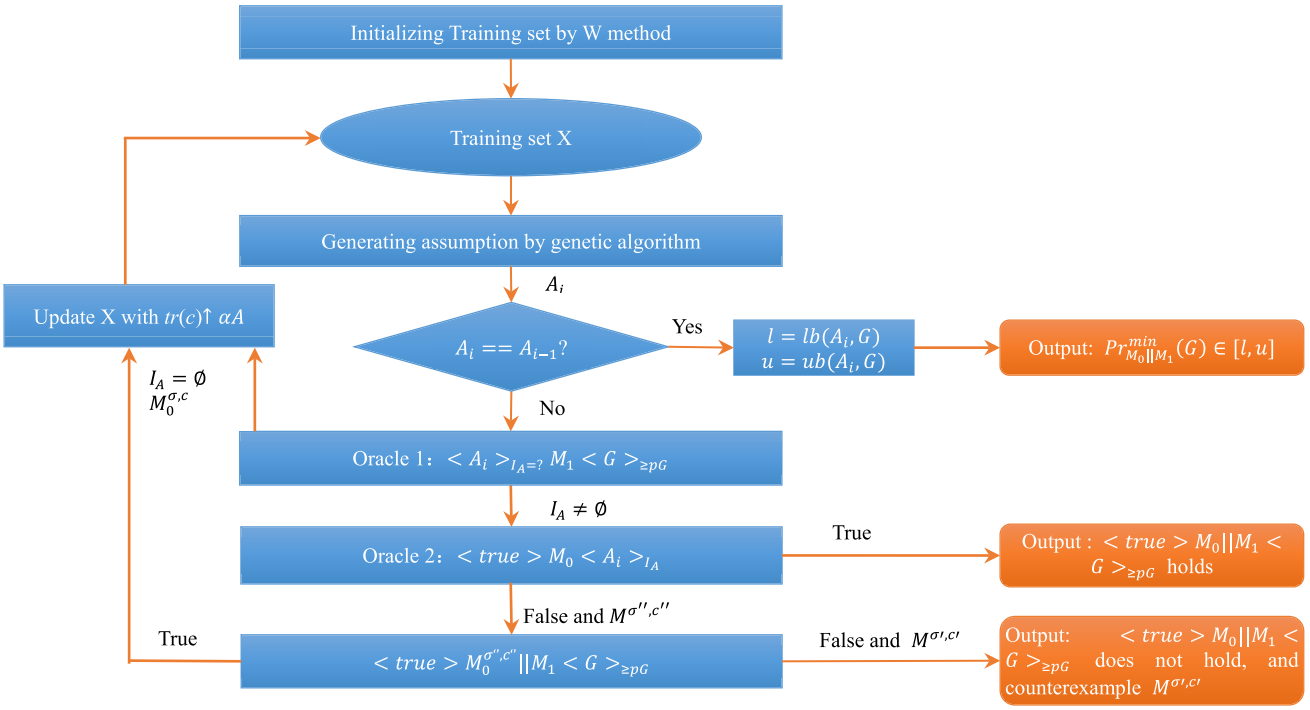



**FIGURE 1.** Probabilistic assume-guarantee reasoning framework.

answers whether the assumption $A$ is equivalent to $M_0$ with true or counterexamples, which is performed by traditional stochastic model checking.

Let $M_0$ and $M_1$ be components, $< G >_{\geq pG}$ be a probabilistic safety property, $\alpha A$ be the interface alphabet. Our goal is to judge $M_0 \parallel M_1 \models < G >_{\geq pG}$ by producing a probabilistic assumption $< A >_{\geq pA}$ over $\alpha A$, and the genetic algorithm-based probabilistic assume-guarantee reasoning framework is shown in Fig. 1.

**--Initialization of the training set**

The training set X are randomly generated by the method W [27]. It requires that the automaton is a Mealy machine, so the DFAs should be transformed into Mealy machines [28].

First, we check whether the words in training set X can be accepted in $M_0$. The word $w \in \alpha_A^*$ is labeled "+", if $w \in \mathcal{L}(M_0)$. Otherwise, the word is labeled "−". Deciding whether $w \in \mathcal{L}(M_0)$ can be implemented with classical model checking, which is the membership queries. If result of model checking is true, then $w \in \mathcal{L}(M_0)$. Otherwise, it is false.

When the membership query of a word $w$ returns false, it indicates that the answer to any extended query of $w$ will be false. So, we can improve the efficiency of the algorithm by reducing the cost of member queries. For example, if the query for $< a >$ is false, then the teacher can return false for the queries $< a, a >, < a, b >, < a, b, b >$ without invoking the model checking.

**--Generating assumption**

According to Algorithm 2, the optimized genetic algorithm generates assumption $A_i$ of $M_0$ from the initial training set X constituted by accepted and rejected words. The obtained assumption may be not correct, which is decided by answering conjectures.

**--Answering conjectures**

The teacher will answer whether the assumption $A_i$ is appropriate for $M_0$. Probabilistic assumption $< A >_{\geq pA}$ needs to be satisfied in both $< true > M_0 < A >_{\geq pA}$ and $< A >_{\geq pA} M_1 < G >_{\geq pG}$. As shown in Fig.1, the teacher checks the above two premises with two separate oracles, respectively.

### 1) ORACLE 1

We first check the query $< A >_{I_A=?} M_1 < G >_{\geq pG}$ by multi-objective model checking to determine the widest interval $I_A \subseteq [0, 1]$. We can get the weakest possible assumption that guarantees $< G >_{\geq pG}$. However, $I_A = \emptyset$ is possible. This situation means $M_1$ would still not meet $< G >_{\geq pG}$, even under the assumption $< A >_{\geq 1}$. A probabilistic counterexample $M^{\sigma,c}$ is generated to indicate that $< A >_{\geq 1} M_1 < G >_{\geq pG}$ does not hold, which can be used to refine the assumption $A$. In other words, there is a trace (or some traces) which is currently included in $A$, but it (or they) should be excluded in fact, since it causes a violation of $< G >_{\geq pG}$. We use the counterexample $M^{\sigma,c}$ to generate traces set $\mathrm{Tr}_c = tr(c)$ for paths in $c$. The traces set $\mathrm{Tr}_c$ is restrict to the alphabet $\alpha A$, denoted as $Tr = tr(c) \uparrow_{\alpha A}$. We add the trace to training set X and refine the assumption by constructing all paths in $c$ meet $A$. If $I_A \neq \emptyset$, it is a nonempty, closed and lower bounded interval. This means that $< A >_{I_A}$ is a valid probabilistic safety property essentially. Oracle 2 will be checked.

2) ORACLE 2

Oracle 2 is invoked to discharge $< A >_{\geq pA}$ on $M_0$, i.e., verifying whether $< true > M_0 < A >_{\geq pA}$ is met. If it is, we can conclude that the assumption which satisfies both $< true > M_0 < A >_{\geq pA}$ and $< A >_{\geq pA} M_1 < G >_{\geq pG}$ is exist. The framework is terminated with an output $M_0||M_1 \models < G >_{\geq pG}$.Otherwise, we can get a counterexample $M^{\sigma'',c''} M\sigma'', c''$ to indicate that $M_0 \not\models < A >_{\geq pA}$. Then we analysis the counterexample $M^{\sigma'',c''}$ and determine whether it is a spurious counterexample for $M_0||M_1$.

**--Counterexample analysis**

To determine whether $M^{\sigma'',c''}$ is a spurious counterexample, we check whether $M^{\sigma'',c''}||M_1 \not\models < G >_{\geq pG}$. If it is, we can conclude that $M_0||M_1 < G >_{\geq pG}$, and a counterexample $M^{\sigma',c'}$ from $M^{\sigma'',c''}||M_1$ is generated to illustrate $< G >_{\geq pG}$ is refuted in $M_0||M_1$. Otherwise, $M^{\sigma'',c''}$ is a spurious counterexample, and we refine the assumption $A$ by adding a trace $t$ that comes from a counterexample showing $M_0 \not\models < A >_{\geq pA}$. Consider the case where $Tr = tr(c) \uparrow_{\alpha A}$ comprises a single trace $t$ which is likely $t||M_1 \models < G >_{\geq pG}$. We add the trace to training set X for generating the next new assumption.

**--Producing lower and upper bounds**

The trace from a counterexample is added into training set for generating new assumption. If it cannot induce the new assumption, i.e., no further conjectures are possible, the framework is terminated. In this case, the third possible output is presented, which is lower and upper bounds from the genetic algorithm.

For any assumption $A$, it is possible to produce lower and upper bounds on the minimum probability of safety property $G$. Firstly, $p_A^* = Pr_{M_0}^{min}(A)$ is computed. meanwhile, an adversary $\sigma \in Adv_{M_0}$ is generated, which obtains the minimum probability. Finally, the interval $I_G$ is got from checking the quantitative assume-guarantee query $< A >_{\geq p_A^*} M_1 < G >_{I_G=?}$. The upper bound is $ub(A, G) = min(I_G)$ and the lower bound is $lb(A, G) = Pr_{M^{\sigma,c}||M_1}^{min}(G)$.

*Theorem 1:* $lb(A, G) \leq Pr_{M_0||M_1}^{min}(G) \leq ub(A, G)$.

For lower bound, $M^{\sigma,c}$ is a sub-model of $M_0$, so $lb(A, G) = Pr_{M^{\sigma,c}||M_1}^{min}(G) \leq Pr_{M_0||M_1}^{min}(G)$ can be got from the fact that strong probabilistic simulation preserves safety properties [28]. For upper bound, $M_0 \models < A >_{\geq p_A^*}$ and $< A >_{\geq p_A^*} M_1 < G >_{\geq ub(A,G)}$ can be got by construction, thus, $Pr_{M_0||M_1}^{min}(G) \leq ub(A, G)$ is got from formulae (2).

The lower and upper bounds can be provided by the current assumption $A$. Meanwhile, they also can be provided by earlier assumptions which may produce tighter bounds.

### D. CORRECTNESS AND TERMINATION

The correctness of three outputs has been discussed in the above sections. Although the series of assumptions produced is not monotonic, the framework can terminate with the third output, i.e., the lower and upper bounds. Since $M_0$ is finite automata and genetic algorithm can converge to the optimal solution in a certain time, the framework can terminate with an assumption $A$ for $M_0$ or the assumption remains unchanged in two successive iterations in a given time.

**TABLE 1.** Basic information of randomized dining philosophers problem.

| Para ($N$) | MSMC (PRISM) | | | PAG | |
|---|---|---|---|---|---|
| | States | Time | Value | $\|M_1 \otimes G^{err}\|$ | $\|M_0\|$ |
| 5 | 93K | 0.7 | 0.799879 | 2868 | 97 |
| 6 | 917K | 4.8 | 0.399614 | 2868 | 956 |
| 7 | 9043K | 30.1 | 0.096542 | 28320 | 9440 |
| 8 | 89M | 328.1 | 0.046135 | 28320 | 93K |
| 9 | 879M | 3080 | 0.009763 | 279204 | 917K |
| 10 | 8662M | TO | -- | 279204 | 9043K |

### E. INTEGRATING WITH INTERFACE ALPHABET REFINEMENT

In order to improve the efficiency, we integrate interface alphabet refinement [23] with our framework. We construct the training set for generate assumption from a smaller interface alphabet set $\Sigma$ instead of $\alpha A$, where $\Sigma = \alpha G \cap \alpha A$. If the assumption is not appropriate, $Tr = tr(c) \uparrow_\Sigma$ is added to training set for generating new assumption. According to the correctness of the probabilistic assume-guarantee, if the framework returns true, $M_0||M_1 \models < G >_{\geq pG}$; If it returns false, this means that a counterexample $M_0^{\sigma,c\uparrow\Sigma}$ exists in $M_0$, which falsifies $M_0^{\sigma,c\uparrow\Sigma}||M_1 \models < G >_{\geq pG}$. It does not necessarily indicate that $M_0||M_1 \not\models < G >_{\geq pG}$; if it returns the interval $Pr_{M_0||M_1}^{min}(G) \in [l, u]$, this does not mean that no further conjectures are possible. In the above two situations, the alphabet $\Sigma$ should be extended according to algorithm 3, and then it is used to generate assumption again, until $\Sigma = \alpha A$.

**Algorithm 3** Interface Alphabet Refinement

1) **Initialize** $\Sigma$ such that $\Sigma = \alpha G \cap \alpha A$.
2) Use the above learning framework for $\Sigma$. If the framework returns true, then report true and go to step 5). If the framework returns false with counterexamples $M_0^{\sigma,c\Sigma}$ and $M_0^{\sigma',c'_\Sigma}$, go to the next step. If the framework return interval which is provided by the current assumption $A$ and $\Sigma \subset \alpha A$, go to step 4).
3) Perform **extended counterexample analysis** with $M_0^{\sigma,c\uparrow\Sigma}$ and $M_0^{\sigma',c'\uparrow\Sigma}$. if $M_0^{\sigma',c'\uparrow\alpha A}$, is a real counterexample, then report false and go to step 5). If $M_0^{\sigma',c'\uparrow\alpha A}$ is spurious and $\Sigma \subset \alpha A$, then **refine** $\Sigma$, which consists of adding to $\Sigma$ actions from $\alpha A$. Go to step 2).
4) Refine $\Sigma$ by adding all the actions from $(\alpha A \backslash \Sigma)$. Go to step 2).
5) END.

In the following, we further specify the above algorithm.

**--Alphabet initialization**

We initialize alphabet with those actions which are the alphabet in probabilistic safety property and $\alpha A$, i.e., $\alpha G \cap$

**TABLE 2.** Comparing of PAG-L*, PAG-SL* and PAG-GA on philosophers problem.

| Para (*N*) | PAG-L* | | | | PAG-SL* | | | | PAG-GA | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\|A\|$ | Size | Time | Bounds[*l*,*u*] | $\|A\|$ | Size | Time | Bounds[*l*,*u*] | $\|A\|$ | Size | Time | Bounds[*l*,*u*] |
| 5 | 8 | 48 | 2.3 | [0.799879,1] | 3 | 55 | 2.0 | [0.799879,1] | 6 | 32 | 1.7 | [0.799879,1] |
| 6 | 8 | 127 | 6.7 | [0.399617,1] | 3 | 136 | 7.1 | [0.399617,1] | 7 | 41 | 5.5 | [0.399614,1] |
| 7 | 20 | 316 | 67.1 | [0.057483,1] | 7 | 284 | 68.8 | [0.073572,1] | 12 | 43 | 43.0 | [0.126542,1] |
| 8 | 35 | 620 | 388.1 | [0.003464,1] | 11 | 453 | 278.6 | [0.003464,1] | 21 | 43 | 224.3 | [0.050135,1] |
| 9 | 103 | 1388 | 530.3 | [0.005112,1] | 25 | 953 | 540.4 | [0.006341,1] | 42 | 44 | 247.8 | [0.007507,1] |
| 10 | 130 | 2654 | 1145.6 | [0,1] | 25 | 1877 | 1035.6 | [0,1] | 57 | 56 | 935.2 | [0.001728,1] |

$\alpha A$. A good initialization of the alphabet can reduce time cost of assumption generation, as it results in fewer refinement iterations.

**--Extended counterexample analysis**

Extended counterexample analysis takes two counterexamples as inputs, which are counterexample $M_0^{\sigma,c\restriction\Sigma}$ returned by Oracle 2, and counterexample $M^{\sigma',c'\restriction\Sigma}$ returned by the original counterexample. Then, the framework (Fig.1) is modified to return $M_0^{\sigma,c\restriction\Sigma}$ and $M^{\sigma',c\restriction\Sigma}$ to be used in alphabet refinement. If $M^{\sigma',c'\restriction\alpha A}||M_1 \not\models < G >_{\geq pG}$, the counterexample $M^{\sigma',c'\restriction\alpha A}$ is obtained. Otherwise, the alphabet $\Sigma$ needs to be refined.

**--Alphabet refinement**

If the counterexample is spurious, the current alphabet $\Sigma$ needs to be refined so as to eliminate this counterexample. The interface alphabet refinement will add the actions in $\alpha A$ to the learning alphabet, which is discovered based on comparing $M_0^{\sigma,c\restriction\Sigma}$ wtih $M^{\sigma',c'\restriction\Sigma}$. The actions are all added, if they are different between $M_0^{\sigma,c\restriction\Sigma}$ and $M^{\sigma',c'\restriction\Sigma}$ accordingly. We randomly generate prefix DFAs with fixed number of alphabets in each iteration.

## V. IMPLEMENTATION AND RESULTS

We implement a prototype tool PAG-GA for the framework based on PAT [29]. The architecture of it is shown in Figure 2. A symbolic (MTBDD-based) implementation of PAG-GA is under development, which will be released as the open source software at PAT website (http://pat.comp.nus.edu.sg/). We use PRISM as front-end to parse the input files. The queries in the assumption generation process are performed by either PRISM or multi-objective model checking which is the extension of PRISM [30], [31].

All experiments are run on a PC with Intel Pentium CPU (2.20 GHz) and 8GB RAM. Time limit is set to 24 hours. We use "TO" to denote the verification cannot be completed within the specified time. We configure the parameters of genetic algorithm as follows: the number of population $np$ = 100, crossover rate $p_c$ = 0.6, minimum mutation rate $p_m^{min}$ = 0.01, and maximum mutation rate $p_m^{max}$ = 0.05. The maximum threshold of the same highest fitness value sequence MAX_STABILITY is 9; the inertia punishment

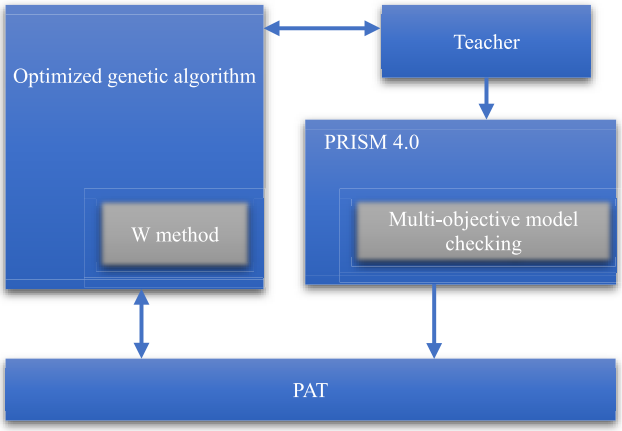



**FIGURE 2.** Architecture of PAG-GA.

**TABLE 3.** Basic information of randomized consensus protocol of aspnes and herlihy.

| Para ($N$, $R$, $K$) | PRISM | | | PAG | |
|---|---|---|---|---|---|
| | States | Time | Value | $\|M_1 \otimes G^{err}\|$ | $\|M_0\|$ |
| 2,3,2 | 5158 | 1.8 | 0.891667 | 391 | 337 |
| 2,3,20 | 40K | 105.7 | 0.987508 | 391 | 3217 |
| 2,4,2 | 21K | 3.7 | 0.988263 | 573 | 114K |
| 2,4,4 | 43K | 8.6 | 0.996501 | 573 | 432k |
| 3,3,2 | 1419K | 18971 | 0.770908 | 8843 | 4065 |
| 3,3,20 | 40M | TO | -- | 8843 | 38K |

mutation rate is 1.0; the number of best assumption preserved is 2.

We compare our work with monolithic stochastic model checking (MSMC) by PRISM, probabilistic assume-guarantee reasoning by L* learning algorithm (PAG-L*) and probabilistic assume-guarantee reasoning by symbolic L* learning algorithm(PAG-SL*), respectively. We also compare our probabilistic assume-guarantee reasoning framework with interface alphabet refinement and without alphabet refinement.

The first case is randomized dining philosophers problem [30]. $N$ philosophers sit around a circular table. Each of them can only eat food when he has both right and left forks. Each fork can be held by only one philosopher at one time. When a philosopher finishes eating, he needs to

**TABLE 4.** Comparing of PAG-L*, PAG-SL* and PAG-GA on consensus protocol of aspnes and herlihy.

| Para | PAG-L* | | | | PAG-SL* | | | | PAG-GA | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $(N, R, K)$ | $\|A\|$ | Size | Time | Bounds[$l$,$u$] | $\|A\|$ | Size | Time | Bounds[$l$,$u$] | $\|A\|$ | Size | Time | Bounds[$l$,$u$] |
| 2,3,2 | 7 | 34 | 17.3 | [0.891667,1] | 3 | 27 | 17.5 | [0.891667,1] | 6 | 17 | 15.1 | [0.891667,1] |
| 2,3,20 | 7 | 102 | 24.6 | [0.987508,1] | 3 | 74 | 22.5 | [0.987508,1] | 7 | 36 | 17.7 | [0.987508,1] |
| 2,4,2 | 12 | 509 | 120.8 | [0.988263,1] | 5 | 337 | 100.6 | [0.988263,1] | 10 | 52 | 53.2 | [0.988263,1] |
| 2,4,4 | 12 | 1011 | 115.4 | [0.996501,1] | 5 | 562 | 108.3 | [0.996501,1] | 12 | 90 | 71.8 | [0.996501,1] |
| 3,3,2 | 15 | 1835 | 677.4 | [0.771037,1] | 6 | 860 | 474.2 | [0.771037,1] | 12 | 331 | 115.4 | [0.770923,1] |
| 3,3,20 | 18 | 2060 | 912.1 | [0.978013,1] | 7 | 1787 | 553.3 | [0.978013,1] | 15 | 455 | 236.0 | [0.976230,1] |

put down both forks, and the forks become available to other philosophers at the same time. For randomized dining philosophers' problem, we model the composition of half of processes as $M_0$, and the other processes composed as $M_1$. The property $G$ to be verified is ''what is the probability that if a philosopher is hungry, he eats eventually''. The basic information and comparisons of PAG-L*, PAG-SL* and PAG-AG for this case are shown in TABLE 1 and TABLE 2, respectively.

The second case is randomized consensus protocol [31]. It consists of $N$ asynchronous processes that communicate via read/write shared registers. The processes proceed through possibly unboundedly many rounds; at each round, a shared coin protocol is parameterized by $K$. The protocol attempts to involve a distributed random walk: when the processes disagree, they use a shared coin-flipping protocol to decide their next preferred value. Achieving polynomial expected time depends on ensuring that the probability that all non-failed processes draw the same value is above an appropriate bound. For randomized consensus protocol, we model the composition of asynchronous processes as $M_0$, and the composition of coin-flipping processes as $M_1$. The property $G$ to be verified is ''what is the minimum probability that the asynchronous processes reach a consensus by round $R$''. The basic information and comparisons of PAG-L*, PAG-SL* and PAG-AG for this case are shown in TABLE 3 and TABLE 4, respectively.

The third case is synchronous leader election protocol [32]. Given a synchronous ring of $N$ processors, the protocol will be able to elect a leader (a uniquely designated processor) by sending messages around the ring. It proceeds in some rounds and is parameterized by a constant $K$. At the beginning of each round, all processors independently and uniformly choose a random number from $\{1, \ldots, K\}$ as the id. Then they pass their ids around the ring. If there is a unique id, the processor with the maximum unique id is elected as the leader, otherwise, the processors begin a new round again. The processors proceed in the form of synchronization. For synchronous leader election protocol, we model the composition of processors as $M_0$, and the composition of messages as $M_1$. The property $G$ to be verified is ''what is the probability that eventually a leader is elected''. The basic information and comparisons of PAG-L*, PAG-SL* and PAG-AG for this case are shown in TABLE 5 and TABLE 6, respectively.

**TABLE 5.** Basic information of synchronous leader election.

| Para | PRISM | | | PAG | |
|---|---|---|---|---|---|
| $(N, K)$ | States | Time | Result | $\|M_1 \otimes G^{err}\|$ | $\|M_0\|$ |
| 3,8 | 1031 | 0.7 | 1 | 181 | 76 |
| 3,10 | 2007 | 1.0 | 1 | 307 | 214 |
| 3,12 | 3466 | 1.3 | 1 | 429 | 392 |
| 3,14 | 5495 | 1.6 | 1 | 683 | 705 |
| 4,6 | 3902 | 1.4 | 1 | 472 | 1501 |
| 4,8 | 12306 | 11.0 | 1 | 215 | 2261 |
| 4,10 | 30014 | 38.4 | 1 | 441 | 5114 |
| 4,12 | 62222 | 163.6 | 1 | 885 | 10K |

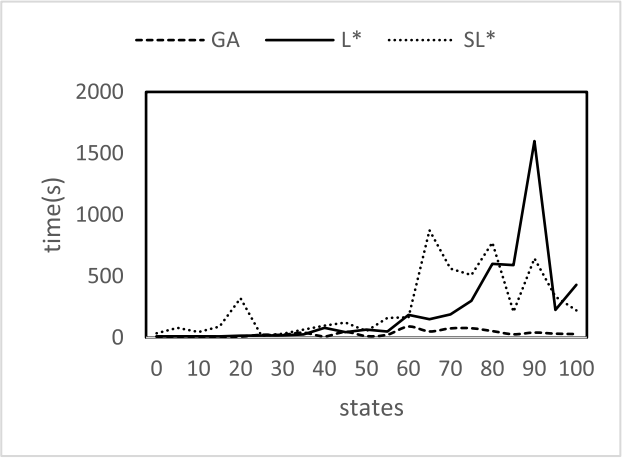



**FIGURE 3.** Time consumption of GA, L* and SL* with $|\Sigma| = 2$.

The ''MSMC (PRISM)'' columns give the number of states, the total time (in seconds) and the exact probability value obtained by monolithic stochastic model checking (MSMC) with PRISM. The ''PAG'' columns give the number of states of $M_0$ and $M_1 \otimes G^{err}$ in probabilistic assume-guarantee, where $G$ is the safety property to be checked. Column ''para'' represents the parameter value of the case. For each case, we report the number of states (''$|A|$'') of the assumption $A$, the total time (''Time'')in seconds, memory consumption(''Size'') in megabytes and bounds of guaranteed property $G$, respectively.

The results are very encouraging. In all cases, our framework PAG-GA successfully generates the assumption for verifying the corresponding property $G$. It also generates lower and upper bounds on the minimum probability, if $< G >_{\geq pG}$

**TABLE 6.** Comparing of PAG-L*, PAG-SL* and PAG-GA on synchronous leader election.

| Para $(N, K)$ | PAG-L* | | | | PAG-SL* | | | | PAG-GA | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\|A\|$ | Size | Time | Bounds[$l$,$u$] | $\|A\|$ | Size | Time | Bounds[$l$,$u$] | $\|A\|$ | Size | Time | Bounds[$l$,$u$] |
| 3,8 | 5 | 48 | 2.3 | [1,1] | 2 | 37 | 2.5 | [1,1] | 5 | 23 | 1.5 | [1,1] |
| 3,10 | 8 | 92 | 2.6 | [1,1] | 2 | 77 | 2.3 | [1,1] | 7 | 34 | 1.5 | [1,1] |
| 3,12 | 8 | 142 | 3.3 | [1,1] | 4 | 117 | 2.7 | [1,1] | 7 | 48 | 1.8 | [1,1] |
| 3,14 | 12 | 186 | 3.3 | [0.997590,1] | 4 | 133 | 2.5 | [0.997590,1] | 10 | 85 | 1.8 | [0.997590,1] |
| 4,6 | 12 | 155 | 3.5 | [1,1] | 4 | 148 | 3.4 | [1,1] | 10 | 72 | 1.9 | [1,1] |
| 4,8 | 14 | 484 | 3.7 | [1,1] | 4 | 451 | 3.2 | [1,1] | 13 | 92 | 3.4 | [1,1] |
| 4,10 | 17 | 718 | 52.8 | [0.872164,1] | 4 | 702 | 61.5 | [0.872164,1] | 13 | 105 | 35.9 | [0.872164,1] |
| 4,12 | 17 | 1227 | 79.4 | [0.754773,1] | 4 | 785 | 66.4 | [0.754773,1] | 13 | 216 | 53.8 | [0.754773,1] |

**TABLE 7.** Comparing of PAG-GA on philosophers with and without interface refinement.

| Para $(N)$ | PAG-GA | | | | PAG-GA with interface alphabet refinement | | | |
|---|---|---|---|---|---|---|---|---|
| | $\|A\|$ | Size | Time | Bounds[$l$,$u$] | $\|A\|$ | Size | Time | Bounds[$l$,$u$] |
| 5 | 6 | 32 | 1.7 | [0.799879,1] | 5 | 21 | 1.2 | [0.799879, 0.799879] |
| 6 | 7 | 41 | 5.5 | [0.399614,1] | 6 | 28 | 4.9 | [0.399614, 0.399614] |
| 7 | 12 | 43 | 43.0 | [0.126542,1] | 10 | 29 | 31.6 | [0.096542, 0.286593] |
| 8 | 21 | 43 | 224.3 | [0.050135,1] | 14 | 36 | 112.7 | [0.046135, 0.195634] |
| 9 | 42 | 44 | 247.8 | [0.007507,1] | 21 | 41 | 125.3 | [0.009763,0.013572] |
| 10 | 57 | 56 | 935.2 | [0.001728,1] | 26 | 63 | 163.8 | [0.002487,0.023858] |

**TABLE 8.** Comparing of PAG-GA on consensus with and without interface refinement.

| Para $(N, R, K)$ | PAG -GA | | | | PAG-GA with interface alphabet refinement | | | |
|---|---|---|---|---|---|---|---|---|
| | $\|A\|$ | Size | Time | Bounds[$l$,$u$] | $\|A\|$ | Size | Time | Bounds[$l$,$u$] |
| 2,3,2 | 6 | 17 | 15.1 | [0.891667,1] | 5 | 12 | 3.7 | [0.891667,0.891667] |
| 2,3,20 | 7 | 36 | 17.7 | [0.987508,1] | 5 | 37 | 8.5 | [0.987508,0.987508] |
| 2,4,2 | 10 | 52 | 53.2 | [0.988263,1] | 8 | 43 | 18.4 | [0.988263,0.98263] |
| 2,4,4 | 12 | 90 | 71.8 | [0.996501,1] | 10 | 67 | 27.6 | [0.996501,1] |
| 3,3,2 | 12 | 331 | 115.4 | [0.770923,1] | 11 | 548 | 32.1 | [0.770908,1] |
| 3,3,20 | 15 | 455 | 236.0 | [0.976230,1] | 11 | 674 | 157.4 | [0.975040,1] |

is true on the compositional model $M_0 \parallel M_1$. The lower bounds produced by our framework is in coincidence with the result from PRISM, and yields the exact values in all cases. Compared with PRISM, the time and memory consumption of PAG-GA is significantly reduced, when the verified system scale is large. Even PRISM fails to verify consensus with parameters (3,3,20) in TABLE 3, our framework PAG-GA verifies it successfully. However, PAG-GA don not have the advantages of time over PRISM, if the verified system scale is small.

Compared with PAG-L* and PAG-SL*, PAG-GA has obvious advantage for all cases in terms of memory consumption, because it does not need to store the intermediated results. PAG-GA takes less time than PAG-L* and PAG-SL* to verify all cases. There are two reasons for this: 1) constructing counterexample in PAG-GA framework is more efficient than PAG-L* and PAG-SL*, which is shown in reference [24]; PAG-GA framework reduces the number of membership queries, as the population evolution of genetic algorithm. We cannot assert that PAG-GA is promising in time consumption than PAG-L* and PAG-L*, because of the randomness in genetic algorithm. They obtain the similar bounds of $G$ for all the cases. The state space of assumption generated by PAG-SL* is smaller than PAG-L* and PAG-GA. This is because PAG-SL* uses compact data-structure MTBDD to represent assumption. The state space of assumption generated by PAG-L* and PAG-GA are similar at the scale.

As shown in TABLE 7, 8 and 9, we compare PAG-GA with and without interface alphabet refinement for the above three cases. When our framework integrates the interface alphabet refinement orthogonally, it takes obvious less time, and

**TABLE 9.** Comparing of PAG-GA on synchronous leader election with and without interface refinement.

| Para | PAG-GA | | | | PAG-GA with interface alphabet refinement | | | |
|---|---|---|---|---|---|---|---|---|
| $(N, K)$ | $\|A\|$ | Size | Time | Bounds$[l,u]$ | $\|A\|$ | Size | Time | Bounds$[l,u]$ |
| 3,8 | 5 | 23 | 1.5 | [1,1] | 3 | 20 | 1.2 | [1,1] |
| 3,10 | 7 | 34 | 1.5 | [1,1] | 5 | 33 | 1.4 | [1,1] |
| 3,12 | 7 | 48 | 1.8 | [1,1] | 5 | 37 | 1.3 | [1,1] |
| 3,14 | 10 | 85 | 1.8 | [0.997590,1] | 5 | 68 | 1.7 | [1,1] |
| 4,6 | 10 | 72 | 1.9 | [1,1] | 5 | 66 | 1.3 | [1,1] |
| 4,8 | 13 | 92 | 3.4 | [1,1] | 6 | 81 | 2.1 | [1,1] |
| 4,10 | 13 | 105 | 35.9 | [0.872164,1] | 6 | 102 | 12.8 | [1,1] |
| 4,12 | 13 | 216 | 53.8 | [0.754773,1] | 6 | 137 | 16.5 | [1,1] |

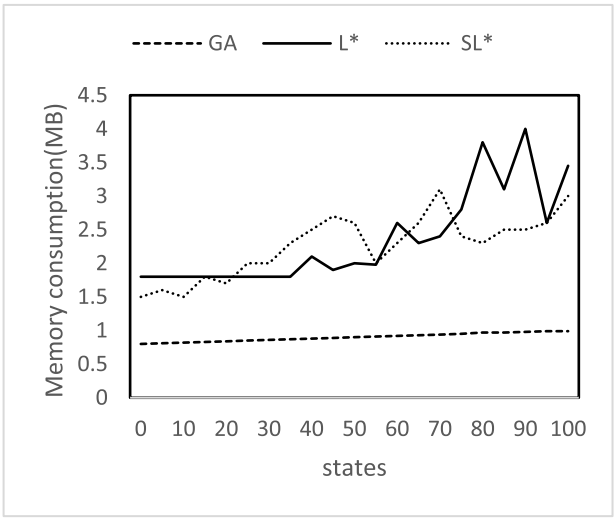



**FIGURE 4.** Memory consumption of GA, L* and SL* with $|\Sigma| = 2$.

makes the advantage of memory consumption more apparent. Moreover, it can generate a smaller assumption, in few, up to almost one order of magnitude (e.g., randomized dining philosophers' problem with N = 9,10 and synchronous leader election with parameters (4,12)). With interface alphabet refinement, PAG-GA framework adds actions in set-theoretic difference $\alpha A\Sigma$ to training set, which means adding more behaviors to the next assumption that genetic algorithm tries to generate.

In order to further illustrate the advantages of genetic algorithm, we generate assumption DFAs with states from 5 to 100 states, randomly. We fix $|\Sigma| = 2$, prefix-closed DFAs are constructed by GA, L* and SL*, respectively. We compare the time and memory consumption of genetic algorithm (GA), L* learning algorithm (L*) and symbolic L* learning algorithm (SL*), which is shown in Fig.3 and Fig.4, respectively.

## VI. CONCLUSION AND FUTURE WORKS

In this paper, we propose a probabilistic assume-guarantee framework based on genetic algorithm. As far as we know, this is the pioneering work in the area of fully-automated assumption generation. It only needs to record the encode of the problem domain and the training set. So, the space complexity is largely reduced. In order to improve efficiency further, we also combine the interface alphabet refinement with our framework orthogonally. It is the first probabilistic assume-guarantee reasoning framework with interface alphabet refinement. Moreover, we apply counterexample expressed by diagnostic submodel to augment the training set, which can reduce the time cost of generating assumption. Our framework can either generate assumption from scratch or refine it from a coarser one. If a coarse assumption is provided at the beginning, whatever it is true or not, the speed of getting the final right assumption is higher than assumption is not provided. Experimental results show that our framework has the potential to handle large systems.

In this paper, assumption generation largely depends on the quality of the training set. In addition to the three cases in Section IV, PAG-GA framework can verify any stochastic systems by providing valid and reliable training set for assumption generation.

In the future, we would like to explore symbolic implementations [32] for our framework in order to increase completeness further. Also, we will extend our framework for verifying PCTL over MDP [33].

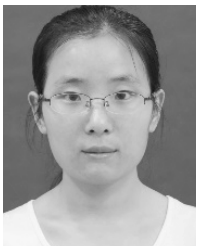
## ACKNOWLEDGMENTS

Thanks to Professor Marta Kwiatkowska at University of Oxford, UK, she has inspired us a lot from her books and papers, especially the direct discussion with her.

## REFERENCES

[1] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: MIT Press, 2008.
[2] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems* (Lecture Notes in Computer Science), vol. 4486, M. Bernardo and J. Hillston, Eds. Berlin, Germany: Springer, 2007, pp. 220–270.
[3] J.-P. Katoen, "The probabilistic model checking landscape," in *Proc. 31st Annu. ACM/IEEE Symp. Logic Comput. Sci. (LICS)*, Jul. 2016, pp. 31–45.
[4] C. Baier, B. R Haverkort, H Hermanns, and J.-P. Katoen, "Performance evaluation and model checking join forces," *Commun. ACM*, vol. 53, no. 9, pp. 76–85, Sep. 2010.
[5] E. M. Clarke, E. A. Emerson, and J. Sifakis, "Model checking: Algorithmic verification and debugging," *Commun. ACM*, vol. 52, no. 11, pp. 74–84, 2009.
[6] A. Pnueli, "In transition from global to modular temporal reasoning about programs," in *Logics and Models of Concurrent Systems* (NATO ASI Series), vol. 13. New York, NY, USA: Springer-Verlag, 1985, pp. 123–144.
[7] C. S. Păsăreanu, D. Giannakopoulou, M. G. Bobaru, J. M. Cobleigh, and H. Barringer, "Learning to divide and conquer: Applying the L∗ algorithm to automate assume-guarantee reasoning," *Formal Methods Syst. Des.*, vol. 32, no. 3, pp. 175–205, Jun. 2008.

[8] L. de Alfaro, T. A. Henzinger, and R. Jhala, "Compositional methods for probabilistic systems," in *Proc. 12th Int. Conf. Concurrency Theory*, Aug. 2001, pp. 351–365.
[9] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu, "Assume-guarantee verification for probabilistic systems," in *Proc. Int. Conf. Tools Algorithms Construct. Anal. Syst.* Berlin, Germany: Springer, 2010, pp. 23–37.
[10] L. Feng, M. Kwiatkowska, and D. Parker, "Compositional verification of probabilistic systems using learning," in *Proc. 7th Int. Conf. Quant. Eval. Syst.*, Sep. 2010, pp. 133–142.
[11] L. Feng, "On learning assumptions for compositional verification of probabilistic systems," Ph.D. dissertation, Univ. Oxford, Oxford, England, Oct. 2013.
[12] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, Aug. 1987.
[13] B. Bollig, P. Habermehl, C. Kern, and M. Leucker, "Angluin-style learning of NFA," in *Proc. 21st Int. Joint Conf. Artif. Intell.*, Jul. 2009, pp. 1004–1009.
[14] T. Han, J.-P. Katoen, and D. Berteun, "Counterexample generation in probabilistic model checking," *IEEE Trans. Softw. Eng.*, vol. 35, no. 2, pp. 241–257, Mar./Apr. 2009.
[15] F. He, X. Gao, M. Wang, B.-W. Wang, and L. Zhang, "Learning weighted assumptions for compositional verification of Markov decision processes," *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 3, Aug. 2016, Art. no. 21.
[16] R. Bouchekir and M. C. Boukala, "Learning-based symbolic assume-guarantee reasoning for Markov decision process by using interval Markov process," *Innov. Syst. Softw. Eng.*, vol. 14, no. 3, pp. 229–244, Sep. 2018.
[17] A. Komuravelli, C. S. Păsăreanu, and E. M. Clarke, "Assume-guarantee abstraction refinement for probabilistic systems," in *Proc. Int. Conf. Comput. Aided Verification*. Berlin, Germany: Springer, 2012, pp. 310–326.
[18] R. Chadha and M. Viswanathan, "A counterexample-guided abstraction-refinement framework for Markov decision processes," *ACM Trans. Comput. Logic*, vol. 12, no. 1, Oct. 2010, Art. no. 1.
[19] T. Berg, B. Jonsson, M. Leucker, and M. Saksena, "Insights to angluin's learning," *Electr. Notes Theor. Comput. Sci.*, vol. 118, pp. 3–18, Feb. 2005.
[20] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Cambridge, MA, USA: MIT Press, 1975.
[21] D. Whitley and T. Starkweather, "GENITOR II: A distributed genetic algorithm," *J. Exp. Theor. Artif. Intell.*, vol. 2, no. 3, pp. 189–214, Jul. 1990.
[22] J. Meena, M. Kumar, and M. Vardhan, "Cost effective genetic algorithm for workflow scheduling in cloud under deadline constraint," *IEEE Access*, vol. 4, pp. 5065–5082, 2016.
[23] M. Gheorghiu, D. Giannakopoulou, and C. S. Păsăreanu, "Refining interface alphabets for compositional verification," in *Proc. Int. Conf. TOOLS Algorithms Construct. Anal. Syst.* Berlin, Germany: Springer-Verlag, 2007, pp. 292–307.
[24] Y. Ma, Z. Cao, and Y. Liu, "Counterexample generation in stochastic model checking based on PSO algorithm with heuristic," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 26, no. 7, pp. 1117–1143, Sep. 2016.
[25] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE Trans. Softw. Eng.*, vol. SE-4, no. 3, pp. 178–187, May 1978.
[26] Z. Lai, S. C. Cheung, and Y. Jiang, "Dynamic model learning using genetic algorithm under adaptive model checking framework," in *Proc. 6th Int. Conf. Qual. Softw.*, Oct. 2006, pp. 410–417.
[27] J. E. Beasley and P. C. Chu, "A genetic algorithm for the set covering problem," *Eur. J. Oper. Res.*, vol. 94, no. 2, pp. 392–404, Oct. 1996.
[28] P. Yang, D. N. Jansen, and L. Zhang, "Distribution-based bisimulation for labelled Markov processes," in *Formal Modeling and Analysis of Timed Systems* (Lecture Notes in Computer Science), vol. 10419, 2017, pp. 170–186.
[29] Y. Liu, J. Sun, and J. S. Dong, "PAT 3: An extensible architecture for building multi-domain model checkers," in *Proc. Int. Symp. Softw. Rel. Eng.*, Nov./Dec. 2011, pp. 190–199.
[30] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd Int. Conf. Comput. Aided Verification*, in Lecture Notes in Computer Science, vol. 6806. Berlin, Germany: Springer, 2011, pp. 585–591.
[31] M. Lahijanian and M. Kwiatkowska, "Specification revision for Markov decision processes with optimal trade-off," in *Proc. IEEE 55th Conf. Decision Control*, Dec. 2016, pp. 7411–7418.
[32] G. Argyros and L. D'Antoni, "The learnability of symbolic automata," in *Proc. 30th Int. Conf. Comput. Aided Verification*, Oxford, U.K., 2018, pp. 427–445.
[33] R. Li and Y. Liu, "Compositional stochastic model checking probabilistic automata via symmetric assume-guarantee rule," in *Proc. 17th IEEE/ACIS Int. Conf. Softw. Eng. Res.*, Honolulu, Hawaii, May 2019, pp. 29–31.



**YAN MA** was born in Zaozhuang, Shangdong, China, in 1981. She received the B.S. degree in computer science and technology from Jiangnan University, in 2007. She is currently pursuing the Ph.D. degree with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics. She is also a Researcher with the School of Computing, National University of Singapore. She is a Student Member of ACM and CCF. Her research interests include formal methods, software engineering, and optimization algorithm.

**ZINING CAO** received the B.S., M.S., and Ph.D. degrees from Tsinghua University, in 1995, 1998, and 2001, respectively. He is currently a Professor with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics. He has published more than 30 papers. His current research interests include formal methods and Logic in computer science.

**YANG LIU** received the Ph.D. degree in computer science and technology from Shanghai University, in 2012. He is currently a Professor with the College of Information Engineering, Nanjing University of Finance and Economics, and also a Senior Researcher with the School of Computing, National University of Singapore. He has published about 40 papers in the famous conferences or journals. His research interests include software engineering, formal verification, and blockchain. He is a member of CCF and ACM.