Pira E. Using Markov chain based estimation of distribution algorithm for model-based safety analysis of graph transformation. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 36(4): 839–855 July 2021. DOI 10.1007/s11390-020-1003-3

Using Markov Chain Based Estimation of Distribution Algorithm for Model-Based Safety Analysis of Graph Transformation

Einollah Pira

Faculty of Information Technology and Computer Engineering, Azarbaijan Shahid Madani University Tabriz 5375171379, Iran

E-mail: pira@azaruniv.ac.ir

Received October 6, 2019; accepted October 11, 2020.

Abstract The ability to assess the reliability of safety-critical systems is one of the most crucial requirements in the design of modern safety-critical systems where even a minor failure can result in loss of life or irreparable damage to the environment. Model checking is an automatic technique that verifies or refutes system properties by exploring all reachable states (state space) of a model. In large and complex systems, it is probable that the state space explosion problem occurs. In exploring the state space of systems modeled by graph transformations, the rule applied on the current state specifies the rule that can perform on the next state. In other words, the allowed rule on the current state depends only on the applied rule on the previous state, not the ones on earlier states. This fact motivates us to use a Markov chain (MC) to capture this type of dependencies and applies the Estimation of Distribution Algorithm (EDA) to improve the quality of the MC. EDA is an evolutionary algorithm directing the search for the optimal solution by learning and sampling probabilistic models through the best individuals of a population at each generation. To show the effectiveness of the proposed approach, we implement it in GROOVE, an open source toolset for designing and model checking graph transformation systems. Experimental results confirm that the proposed approach has a high speed and accuracy in comparison with the existing meta-heuristic and evolutionary techniques in safety analysis of systems specified formally through graph transformations.

Keywords safety analysis, model checking, Markov chain, estimation of distribution algorithm, graph transformation system

1 Introduction

Modern safety-critical systems such as nuclear power plants or spacecraft controllers are very sensitive to failures because the existence of a failure (even very minor) in these systems can result in loss of life or irreparable damage to the environment [1]. With the increasing scale and complexity of these systems, traditional verification methods such as testing and simulation cannot have the high efficiency in assessing reliability because these methods can analyze only a limited number of system behaviors due to time limitations. The aim of testing is to examine the correctness of the implemented system using a collection of test cases, whereas the simulation method employs models to capture the system behaviors and simulates different scenarios on these models to verify the correctness of the system. Recently, model-based safety analysis has been introduced for designing models of systems and automatically assessing the safety properties in them. A well-known technique in such analysis is model checking by which system properties are verified (or refuted) through exploring all reachable states (state space) of the model [2]. However, in large and complex systems, model checking confronts with the state space explosion problem in which all reachable states cannot be generated due to exponential memory usage. Hence, the techniques employed in model checking should explore only a portion of the state space. Using evolutionary algorithms (EAs) such as Genetic Algorithm (GA) [3], Ant Colony Optimization (ACO) [4, 5], Particle Swarm Optimization (PSO) [6] and Bayesian Optimization Algorithm (BOA) [7] is one of these techniques. As another technique, applying knowledge discovery meth-

Regular Paper
©Institute of Computing Technology, Chinese Academy of Sciences 2021

ods such as data mining [8,9] and learning a Bayesian network (BN) [9] can be mentioned.

A good/bad event that always/never must happen in a system is called a safety property. In large and complex systems, a safety property q must be refuted (instead of verification) by finding a state in the state space in which q is not satisfied, i.e., the reachability property "not q" is satisfied. A reachability property p implies that there is a state in the state space in which p is satisfied [10]. In some cases, finding such a state may cause the entire exploration of the state space. When the state space of systems modeled by graph transformations is explored, the allowed rule on the current state depends only on the applied rule on the previous state, not the one on earlier states. This fact motivates us to use a Markov chain (MC) to capture this type of dependencies and also apply the Estimation of Distribution Algorithm (EDA) to improve the quality of the MC, i.e., some of sampled solutions approach to an optimum solution [11]. EDA belongs to the class of population-based optimization algorithms in which a graphical probabilistic model is learned from a set of promising solutions, and then the model is employed to produce new solutions [12]. The main contributions of the paper are as follows.

1) The reason of using Markov chain to capture the dependencies between applied rules on the current and previous states is described.

2) The Markov chain based Estimation of Distribution Algorithm (EDA) is used to verify reachability properties and refute safety properties.

3) A practical analysis is performed using well-known benchmarks, and the proposed approach is compared with the existing meta-heuristic and evolutionary techniques.

As a motivation example, an emergency diesel generator control system can be mentioned by which a reserve emergency power is provided for nuclear power plants [13]. In other words, emergency diesel systems are used in safety systems that they should continuously connect to a power supply. For more information about modeling this system, interested readers can refer to [13].

In the model-based safety analysis, the considered system should be modeled by a modeling language such as Graph Transformation System (GTS) [14]. To implement the proposed approach, we use GROOVE [15] as a test bed to implement the proposed approach because it does automatic verification by generating the model's state space.

In the rest of the paper, we survey the related work in Section 2. Section 3 describes the required background such as Markov chain, Markov chain based EDA, GTS formalism, and model checking. In Section 4, the proposed approach is described in detail. Section 5 presents detailed experimental results. In Section 6, we discuss the advantages and limitations of the proposed approach. Finally, in Section 7, we conclude the paper with some directions for possible future work.

2 Related Work

In the literature, various approaches have been proposed to tackle the state space explosion problem in the model-based safety analysis. These approaches, which have the most relation to our approach, can be divided into two categories: those that use EDA and those that are applied to systems modeled by GTS.

2.1 Approaches Using Estimation of Distribution Algorithm (EDA)

The approach in [16] is applied to find common concurrent errors such as deadlocks in multithreaded software. This EDA-based approach uses n-gram distributions as a probabilistic model for learning through the best individuals and sampling the new individuals. To evaluate the performance of the approach, it is implemented using the Java PathFinder (JPF) model checker and the ECJ toolkit. In [17], the authors proposed an EDA-based approach to find deadlocks in multithreaded Java programs. They extended the work in [16] to find counterexamples in Promela models. The EDA-based work in [18] extracts the information from an earlier execution of EDA and reuses this information to aid the search in a future execution. The proposed approaches in [7, 19] employ a Bayesian network based EDA, also called BOA, to verify reachability properties and refute safety properties. For simplicity, the authors [7, 19] supposed that the structure of BNs is fixed. Based on this assumption, they proposed three different versions of the approach with different structures. The first version, which is called BOAcl2 (BOA with a chain of two nodes), uses a chain structure with two nodes. Whereas, in the second version, which is called BOAcln (BOA with a chain of n nodes), the used BNs have the structure of a chain with n nodes such that each node depends conditionally on the previous node. Also, the third version, which is called BOActp (BOA with a chain structure and two parent

nodes), employs the BNs that their structures are similar to the ones in cBOA with a difference that each node in BOActp depends conditionally on two previous nodes.

2.2 Approaches Applied to Systems Modeled by GTS

The second category includes approaches whose important feature is that they are applied to systems modeled by GTS. To refute safety properties in systems modeled by GTS, the authors proposed an approach using data mining for efficient exploring of the state space [8]. This approach, called EMCDM, obtains special knowledge from the information of checking some smaller models to check the complex and large models. In spite of having the advantages such as high speed and generating shorter counterexamples in comparison with others, EMCDM fails in systems with infinite state space, because it cannot explore the state space of smaller models exhaustively to obtain the required knowledge. To handle the shortage of EMCDM, the authors proposed two approaches, called LDM (Learning by Data Mining) and LBN (Learning a BN), by which only a slight portion of the state space is explored rather than the complete exploration of some smaller models [9]. LDM uses a modified version of the Apriori algorithm to find a frequent pattern through the partially explored state space. Whereas, LBN learns a BN through the explored state space. After obtaining the required knowledge, LDM and LBN use it to explore the remainder of the model's state space intelligently until a goal state is found.

In [3], the authors proposed a GA-based solution to search safety violations in systems modeled by GTS. The approach in [6] employs the PSO algorithm to refute safety properties in systems modeled by GTS. In [4], the authors proposed an ACO-based approach to detect safety violations in concurrent systems modeled by calculus of communicating systems (CCS). Other ACO-based approaches were also proposed to refute safety properties described in the linear temporal logic [5]. In [20], the authors proposed a hybrid approach based on BAT and PSO algorithms (also called BAPSO) to refute complex systems specified through GTS. The Bat algorithm (BA) is a meta-heuristic algorithm for global optimization and inspired from echolocation behavior of bats to search for food and distinguish prey from barriers [21].

3 Background

3.1 Markov Chain

The Markov property expresses that the occurrence probability of the next event on the system (and in fact all future events) depends only on the current event, not the previous events [11]. A Markov chain (MC) is a stochastic model that satisfies the Markov property. In simpler terms, an MC is a stochastic model to describe a sequence of possible events such that the occurrence probability of future events depends only on the present event, not the previous events [11]. Assuming that X denotes random variables and $S(x)$ defines the possible assignments (states) of variables, an MC is specified through $S(x)$ and a model M that assigns every state $x \in S(x)$ a next state $x' \in S(x)$. Actually, the transition model M determines the probability $M(x \to x')$ of going from x to x' for each pair of states x and x'. Moreover, if the probability $M(x \to x')$ is zero, the transition $x \to x'$ is not shown in the model. Fig.1 displays an example of a simple MC. As seen in Fig.1, $S(x)$ and M are the sets of $\{x_0, x_1, x_2\}$ and $\{x_0 \to x_1$: 0.75, $x_0 \to x_2$: 0.25, $x_1 \to x_0$: 0.45, $x_1 \to x_2$: 0.55, $x_2 \to x_0$: 1.00$\}$ respectively. Random variables (and their possible states) and the transition probabilities of the model may be induced by a domain expert or learned from the dataset using machine learning algorithms. There are different methods for learning the transitions probabilities of an MC, for example, relative frequencies observed in the dataset based on the maximum likelihood hypothesis can be employed.

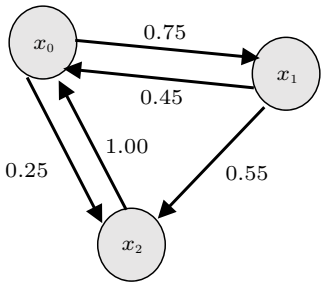


Fig.1. A simple Markov chain.

3.2 Markov Chain Based Estimation of Distribution Algorithm

Estimation of distribution algorithm (EDA) belongs to evolutionary algorithms and is based on the estimation of distributions. These algorithms use the distribution of promising solutions instead of traditional genetic

operators like crossover and mutation. This distribution, which can be considered as a probabilistic graphical model, captures dependencies/independencies between variables of the problem. The estimated distribution is then employed to generate new candidate solutions. EDA processes as the following. The initial population of solutions is generated randomly. After evaluating the current population by a fitness function, the promising solutions are selected and a distribution is estimated using them. Finally, EDA samples the estimated distribution to produce new candidate solutions and adds them into the old population by replacing some of the old ones. These steps are repeated until the termination criteria are met [12].

In Markov Chain based Estimation of Distribution Algorithm (MCEDA), a Markov chain is used to estimate the distribution using the selected solutions in each step [11]. As mentioned in Subsection 3.1, to estimate the distribution in the MC, we can use the relative frequencies observed in the dataset based on the maximum likelihood hypothesis. Moreover, the Markov Chain Monte Carlo (MCMC) sampling method is used to sample the MC. MCMC is a strategy to generate a path from the distribution.

3.3 Graph Transformation System

Graph Transformation System (GTS) is a graph-based formalism in which graphs and graph transformations are used to formally describe the structural and behavioral aspects of systems respectively [14]. An attributed GTS contains three following components. 1) A type graph TG which specifies all node types and edge types. Furthermore, TG has two functions that determine the source/destination nodes of an edge. 2) A host graph HG which determines the initial configuration of a system that should be instantiated from the type graph TG. 3) A set R of transformation rules by which all possible configurations (also called state space) of a system are generated. Each transformation rule includes pre-conditions, post-conditions, and (probably) a negative application condition (NAC). Pre-conditions (also called left-hand side or briefly LHS) of a rule specify the conditions that the current host graph should have before applying the rule. Whereas, post-conditions (also called right-hand side or briefly RHS) of a rule determine the conditions that the current host graph should have after applying the rule. NAC specifies the conditions that should not occur in the host graph before applying the rule (similar to the pre-conditions).

In this paper, we use GROOVE, which is an open source toolset and automatic verifier (model checker), to implement our approach and analyze systems specified through GTS. To clarify the functionality of GROOVE, the readers-writers problem is considered in which several processes (e.g., readers/writers) attempt to access (e.g., read/write) the shared resource (e.g., a book) simultaneously. The shared resource can be read by two or more readers provided that no writer is writing. Moreover, only one writer can write to the shared resources at a time. In Fig.2, the designed model of this problem with four readers (denoted by R) and four writers (denoted by W) is displayed. Fig.2(a) displays the type graph TG of the model. As in Fig.2(a), this

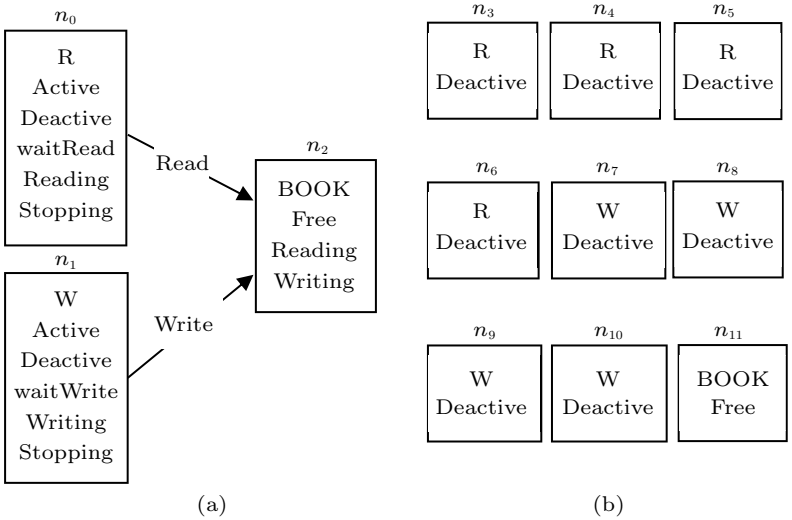


Fig.2. (a) Type and (b) host graphs of a model of the readers-writers problem modeled by GTS in the GROOVE toolset.

graph has three node types n_0, n_1, and n_2 and two edge types n_0n_2 and n_1n_2. The type of n_0/n_1 is R/W and it (i.e., a reader/writer) can get one of the labels active, deactive, waitRead/waitWrite, etc.. Also, the type of n_2 is BOOK and it can get one of the labels free, reading and writing.

Fig.2(b) shows the host graph HG of the model. HG contains the set $\{n_3, n_4, \cdots, n_{11}\}$ of nodes and it does not have any edges. As mentioned in the above, HG should be an instance of the type graph, i.e., Fig.2(a). In other words, each node/edge in Fig.2(b) must be an instance of a(n) node/edge type in Fig.2(a). For example, node n_{11} in Fig.2(b) is one instance of the node type n_2 in Fig.2(a).

In GROOVE, the graphs of LHS, RHS, and NAC are merged together and they are distinguished by colour coding. The commonalities of LHS and RHS graphs are colored in black. The nodes and edges of the LHS/RHS graph which are removed/created after rule application are specified in blue/green. Furthermore, the NAC graph is specified by bold red double-bordered nodes and dashed edges.

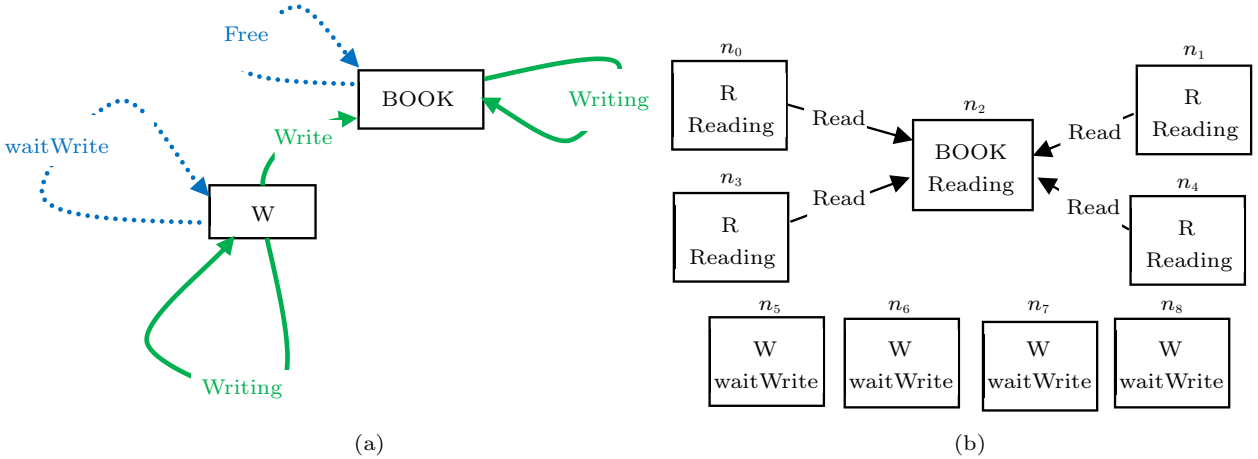
A transformation rule r is applicable on state s, which is actually a graph, when there is at least one image of LHS on state s such that this occurrence should not contain any subgraph of NAC. If such occurrence is found, it is replaced by the RHS graph. For example, we consider the transformation rule of goWrite in the readers-writers problem. As seen in Fig.3(a), this rule is applicable when there are a writer with label waitWrite and the book with label free. If such conditions are established, this rule will change the labels of the writer and the book to writing. Moreover, a new edge is created between these nodes.

State properties specify some special configurations that may happen in a system. A state property is defined by a transformation rule such that its LHS and RHS graphs are equal. Hence, applying such a rule does not change the structure of any state. For example, a state property of "all readers are reading and all writers in the waitWrite mode" for the readers-writers problem with four readers and four writers is illustrated in Fig.3(b). It is noteworthy that since all nodes and edges of this state property (i.e., Fig.3(b)) belong to the commonalities of the LHS and RHS graphs, they are colored in black.

In GTS, some of transformation rules can be applicable on the current state (so-called allowed rules). After applying one of these allowed rules (e.g., r_1) on the current state, another rule (e.g., r_2) may be allowed. In fact, rule r_1 is a trigger for rule r_2 or r_2 depends on r_1. For example, in the readers-writers problem, the rule of goWrite can be applied when the status of a writer is changed into waitWrite using the rule of requsetWrite. It means that the rule of goWrite depends on that of requsetWrite, not the previous rules. Due to the definition of the Markov property, we can conclude that the problem of dependencies between applied and allowed rules on the states has the Markov property.

3.4 Model Checking

Model checking is a well-known and automatic technique by which system properties are verified (or refuted). For this purpose, it produces all possible states of the model in the format of a graph whose nodes and edges are states and transitions (i.e., applied rules on the states) between them respectively [2]. To use



(a) (b)

Fig.3. Example of (a) a transformation rule and (b) a state property of a model of the readers-writers problem modeled by GTS in the GROOVE toolset.

model checking, a system property must be described by a temporal logic such as LTL (linear temporal logic) and CTL (computation tree logic). In LTL, infinite sequences of states are considered such that each state at any point in time has a unique successor, whereas in CTL, each state can have several successors [22]. Since CTL is linear in both the size of the state space and the size of the formula, in this paper, we have chosen CTL to describe system properties. If a given property is verified (or refuted), a witness (or counterexample) will be generated. Witnesses/counterexamples are some special behaviors of the system that they can be used by designers for analyzing the design. In the model's state space, witnesses/counterexamples are paths that begin from the initial state and end in the state (also called goal state) in which the property is satisfied/violated.

Reachability and safety are common properties examined by the model checking technique. A reachability property is a special configuration that occurs in at least one state of the system. Assuming that g is a state property, g is a reachability property if there is at least one state in the state space in which g is satisfied. If a reachability property is verified, a witness will be generated. For expressing the reachability property g in CTL, the formula $E <> g$ is used (and also the formula $EF\ g$ in GROOVE). Safety properties are special configurations that happen in all states of the system. In other words, g is a safety property if it is satisfied in all states of the system. To express the safety property g in CTL, the formula $A[]g$ is used (and also the formula $AG\ g$ in GROOVE). In large and complex systems, verifying a safety property g is impossible due to occurring the state space explosion problem. In this case, the safety property g is refuted rather than verified. For this purpose, we should search a state in the state space in which g is not satisfied, i.e., the reachability property "not g" is satisfied.

4 Proposed Approach

In this section, we propose an approach, which is called SAMEDA, using Markov chain based EDA (MCEDA) to analyze safety properties in systems specified formally through graph transformations. In the state space of such systems, allowed rules on the current state only depend on applied rules on the previous state, not the ones on earlier states. To capture these dependencies between rules, a Markov chain can be used. In the following of this section, the structure and encoding of chromosomes are explained in Subsection 4.1. In Subsection 4.2, we propose fitness functions. In Subsection 4.3, the structure and the estimation of a Markov chain are explained. In Subsection 4.4, we talk about sampling the estimated MC. The detail of SAMEDA is described in Subsection 4.5.

4.1 Structure and Encoding Chromosomes

As mentioned before, if the proposed approach can find a goal state, the output will be a witness/counterexample, a path with specific length starting from an initial state and ending in the goal state. Consequently, each sequence of applied rules in a path of the state space, where the path has a specific length and starts from an initial state, is considered as a chromosome. In this paper, to encode a chromosome, the index of outgoing transitions (applied rules) in the corresponding path is used. It is obvious that the value of each gene is between 0 and the maximum number of possible outgoing transitions of a state. In the following, an index sequence of outgoing transitions along with a path of applied rules is considered as a chromosome. For example, suppose that the set R of rules for a dummy system is $\{r_0, r_1, r_2, r_3\}$. Fig.4 displays the chromosome "1021" along with the path "$r_1r_2r_0r_1$" of applied rules in the state space. As seen in Fig.4, the chromosome is highlighted with the colored states and transitions.

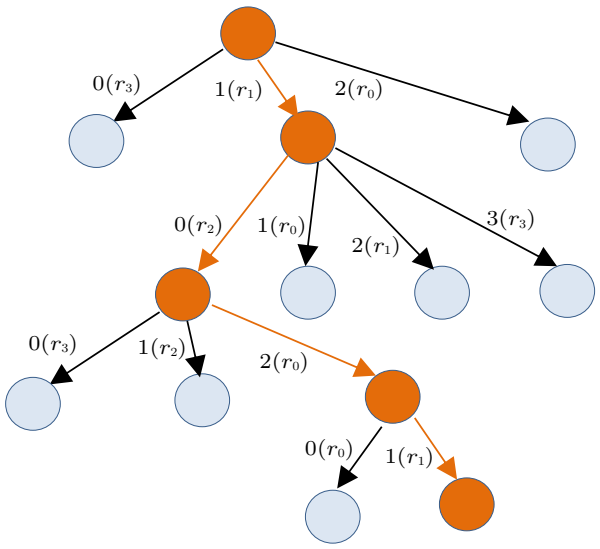


Fig.4. Chromosome "1021" along with path "$r_1r_2r_0r_1$".

4.2 Fitness Functions

A fitness function used in refuting the safety property "there isn't any deadlock state in the system"

should guide the chromosomes in the direction to reach a deadlock state. According to [3], a chromosome is more promising if the outgoing transitions of states in the corresponding path decrease. It means that the sum of the number of outgoing transitions of states in the path can be considered as a fitness value. Also, a fitness function employed in refuting the safety property g should guide the chromosomes in the direction to reach a state in which the reachability property "not g" is satisfied. As mentioned before, we have modeled the systems by GTS; hence reachability properties and states in the state space are graphs. If the last state of a path is similar to a given reachability property, the probability of reaching the path to a goal state will be high, i.e., the corresponding chromosome is more promising than the others. Therefore, the similarity value between the corresponding graphs (named G_t and G_h respectively) of a given reachability property t and the last state s of the path of the chromosome can be considered as a fitness value. To display a label *lbe* for node n, GROOVE uses a self-loop edge labeled by *lbe* on node n. For example, Fig.5 illustrates the concrete graph related to the state property of the model of the readers-writers problem in GROOVE, i.e., Fig.3(b). In the concrete graph, all labels of nodes are shown by self-loop edges. According to Fig.5, self-loop edges of a node can be considered as output edges. Hence, the fitness function considers all labels of outgoing edges of node n as a set of labels of that node (or set of *labels-n*). For example, in Fig.5, the set of labels of n_0 (i.e., *labels-*n_0) is {R, reading}. The fitness function computes the intersection of two sets *labels-*t_n and *labels-*h_n for any pairs ($t_n \in G_t$, $h_n \in G_h$) of nodes and considers its size as the similarity value between t_n and h_n. This function then finds node pairs with the largest similarity value (called semi-similar nodes). After finding all semi-similar nodes, the function sums their similarity values and holds the result in the *EQU_Count* variable. If G_t has NAC, the function computes the summation of similarity values for semi-similar nodes of the NAC and G_h graphs, and saves the result in the *NegEQU_Count* variable. It is obvious that whatever *NegEQU_Count* is smaller and *EQU_Count* is larger, the given chromosome has more chance to reach a goal state. Therefore, we consider the difference between *NegEQU_Count* and *EQU_Count* values as a fitness value.

4.3 Structure and Estimation of a Markov Chain

After evaluating the current population by a fitness function, a set of chromosomes is selected using the truncation selection method [23] to estimate an MC. The functionality of the method is that it sorts the chromosomes according to their fitness values and then selects the best chromosomes using a truncation threshold (also called *estimating rate*). *Estimating rate* specifies the ratio of the population to be selected for estimating an MC. Before the estimation of an MC, we should determine its random variables, possible assignments (states) of variables and a transition model M. In this paper, R is considered as a ran-

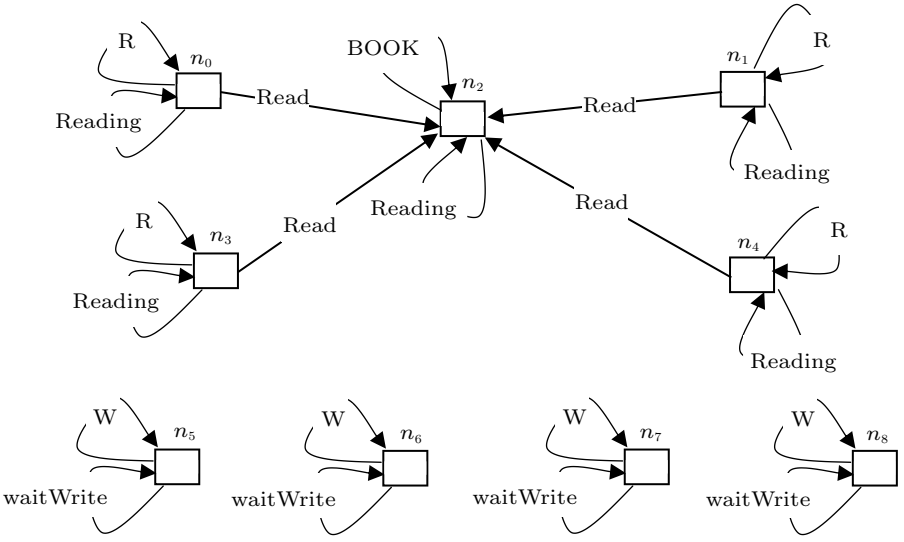


Fig.5. Concrete graph related to the state property of the model of the readers-writers problem in GROOVE, i.e., Fig.3(b).

dom variable and all transformation rules are considered as possible assignments. Assume that the paths encoded by the promising chromosomes are the form "$c_0c_1\cdots c_{n-2}c_{n-1}$", where n is the length of chromosomes and $c_i \in \{r_0, r_1, \cdots, r_{|R|-1}\}$ $(0 \leqslant i \leqslant n-1)$. The probability $M\ (r_i \to r_j)$ of going from r_i to r_j for each pair of r_i and $r_j \in \{r_0, r_1, \cdots, r_{|R|-1}\}$ $(0 \leqslant i \leqslant n-1$ and $0 \leqslant j \leqslant n-1)$ specifies the ratio of the occurrence number of rules r_i and r_j to the total occurrence number of rule r_i. Formally, the probability $M\ (r_i \to r_j)$ is computed by (1):

$$M(r_i \to r_j) = \frac{\sum_{k=1}^{n-1} \#(c_{k-1}c_k = r_i r_j)}{\sum_{k=1}^{n-1} \#(c_{k-1} = r_i)},$$
$$0 \leqslant i \leqslant |R|-1 \quad \text{and} \quad 0 \leqslant j \leqslant |R|-1, \tag{1}$$

where "#" means "the number of" in all paths encoded by the promising chromosomes. Fig.6 displays the structure of an MC for the proposed approach. As seen in Fig.6, the MC includes a transition model M determining the probability $M\ (r_i \to r_j)$ of going from r_i to r_j for each pair of rules r_i and r_j.

To estimate an MC, Algorithm 1 can be used. This algorithm, according to lines 1–5, firstly initializes the MC as the followings: for each rule in *Rules* (a set of transformation rules), it considers a node (as a structure) *node* containing the rule and two lists of *nextRules* and *nextProbs*. *nextRules* and *nextProbs* hold the rule and the probability of other nodes that are dependent on the current node respectively. In line 6, all chromosomes are sorted based on their fitness values to select the promising chromosomes proportional by *estimating rate* (line 7). In lines 13–16, for each node *node*, the algorithm finds the repetition number of *node.rule* in promising chromosomes and saves it in the *count* variable. Moreover, the same process is done for the sequence of *node.rule* and the i-th rule of *Rules* and saves the result in *next_count*. In lines 17–20, if the both of *next* and *next_count* are positive, the i-th rule of *Rules* and the probability *next*/*next_count* are added to *node.nextRules* and *node.nextProbs* respectively.

4.4 Sampling the Estimated MC

After estimating an MC, it is sampled to generate new chromosomes in order to replace some of the worst ones in the old population. In this paper, the parame-

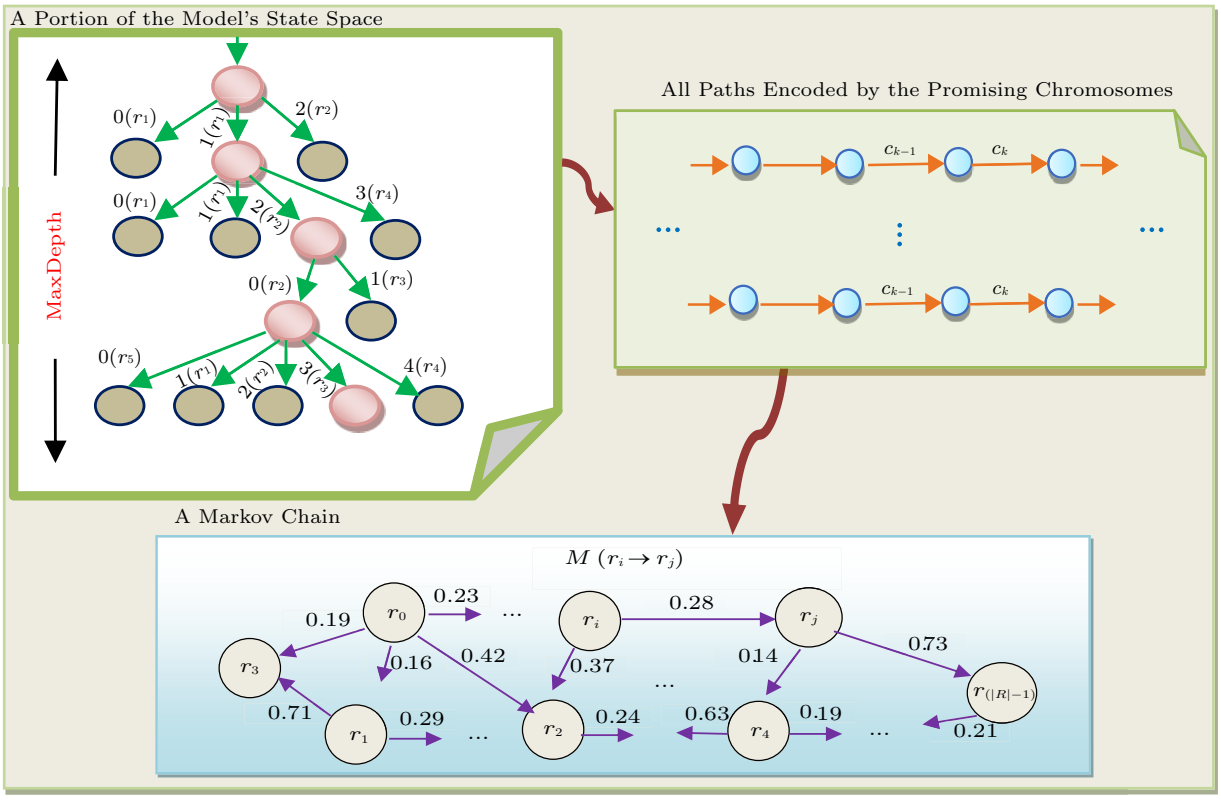


Fig.6. Markov chain of the proposed approach.

ter of *sampling rate* determines the proportion of chromosomes to be sampled and replaced. As mentioned before, the Markov Chain Monte Carlo (MCMC) sampling method is used to sample an MC. Algorithm 2 displays the pseudocode of adapting the MCMC sampling method to model checking. In each explored state of the sampled chromosome, the given property is checked. If a goal state is detected, the algorithm declares the path encoded by the corresponding chromosome as a witness/counterexample and the process will be stopped. Otherwise, the algorithm will be continued to generate other chromosomes.

Algorithm 1. Estimation of a Markov Chain

Input: *Chromosomes*: all chromosomes of current generation; *Rules*: all transformation rules of the considered model, *estimating rate, Depth*;
Output: M: a transition model of a Markov chain;

```
1:  for k = 1 to Rules.size() do
2:      Node node = new Node();
3:      node.rule = Rules.get(k); node.nextRules = null; node.nextProbs = 0.0;
4:      M.nodes.add(node);
5:  end for
6:  Sort all chromosomes of Chromosomes based on their fitness;
7:  NumEstimate = (int) Chromosomes.size()*estimating rate;
8:  for k = 1 to M.nodes.size() do
9:      Node node = M.nodes.get(k);
10:     int count = 0;
11:     for i = 1 to Rules.size() do
12:       int next_count = 0;
13:       for j = 1 to NumEstimate do
14:         Find the repetition number of node.rule in Chromosomes.get(j).paths and save it in count;
15:         Find the repetition number of sequence node.rule and Rules.get(i) in Chromosomes.get(j).paths and save it in
            next_count;
16:       end for
17:       if count > 0 and next_count > 0 then
18:         node.nextRules.Add(Rules.get(i));
19:         node.nextProbs.Add(next_count/count);
20:       end if
21:       M.nodes.set(k, node);
22:       end for
23: end for
24: return M;
```

Algorithm 2. Adapting the MCMC Sampling Method to Model Checking

Input: M: a transition model of a Markov chain, $Population, samplingrate, Depth$;
Output: new sampled chromosomes

```
1:  NumSample = (int) Population * samplingrate;
2:  List <Chromosome> allChromosomes = new List <Chromosome> ();
3:  List <Path> encodedPaths = new List <Path> ();
4:  for k = 1 to NumSample do
5:      Chromosome chromosome = new Chromosome (); Path enpath = new Path ();
6:      GraphState IS = an initial state of the specified model;
7:      Rule curRule = null, nexRule = null;
8:      for i = 1 to Depth do
9:          enpath.Add (IS);
10:         Find all applicable rules over IS state and save them in appRules;
11:         if a goal state is found then
12:           Display "a goal state is found" and enpath as a counterexample;
13:           return;
14:         end if
15:         if curRule is null then
16:           nexRule = choose a rule r from appRules randomly;
17:         else
18:           nexRule = choose a rule r from appRules by which the value of M(curRule → nextRule) is maximized;
19:         end if
20:         chromosome.Add (the index of nexRule in appRules);
21:         Apply nexRule over IS state and update the IS state;
22:         enpath.Add (nexRule); curRule = nexRule;
23:     end for
24:     allChromosomes.Add (chromosome); encodedPaths.Add (enpath);
25: end for
26: return allChromosomes, encodedPaths;
```

4.5 SAMEDA Approach

SAMEDA uses an MC to capture the dependencies between the applied rule on the previous state and the allowed rules on the current state. Moreover, it employs EDA to improve the quality of the MC in hoping to some of sampled solutions approach to an optimum solution. For this purpose, SAMEDA evaluates the current population using the fitness function proposed in Subsection 4.2. SAMEDA then estimates an MC using the selected promising chromosomes. Finally, the estimated MC is sampled to generate new candidate chromosomes in order to replace some of the worst ones in the old population. SAMEDA repeats these steps until a termination criterion such as finding a path leading to a goal state or reaching to a maximum number of generations occurs. In this paper, we consider safety properties in particular and general cases. In the particular case, each safety property g is specific to only one system and it is meaningless for other systems. As mentioned in the above, refuting this type of safety properties is converted into verification of reachability properties. In the general case, a safety property "there isn't any deadlock state in the system" is meaningful for any system and its refutation is performed by detecting a deadlock state (a state with no subsequent state) in the state space. Fig.7 illustrates the architecture of the proposed approach. Moreover, we have presented a comprehensive example to describe the operation of the SAMEDA approach which can be accessed through the web①.

5 Experimental Results

To evaluate the performance of the SAMEDA approach, we implement it by the Java language in the GROOVE toolset. Also, we compare its efficiency with the existing state-of-the-art approaches such as metaheuristic and evolutionary algorithms like BOAcl2 [7, 19], LBN and LDM [9], EMCDM [8], GA [20], PSO [6], and

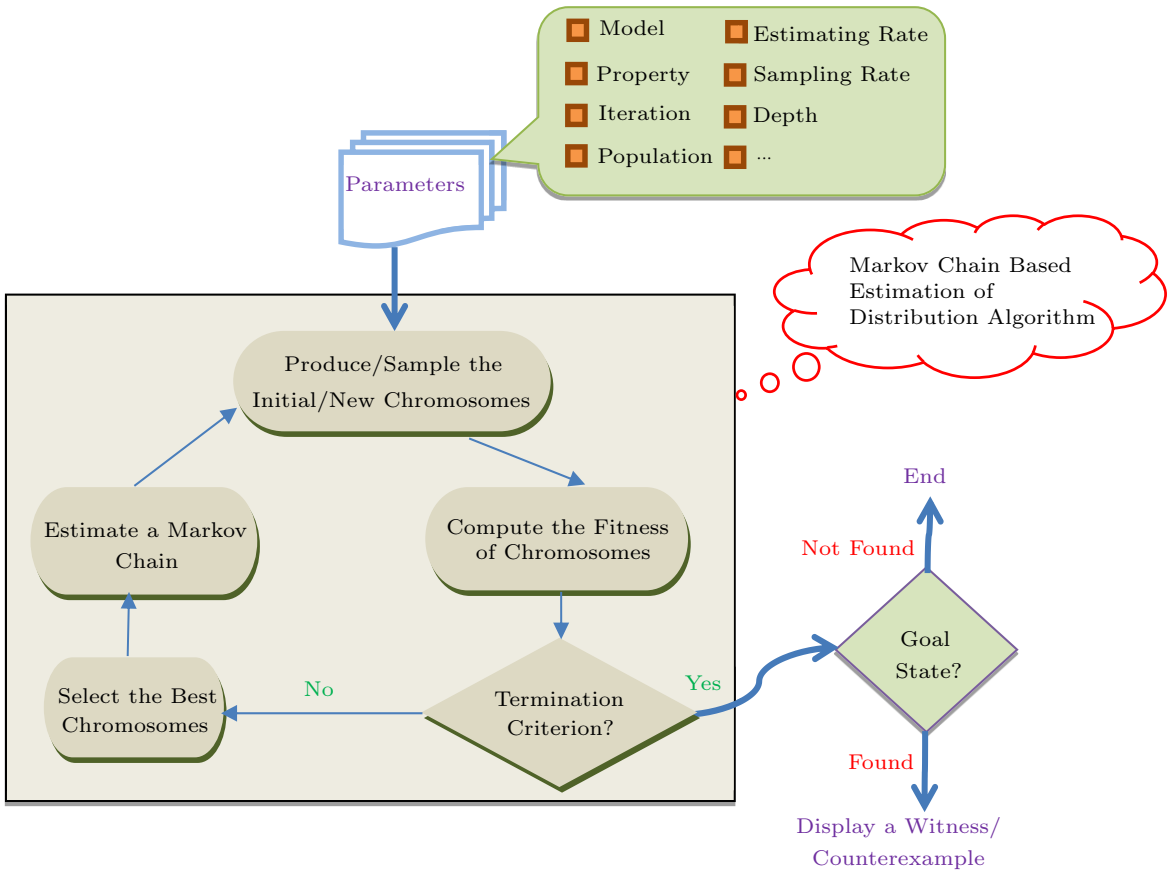


Fig.7. General architecture of the proposed approach.

①https://sourceforge.net/projects/groove-and-mceda/files/example_SAMEDA.pdf/, May 2021.

heuristic search algorithms like BS [24] and IDA* [25]. Since BOAcl2 has a better performance in comparison with BOAcln and BOActp, it is only selected to comparison. To make a fair comparison, we consider the same fitness function (mentioned in Subsection 4.2) for all approaches and also implement them in the GROOVE toolset. The benchmarks used in this paper are different models of the dining philosophers [26], process life cycle [9], readers-writers [9], firewalls [27], and railway switching control [28]. All results have been generated on a system with an Intel CORE i5 processor and 3 GB of memory.

To execute the approaches, we should specify the appropriate values for their parameters. For this purpose, we have performed particular experiments. Table 1 displays these parameters along with their suitable values. The *beam width* parameter specifies the maximum number of states with the highest heuristic values stored at each depth. In some benchmarks such as firewalls, BS cannot find a goal state for any value of the *beam width* parameter and thus no value is displayed for such benchmarks. Moreover, the *iteration* parameter determines the maximum number of allowed generations of SAMEDA. These parameters usually are the same for all benchmarks, whereas some parameters such as *maxDepth* and *population size* (denoted by *popuSize*) are variable for different models of benchmarks. The *maxDepth* parameter determines the maximum deep of the state space that the approaches can explore to search a goal state.

The result tables show the average running time of 20 independent runs of all approaches to refute the safety properties. If EMCDM does not support a model, the term of "NS" is used. Moreover, if an approach uses up all available memory or reaches the maximum number of iterations, but still unable to find any goal state, we will use the term "NF" in the corresponding cell of the result tables. It is possible that by increasing the memory, iterations or *maxDepth*, a goal state may be found. Also, in some benchmarks, there are several goal states in the state space and finding one of them can be simple. Hence, in these benchmarks, it is not necessary to use the large values for the *maxDepth* and *popuSize* parameters. Conversely, in some benchmarks that only one goal state exists, the approaches should explore the deeper level of the state space of large models. It means that the values of these parameters in large models should be increased. Since each pair of LBN and LDM, GA and PSO, and BS and IDA* has the almost same average running time, we display only one of them in the following result tables. The results of other approaches can also be accessed through the web②.

In addition to the result tables, we will consider some charts illustrating the impact of *estimating* and *sampling rates* on average running time of the SAMEDA approach. These charts can be accessed through the web③.

5.1 Dining Philosophers Problem

This problem, used to examine the concurrent algorithms, describes the situation of some philosophers

Table 1. Initial Parameters of the Approaches

Approach	Benchmark	Parameter	Value
SAMEDA	–	*iteration*	100.0
		estimating rate	0.4
		sampling rate	0.6
BOAcl2	–	*learning rate*	0.4
		replacement rate	0.6
BS	dining philosophers, process life cycle, railway switching control readers-writers	*beam width*	10.0, 40.0
LBN and LDM	–	*numPromis*	3.0
	dining philosophers	*minst*	0.6, 0.1
	readers-writers, process life cycle, railway switching control		
PSO	–	$C1$	2.0
		$C2$	2.0
		W	0.8
GA	–	*crossover rate*	0.6
		mutation rate	0.3
		position of crossover	Middle of the chromosomes

②https://sourceforge.net/projects/groove-and-mceda/files/extra_results.pdf/, May 2021.

③https://sourceforge.net/projects/groove-and-mceda/files/impact_runningtime.pdf, May 2021.

sitting around a table with a fork between each two philosophers. At first, all philosophers are in the *thinking* mode. After a while, they may change their modes and go to the *hungry* one (i.e., the rule of *go_hungry*). Each hungry philosopher can get the left fork and go to the *hasLeft* mode if it is free, i.e., it is not held by an adjacent philosopher (i.e., the rule of *get-left*). If a right fork of a philosopher with the *hasLeft* mode is free, he/she gets the fork and goes to the *eating* mode (i.e., the rule of *get-right*). Later, a philosopher with the *eating* mode releases the left (i.e., the rule of *release-left*) and right (i.e., the rule of *release-right*) forks respectively and goes to the *thinking* mode. This process is repeated eternally unless a deadlock state happens. In a deadlock state, all philosophers have picked up their left fork and wait for their right fork. In other words, all philosophers are in the *hasLeft* mode.

Tables 2 and 3 display the average running time of all approaches to refuting the safety properties "there exists no state in which all philosophers are in the hasLeft mode" in the particular case and "there isn't any deadlock state in the system" in the general case for the dining philosophers problem, respectively.

As seen in the tables, the average running time of SAMEDA is less than that of the others except EMCDM. In this problem, EMCDM can obtain precise knowledge from the information of checking smaller models (e.g., a model with three philosophers) and it has the best result in comparison with the others. When the size of models grows, GA and BS face the state space explosion problem.

5.2 Process Life Cycle Problem

In this problem, the life cycle of a process such as creation, execution, and death is described in the operating system. The created process is loaded into the memory provided that there is enough free memory. The loaded process requests CPU or I/O devices. If these devices are available, they are allocated to the process. Otherwise, the process should wait for them until they are free. When the waited process gets the devices and executes completely, it gives up all allocated resources and stops. In this problem, a situation in which all processes have stopped can be considered as a deadlock state.

Tables 4 and 5 display the average running time of all approaches to refuting the safety properties "there exists no state in which all existing processes are in the waiting state for I/O devices" in the particular case and "there isn't any deadlock state in the system" in the general case for the process life cycle problem, respectively. According to the tables, all approaches except BS can find a goal state in the large models successfully.

Table 2. Comparison of Average Running Time (s) of All Approaches to Refuting the Safety Property in the Particular Case for the Dining Philosophers Problem

Number of Philosophers	*maxDepth*	*popuSize*	SAMEDA (s)	BOAcl2 (s)	LBN (s)	EMCDM (s)	GA (s)	BS (s)
20	100	20	1.64	1.92	1.98	1.21	16.42	110.37
40	120	40	8.37	9.12	9.23	1.39	NF	1 028.00
60	140	60	23.79	24.90	25.43	1.43	NF	NF
80	180	80	45.93	46.72	46.04	1.84	NF	NF
100	220	100	80.10	86.34	87.37	2.83	NF	NF

Table 3. Comparison of Average Running Time (s) of All Approaches to Refuting the Safety Property in the General Case for the Dining Philosophers Problem

Number of Philosophers	*maxDepth*	*popuSize*	SAMEDA (s)	BOAcl2 (s)	LBN (s)	EMCDM (s)	GA (s)	BS (s)
20	100	20	3.21	3.67	4.31	1.12	30.45	7.83
40	120	40	4.97	5.21	5.63	1.36	NF	18.36
60	140	60	10.93	11.62	12.54	1.76	NF	NF
80	180	80	13.15	27.39	14.79	2.20	NF	NF
100	220	100	26.43	54.28	29.34	3.84	NF	NF

Table 4. Comparison of Average Running Time (s) of All Approaches to Refuting the Safety Property in the Particular Case for the Process Life Cycle Problem

Process Life Cycle	*maxDepth*	*popuSize*	SAMEDA (s)	BOAcl2 (s)	LBN (s)	EMCDM (s)	GA (s)	BS (s)
10_process_10_memory	40	20	0.67	0.72	76.43	7.94	9.85	NF
12_process_12_memory	45	30	0.95	1.09	126.42	11.03	18.31	NF
15_process_15_memory	50	40	1.31	1.84	156.36	14.61	NF	NF

Table 5. Comparison of Average Running Time (s) of All Approaches to Refuting the Safety Property in the General Case for the Process Life Cycle Problem

Process Life Cycle	*maxDepth*	*popuSize*	SAMEDA (s)	BOAcl2 (s)	LBN (s)	EMCDM (s)	GA (s)	BS (s)
20_process_8_memory	180	20	1.34	0.83	163.41	21.34	12.43	8.30
30_process_8_memory	280	40	1.49	1.21	184.30	25.42	29.93	21.30
40_process_8_memory	350	60	1.87	1.74	194.27	28.72	NF	NF
50_process_8_memory	450	80	2.15	2.13	203.37	31.40	NF	NF

It should be noted that there is only one goal state in the state space of models and the state is placed in the deeper places of the state space. Moreover, the larger models of this problem have the wider state space and we should increase the *beam width* parameter to increase the probability of finding a goal state. This causes BS cannot search the deeper places and thus it will fail.

5.3 Readers-Writers Problem

This problem describes a situation in concurrent programming in which many threads try to access a shared resource simultaneously. Some of these threads play the reader role and some others play the writer role. Several readers can read the shared resources at the same time provided that no writer is writing. Unlike the readers, only one writer can write to the shared resource at any time. In this problem, we consider a situation in which all readers and writers have finished their processing as a deadlock state.

Tables 6 and 7 display the average running time of all approaches to refuting the safety properties "there exists no state in which all readers and writers have finished their processing" in the particular case and "there isn't any deadlock state in the system" in the general case for the readers-writers problem, respectively. According to the tables, the average running time of SAMEDA is smaller than that of the others.

5.4 Firewalls Problem

Firewalls are tools that establish a barrier to prevent unauthorized access to or from secured and controlled internal networks [27]. Fig.8 shows a model of two networks separated by a firewall (FW). In the model, there are an internal and an external interface (IF), connected to a network of in-locations (LI) and a network of out-locations (LO) respectively. It is supposed that the firewall protects the internal network. Many safe or unsafe packets can be created at out-locations, and they flow through the network. Of course, only safe packets are allowed to be created at in-locations. Although the edges are directed, the flow of packets is bi-directional. If an unsafe packet wants to flow from the external network to the internal network, the firewall should deny the packet. Since there is not any deadlock state in the models of this problem, we cannot consider the safety property "there isn't any deadlock state in the system" in the general case of this problem.

Table 8 shows the average running time of all approaches to refuting the safety property "there exists no state in which an unsafe packet is denied by the firewall" in the particular case for the firewalls problem. As seen in Table 8, the running time of SAMEDA is less than that of the others. Since the model of this problem has infinite state space, EMCDM cannot work correctly. Similar to the process life cycle problem and due to the mentioned reason, BS cannot find any goal state.

Table 6. Comparison of Average Running Time (s) of All Approaches to Refuting the Safety Property in the Particular Case for the Readers-Writers Problem

Readers-Writers	*maxDepth*	*popuSize*	SAMEDA (s)	BOAcl2 (s)	LBN (s)	EMCDM (s)	GA (s)	BS (s)
4-R-4-W	50	40	2.14	2.83	3.21	30.28	11.87	183.47
5-R-5-W	70	50	5.21	6.02	14.67	34.75	84.28	530.18
6-R-6-W	90	60	21.74	24.39	36.39	42.93	154.95	563.95

Table 7. Comparison of Average Running Time (s) of All Approaches to Refuting the Safety Property in the General Case for the Readers-Writers Problem

Readers-Writers	*maxDepth*	*popuSize*	SAMEDA (s)	BOAcl2 (s)	LBN (s)	EMCDM (s)	GA (s)	BS (s)
4-R-4-W	50	40	3.14	4.31	4.38	31.28	12.97	7.31
5-R-5-W	70	50	6.03	7.94	16.40	36.49	82.30	14.62
6-R-6-W	90	60	25.42	27.38	37.06	43.63	163.52	19.64

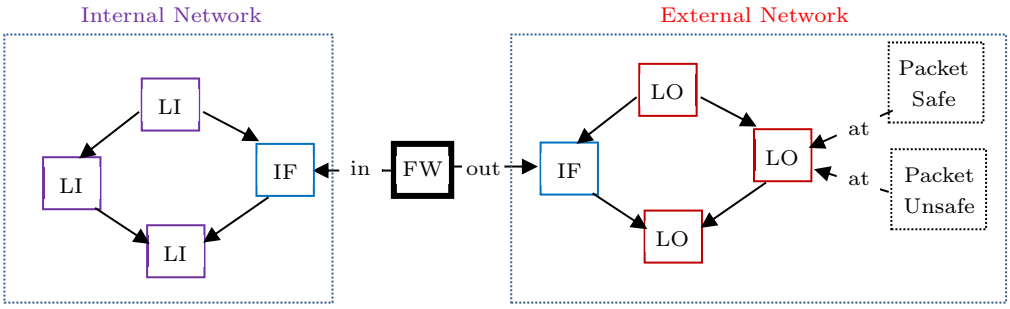


Fig.8. A model of two networks separated by a firewall (FW).

Table 8. Comparison of Average Running Time (s) of All Approaches to Refuting the Safety Property in the Particular Case for the Firewalls Problem

Firewalls	*maxDepth*	*popuSize*	SAMEDA (s)	BOAcl2 (s)	LBN (s)	EMCDM (s)	GA (s)	BS (s)
fire_6_LI_6_LO	60	20	2.370	4.11	25.31	NS	5.27	NF
fire_8_LI_8_LO	80	30	6.410	8.51	52.83	NS	92.51	NF
fire_10_LI_10_LO	100	40	9.173	11.04	80.73	NS	214.31	NF

5.5 Railway Switching Control Problem

This problem describes the Railway Switching Control System (RSCS) which has been designed to control the correct functionality of the railway switching system [28]. RSCS should guarantee the safety of the approaching trains in all railways approaching to the railway switches. Since the models of this problem do not have any deadlock state, we cannot consider the safety property "there isn't any deadlock state in the system" in the general case of this problem.

Table 9 shows the average running time of all approaches to refuting the safety property "there exists no state in which all trains have been crossed successfully" in the particular case for the railway switching control problem. According to Table 9, the average running time of SAMEDA is smaller than that of the others.

6 Discussion

The required time to reach a goal state is an important factor to evaluate the effectiveness of an approach. In fact, whatever an approach can find a goal state (especially an error) sooner, designers can correct the faulty design sooner and prevent spending more time and cost. According to Tables 2–9, SAMEDA outperforms other approaches in terms of such a factor. Generating shorter counterexamples through exploring less states can be another important factor to evaluate the effectiveness of an approach. For this purpose, we compare the length of the generated counterexamples and the number of explored states by all approaches in a sample model of the benchmarks. These tables, which can be accessed through the web④, confirm that SAMEDA generates shorter counterexamples through exploring less states in most of the benchmarks.

The accuracy (i.e., the ratio of the number of successful runs to the total runs) is another important factor to compare the effectiveness of the approaches. To do this, we have executed all approaches 30 times to refute the safety properties in the dining philosopher's problem with 12 philosophers. It should be noted that the *maxDepth* parameter affects the accuracy of the approaches. Large values for this parameter cause all approaches to find a goal state easily. Conversely, small values get harder (even impossible) to find a goal state.

Table 9. Comparison of Average Running Time (s) of All Approaches to Refuting the Safety Property in the Particular Case for the Railway Switching Control Problem

RSCS	*maxDepth*	*popuSize*	SAMEDA (s)	BOAcl2 (s)	LBN (s)	EMCDM (s)	GA (s)	BS (s)
RSCS_8_train_4_railway	50	20	3.19	6.41	12.74	3.83	24.61	17.31
RSCS_10_train_4_railway	60	30	8.93	18.31	30.91	11.49	73.93	43.02
RSCS_12_train_4_railway	70	40	15.29	33.72	43.21	16.32	110.34	64.28

④https://sourceforge.net/projects/groove-and-mceda/files/detailed_results.pdf/, May 2021.

Hence, we have considered the value of *maxDepth* equal to 24, the minimum possible value in this model. As seen in charts of Fig.9, SAMEDA, BOAcl2, LBN, LDM and EMCDM have a higher accuracy in comparison with the others except BS and IDA*. Although BS and IDA* can find a goal state in this model, according to Tables 2 and 3, they cannot find any goal state in the larger models of this problem.

Several parameters such as *maxDepth* and *popuSize* influence the accuracy of the approaches. To determine the impact level of *maxDepth* and *popuSize*, we have executed all approaches 30 times with several values of these parameters to refute the safety properties in the readers-writers problem with six readers and six writers.

As seen in the charts on the web[5], very small values of *maxDepth* have a significant negative effect on the accuracy of the approaches. While, large values of *maxDepth* increase the chance of approaches to reach a goal state. Since there is a minimum value for *maxDepth* in the considered benchmark, the values smaller than this minimum value cause BS and IDA* to fail and the larger ones have no positive effect. Hence, these approaches are ignored in the charts. Similar to the *maxDepth* parameter, very small values for *popuSize* decrease the successful likelihood of approaches. Moreover, large values for *popuSize* raise the level of accuracy. Note that *popuSize* does not have any effect on LBN, LDM, EMCDM, BS and IDA*. Therefore, these approaches are ignored in the charts.

As mentioned before, checking a safety property q in all possible states may expose the problem of state space explosion, especially in large and complex systems. To handle this problem, SAMEDA attempts to refute the safety property q by verifying the reachability property "not q". If SAMEDA can verify the reachability property successfully, a witness/counterexample will be generated. Otherwise, SAMEDA either has used up all available memory or has reached the maximum number of iterations. There is also a third possibility: q describes a correct behavior of the corresponding system. Reaching SAMEDA to the maximum number of iterations shows that q is satisfied in the explored states. Nevertheless, we cannot conclude that q is satisfied in all possible states. But, it can be said that the safety property q is verified with the probability of the ratio of the explored states and all possible states. For more clarity, we consider the safety property "unsafe packets never reach the internal network" in the firewalls problem with 10 LIs and 10 LOs. SAMEDA with parameters values, $maxDepth = 100$, $popuSize = 40$, $iterations = 100$, and $sampling\ rate = 0.6$, explores the following number of states (of course some of them may be repetitive):

$$\begin{aligned} &popuSize \times maxDepth + sampling\ rate \times \\ &popuSize \times maxDepth \times iterations \\ &= 40 \times 100 + 0.6 \times 40 \times 100 \times 100 \\ &= 244\,000. \end{aligned}$$

Whereas, the number of possible states of this model (so-called *nps*) is much larger than this number. It should be noted that the generation of all possible states needs a very powerful computer system. After computing *nps*, we can claim that, with probability of

Fig.9. Comparing the accuracy of the approaches to refuting the safety properties in the (a) particular and (b) general cases for the dining philosopher's problem with 12 philosophers. *maxDepth* = 24.

[5]https://sourceforge.net/projects/groove-and-mceda/files/impact_accuracy.pdf/, May 2021.

(244 000/*nps*), unsafe packets do not reach the internal network in the specified model of the firewalls problem.

7 Conclusions

In this paper, we used model checking for model-based safety analysis of systems modeled by graph transformations. The problem of model checking is that it may face the state space explosion problem in large and complex systems. Unfortunately, recently proposed meta-heuristic and evolutionary algorithms to handle this problem suffer from the low effectiveness in terms of accuracy and speed of convergence. In exploring the state space of such systems, since allowed rules on the current state depend on only applied rules on the previous state, a Markov chain can be employed to capture this type of dependencies. Moreover, Estimation of Distribution Algorithm (EDA) is an evolutionary algorithm to guide the search for the optimal solution by learning and sampling probabilistic models through the best individuals of the population at each generation. Hence, in this paper, we proposed an approach using Markov chain based Estimation of Distribution Algorithm (EDA) to improve the accuracy and speed of convergence in model checking of safety properties of systems. To evaluate the effectiveness of the proposed approach, it was implemented in GROOVE, an open source toolset for designing and model checking graph transformation systems. The comparison of average running time of SAMEDA with other approaches on several benchmarks showed that the proposed approach has a high speed and accuracy in comparison with the existing meta-heuristic and evolutionary techniques in safety analysis of systems. Using other probabilistic models can be a future research. Also, improving the proposed fitness functions can be another piece of future work.

References

[1] Rausand M. Reliability of Safety-Critical Systems: Theory and Applications. John Wiley & Sons, 2014. DOI: 10.1002/9781118776353.

[2] Lahtinen J, Valkonen J, Björkman K, Frits J, Niemelä I, Heljanko K. Model checking of safety-critical software in the nuclear engineering domain. *Reliab. Eng. Syst. Saf.*, 2012, 105: 104-113. DOI: 10.1016/j.ress.2012.03.021.

[3] Yousefian R, Rafe V, Rahmani M. A heuristic solution for model checking graph transformation systems. *Appl. Soft Comput.*, 2014, 24: 169-180. DOI: 10.1016/j.asoc.2014.06.055.

[4] Francesca G, Santone A, Vaglini G, Villani M L. Ant colony optimization for deadlock detection in concurrent systems. In *Proc. the 35th Annual IEEE International Computer Software and Applications Conference*, July 2011, pp.108-117. DOI: 10.1109/COMPSAC.2011.22.

[5] Alba E, Chicano F. Finding safety errors with ACO. In *Proc. the 9th Annual Conference on Genetic and Evolutionary Computation*, July 2007, pp.1066-1073. DOI: 10.1145/1276958.1277171.

[6] Rafe V, Moradi M, Yousefian R, Nikanjam A. A meta-heuristic solution for automated refutation of complex software systems specified through graph transformations. *Appl. Soft Comput.*, 2015, 33: 136-149. DOI: 10.1016/j.asoc.2015.04.032.

[7] Pira E, Rafe V, Nikanjam A. Deadlock detection in complex software systems specified through graph transformation using Bayesian optimization algorithm. *J. Syst. Softw.*, 2017, 131: 181-200. DOI: 10.1016/j.jss.2017.05.128.

[8] Pira E, Rafe V, Nikanjam A. EMCDM: Efficient model checking by data mining for verification of complex software systems specified through architectural styles. *Appl. Soft Comput.*, 2016, 49: 1185-1201. DOI: 10.1016/j.asoc.2016.06.039.

[9] Pira E, Rafe V, Nikanjam A. Searching for violation of safety and liveness properties using knowledge discovery in complex systems specified through graph transformations. *Inf. Softw. Technol.*, 2018, 97: 110-134. DOI: 10.1016/j.infsof.2018.01.004.

[10] Bicarregui J, Matthews B. Proof and refutation in formal software development. In *Proc. the 3rd Irish Workshop on Formal Methods*, July 1999.

[11] Koller D, Friedman N. Probabilistic Graphical Models: Principles and Techniques (1st edition). MIT Press, 2009.

[12] Pelikan M, Goldberg D E, Cantú-Paz E. Linkage problem, distribution estimation, and Bayesian networks. *Evol. Comput.*, 2000, 8(3): 311-340. DOI: 10.1162/10636560075-0078808.

[13] Lahtinen J, Kuismin T, Heljanko K. Verifying large modular systems using iterative abstraction refinement. *Reliab. Eng. Syst. Saf.*, 2015, 139: 120-130. DOI: 10.1016/j.ress.2015.03.012.

[14] Rozenberg G. Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations. World Scientific, 1997. DOI: 10.1142/3303.

[15] Kastenberg H, Rensink A. Model checking dynamic states in GROOVE. In *Proc. the 13th International SPIN Workshop on Model Checking of Software*, March 30–April 1, 2006, pp.299-305. DOI: 10.1007/11691617_19.

[16] Staunton J, Clark J A. Searching for safety violations using estimation of distribution algorithms. In *Proc. the 3rd International Conference on Software Testing, Verification, and Validation*, April 2010, pp.212-221. DOI: 10.1109/ICSTW.2010.24.

[17] Staunton J, Clark J A. Finding short counterexamples in promela models using estimation of distribution algorithms. In *Proc. the 13th Annual Conference on Genetic and Evolutionary Computation*, July 2011, pp.1923-1930. DOI: 10.1145/2001576.2001834.

[18] Staunton J, Clark J A. Applications of model reuse when using estimation of distribution algorithms to test concurrent software. In *Proc. the 3rd International Symposium on Search Based Software Engineering*, September 2011, pp.97-111. DOI: 10.1007/978-3-642-23716-4_12.

[19] Pira E, Rafe V, Nikanjam A. Using evolutionary algorithms for reachability analysis of complex software systems specified through graph transformation. *Reliab. Eng. Syst. Saf.*, 2019, 191: Article No. 106577. DOI: 10.1016/j.ress.2019.106577.

[20] Yousefian R, Aboutorabi S, Rafe V. A greedy algorithm versus metaheuristic solutions to deadlock detection in graph transformation systems. *J. Intell. Fuzzy Syst.*, 2016, 31(1): 137-149. DOI: 10.3233/IFS-162127.

[21] Yang X S. A new metaheuristic bat-inspired algorithm. In *Proc. the 2010 Nature Inspired Cooperative Strategies for Optimization*, May 2010, pp.65-74. DOI: 10.1007/978-3-642-12538-6_6.

[22] Baier C, Katoen J P. Principles of Model Checking. MIT Press, 2008.

[23] Sivanandam S N, Deepa S N. Introduction to Genetic Algorithms. Springer, 2008. DOI: 10.1007/978-3-540-73190-0.

[24] Groce A, Visser W. Heuristics for model checking Java programs. *Int. J. Softw. Tools Technol. Transf.*, 2004, 6(4): 260-276. DOI: 10.1007/s10009-003-0130-9.

[25] Edelkamp S, Lafuente A L, Leue S. Protocol verification with heuristic search. In *Proc. the 2001 Spring Symposium Series*, March 2001.

[26] Schmidt Á. Model checking of visual modeling languages. Bp Univ Technol Hung. 2004.

[27] Bellovin S M, Cheswick W R. Network firewalls. *IEEE Commun. Mag.*, 1994, 32(9): 50-57. DOI: 10.1109/35.312843.

[28] Azim M R S, Mahmud K, Das C K. Automatic train track switching system with computerized control from the central monitoring unit. *International Journal of u- and e-Service, Science and Technology*, 2014, 7(1): 201-212. DOI: 10.14257/ijunesst.2014.7.1.18.

Einollah Pira received his B.Sc. degree in computer engineering (software) from the University of Kharazmi, Tehran, Iran, 2000, his M.Sc. degree in computer engineering (software) from the Sharif University of Technology, Tehran, Iran, 2002, and his Ph.D. degree in computer engineering (software) from Arak University, Arak, Iran, 2017. Currently, he is an assistant professor with Department of Information Technology and Computer Engineering, Azarbaijan Shahid Madani University, Tabriz, Iran. His research interests include model checking, formal methods, software testing, and search-based software engineering.