Using Bayesian Optimization Algorithm for Model based Integration Testing

Somayeh Mohammady[1], Vahid Rafe[1], Erik Cuevas[2]
[1]Department of Computer Engineering, Faculty of Engineering, Arak University, Arak38156-8-8349, Iran
[2]Departamento de Electrónica, Universidad de Guadalajara, CUCEI, Av. Revolución, 1500, Guadalajara, Jal, México

s.mohammadi6888@gmail.com, v-rafe@araku.ac.ir, e.cuevas@guadala.ac.mx

Abstract
Model based testing is an automated process in which executable test cases are derived from behavioral models of a system. Model checking is another verification approach in which all reachable states of a system are generated. In the literature, there are different approaches which suggest using model checkers for model based test case generation. Since all possible states and paths are generated by the model checker, selecting different paths in the state space as test cases based on some coverage criteria seems a promising solution. However, all these approaches suffer from two main challenges. The first challenge is the state space explosion problem which prevents the model checker to generate all the states. The second one is generating redundant test cases (paths). Recently, methods using meta-heuristic and evolutionary approaches have been proposed to cope with these two problems. Therefore, exploring a portion of the state space to detect the test objectives using an optimization approach can be a proper way to manage the state space explosion and generates the optimal test suites with the least redundancy. In this paper, an approach is proposed using Bayesian optimization algorithm (BOA) to generate test cases for service oriented systems specified through graph transformation. In the proposed approach, test suite is a set of paths on the state space that starting from an initial state and leading to the states in which all the test objectives is satisfied. In this research, we have implemented BOA with three different structures in GROOVE, an open source toolset for designing and model checking graph transformation. Experimental results show that our solution generates better results in terms of coverage, memory usage and speed in different case studies in comparison with the existing approaches.

Keywords: Model-based Testing, graph transformation, Bayesian optimization algorithm, integration testing, Data flow.

1. Introduction

Software testing is a validation process which should reveal system bugs, errors and shortcomings. However, this process is an expensive, time-consuming and error-prone task [1]. Usually, 30-50% of software developing efforts is spent to testing [2]. Hence, finding a proper method for software testing is an important task.
Model based testing (MBT) [3] is a black box technique in which executable test cases are derived from behavioral models of a system. Test model is extracted from intended behaviors of a system. MBT is a well-known technique to generate [4] test cases due to easy change, reuse and shared models.
During the past years, in order to improve the MBT, various techniques such as symbolic execution [5], Deductive theorem proving [6], Random testing [7], search-based technique [8, 9], constraint solving [10], AI planning [11] and model checking [12, 13] have been proposed. In addition, recently, a method using randomized algorithms has been proposed to generate the test from systems specified through graph transformation by the underlying model checker [57].
Model checking [14] is an automatic technique to detect system errors that allow automatic generation of test cases (TC) from models by exploring the state space. State space is a set of all reachable states of a system which describes the behavior of that system. A state is a set of variables and their current values in a specific status of execution. In model checking-based test case generation technique (MCT), a model of a system and a test objective as a trap/reachability property is provided to the

model checker. Model checker detects a counterexample/witness to violate/satisfy the property. A counterexample/witnesses is a path starting from an initial state ending in a state that the property is refused/verified. These paths can be used as TCs [15]. However, checking the test objectives in the form of properties suffers from the state space explosion and test case redundancy. State space explosion is the problem in which all reachable states cannot be explored due to resource limitations [14]. Recently, in order to deal with the state space explosion, methods using meta-heuristic and evolutionary algorithms such as Genetic Algorithm (GA) [16], Practical Swarm Optimization (PSO) [17], Ant Colony Optimization [18] and Bayesian Networks [19] have been proposed. The other problem is that the most of generated counterexample/witnesses is redundant.

Estimation of distribution algorithms (EDAs) [20] are evolutionary algorithms that uses probabilistic model of promising solutions to generate new individual instead of biological evolution (i.e. crossover and mutation). This model is learned from the promising solutions through machine learning and then the learned model is sampled to generate offspring. BOA one of the EDAs that performs the optimization process by Bayesian Network (BN). BN [22] is a probabilistic model that indicates random variables and their conditional dependencies via a directed acyclic graph.

In the literature, selecting an appropriate test model is important. One of the proper languages to specify software systems is graph transformation system (GTS). GTS is a formal notation to model behaviors of systems with complex structures that is widely used in software development cycle [23].

In this paper, to improve the model based test case generation approaches in terms of test quality, coverage, convergence speed and time, an approach is proposed using BOA to perform integration testing. To do so, the system should be specified through GTS. Hence, we propose a solution that BOA is applied to the challenges of model checking-based test case generation approaches. In GTS, a function of system is expressed by a rule and all reachable interactions between rules are expanded through graph transformation. Therefore, TCs are paths on the state space that starting from an initial state and leading to the state in which at least one def-use test objectives is satisfied. These paths are executable sequence of functions that produce interactions between system unites. In this paper, the GROOVE toolset [24] is used to implement our approach which is an open source tool. In order to manage the state space explosion and redundant TCs generation problem, BOA is employed for partially exploring the state space graph in which the all def-use test objectives are satisfied. To evaluate the efficiency of the proposed approach, we compare the obtained results on three different case studies with the other techniques.

The rest of the paper is organized as follows: Section 2 explains the main concepts of the proposed approach such as GTS, MBT, data flow coverage and BOA. The related works are surveyed in section 3. In section 4, we discuss the proposed approach in details. Experimental results are presented in section 5. Finally, we conclude the paper in Section 6.

2. Background

In this section, we briefly survey the basic concepts of the proposed approach.

2.1. Graph Transformation system

Graph transformation system [25] is a formal language to model software systems with dynamic structures. A GTS is defined by $\mathrm{GTS} = (\mathrm{TG}, \mathrm{HG}, \mathrm{R})$, where TG is a type graph, HG is a host graph and R is a set of transformation rules. TG is defined by TG= $\{\mathrm{TG}_N, TG_E, Src, Trg\}$, where TG_N is a set of all node types and TG_E is a set of all edge types. $Src: TG_E \to TG_N$ and $Trg: TG_E \to TG_N$ are two functions which determine, respectively, the source and destination node of an edge. The initial configuration of a system is specified through HG.

R is defined by $\mathrm{R} = (\mathrm{LHS}, \mathrm{RHS}, \mathrm{NAC})$. A graph transformation rule manipulates a host graph. Thus, graph transformation (GT) is performed through applying these rules repeatedly to the host graph. LHS (left hand side) and RHS (right hand side) are two graphs that represent, respectively, pre-conditions and post-conditions of a rule and overlap somewhat with HG. NAC (Negative Application

Condition) is an extension of LHS that is used to check the absence of a structure, and of course it can be null.

To apply a rule, LHS must match HG, that is, LHS structure must be found in HG. If so, then RHS structure replaces LHS on the current HG. Usually**,** several rules may match the HG; by applying all applicable rules repeatedly to the HG, the state space is created. A state space is a digraph in which nodes are a set of all reachable states of a system and edges represent transformations between them. The difference between LHS and RHS elements determines the result of applying a GT rule. These differences are divided into three sets of elements, as defined below:

R_D = $\{N_d, E_d\}$ with $N_d=LHS_N-RHS_N$ and $E_d=LHS_E-RHS_E$
R_P =$\{N_p, E_p\}$with $N_p=LHS_N\cap RHS_N$ and $E_p=LHS_E\cap RHS_E$
R_C = $\{N_c, E_c\}$ with $N_c=RHS_N-LHS_N$ and $E_c=RHS_E-LHS_E$

where R_D is a set of deleted elements (nodes/edges), R_P is a set of preserved elements and R_C is a set of created elements. As mentioned earlier, the HG structure specifies the applicable rules. On the other hand, by applying a rule; elements are removed/added from/to the HG. So, the currently applied rules can affect the applicable rules on next states. This is due to the dependencies between rules. Suppose that r_1 and r_2 are two rules and LHS_{r_1} is a set of all edges and nodes of $LHS_{r_1}$and NAC_{r_1}. If $LHS_{r_1} \cap R\_D_{r_2} \neq \emptyset$ or $LHS_{r_1} \cap R\_C_{r_2} \neq \emptyset$, then rule r_1 depends on r_2.

There are several tools for modeling the systems by GTS such as GROOVE, AGG [26], VIATRAL2 [27], and NuSMV [28]. GROOVE is an open-source toolset for designing and model checking graph transformation systems. In GROOVE, the LHS, RHS and NAC of a rule are designed as a single graph. In a rule graph new: /del: label is used to specify the added/removed elements to/from the HG and not: is used to define NAC. NAC elements are marked with red dotted lines, and added and removed elements are specified by green solid lines and blue dashed lines, respectively.

In this paper, the GROOVE editor and simulator is used to model a system and explore the model's state space. As an example, consider an online railway ticket reservation system in which the user is able to register, search travels, book and cancel the e-ticket. The functions signatures for this system are given in pseudocode1. This system is modeled in the GROOVE toolset. The type graph and start graph of the model are illustrated in fig.1 and fig.2, respectively. Fig. 3 shows the booking rule; that is, a seat is reserved for the passenger (RHS) if no ticket is issued for it (NAC).

Pseudocode1. Railway ticket reservation system

```
Public interface
{
Public Void Register (string Name, string Password, string Email address);
Public Void Login (string Name, string Password);
Private String Ticket availability checking (date Time & Date, string Port, string Destination, integer Number);
Public String Book a ticket (string Name, string Phone Number, string Credit Card Identity);
Public Void Cancel a ticket (string Serial);
Public Boolean Payment (string Serial, money Amount);
}
```

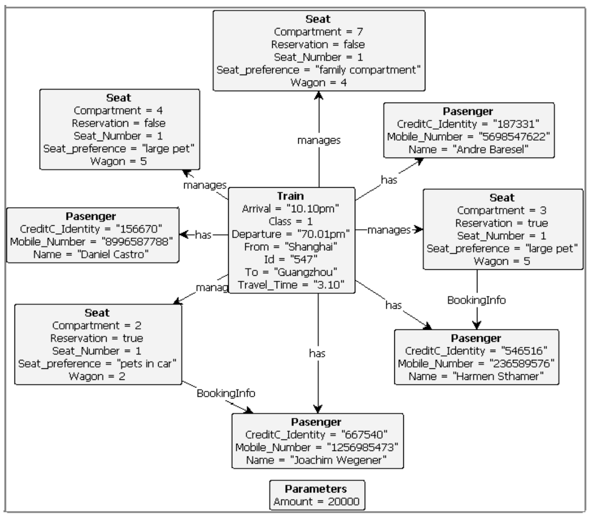

Fig2. The start graph for the railway ticket reservation system

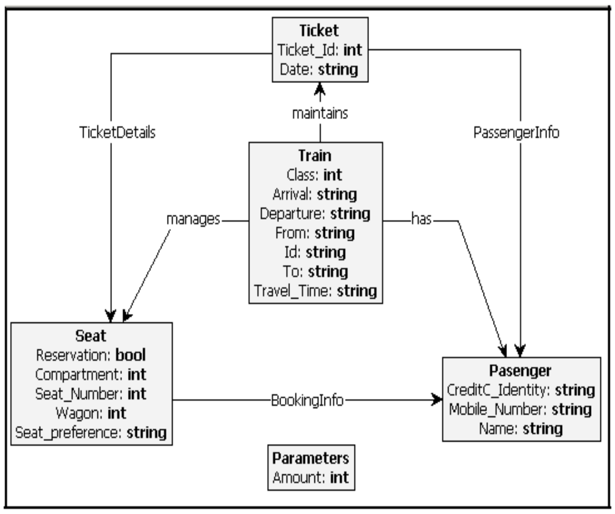

Fig1. The type graph for the railway ticket reservation system

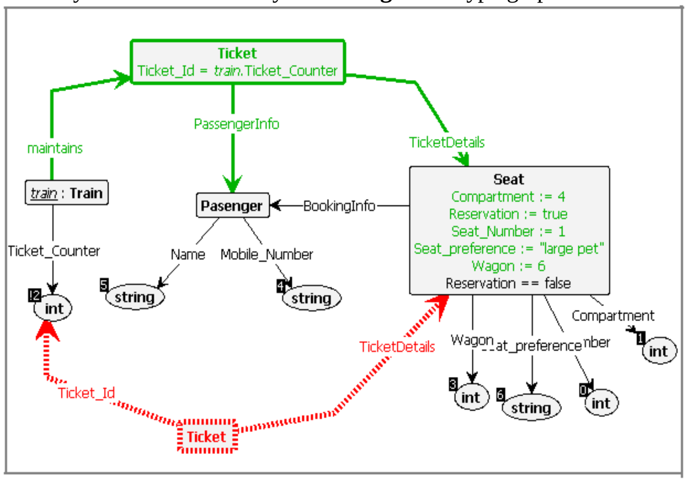

Fig3. The e-ticket booking rule

2.2. Bayesian Optimization Algorithm

Bayesian optimization algorithm is one of the EDAs which captures the partial solutions of promising solutions by a probabilistic model. In BOA, a Bayesian network is learned from the promising solutions and then valuable structures are reused in the offspring through sampling. BN is a probabilistic model that represents conditional dependencies among random variables via a directed acyclic graph. A BN is defined by structure and parameters.

Structure: BN structure is a directed acyclic graph in which the nodes are random variables and edges are conditional dependencies between them. Suppose that x and y are two nodes in BN. If there is an edge from x to y, then x is the parent of y. BN structure can be fixed or constructed through structure learning.

Parameters: parameters are the conditional distribution of variables according to the different values of their parents that are represented by conditional probability tables. A BN defines the joint probability distribution of n random variables [29]; the equation for this distribution is given below:

$$p(X_1, X_2, \dots, X_n) = \prod_{i=1}^{n} p(X_i \mid parent(X_i)) \qquad (1)$$

where x_i is a random variable and p (x_i | parent (x_i)) is the conditional probability distribution of x_i according to its parents. BOA starts with an initial population of random solutions; then, in different iterations, BN is learned from the fittest individuals, and offspring are sampled from this built BN. In the sampling procedure non-significant individuals are replaced by promising offspring. Algorithm 1 shows the BOA procedure.

Algorithm1. The Bayesian Optimization Algorithm

```
BEGIN
    Generate initial population randomly;
    Calculate the fitness of individuals;
    While (termination criteria is not satisfied)
        1.  Select N fittest individual;
        2.  Learn a Bayesian Network;
        3.  Sample M new candidate from built network;
        4.  Incorporate new individuals into population;
        5.  Calculate the fitness of individuals;
    End while
END
```

2.3. Model Based Testing

In the literature of modeling**,** it is noted that abstract description of a system, regardless of implementation details, can be helpful to accurately analyze it. At present, models have a wide range of applications such as Model-driven engineering, Model-driven architecture, software system documentation, software testing, etc. In software testing, models are used for description, documentation, TC generation, test execution, and oracles. MBT [3] is a well-known technique in which the testing process is based on a model that describes the intended behavior of the system under test (SUT). The test model is extracted from SUT specifications and test cases are derived from it; then, SUT is considered as a black box and executed by suggested test cases. In order to validate the software system, TCs outputs are compared with the expected outputs. The TC generation process is a critical activity. MBT simplifies this by automating the TC generation with respect to a coverage criterion.

Correctness, unambiguousness, completeness, verifiability, and modifiability are some of the properties appointed to enhance the quality of the system requirements, that the nature of MBT can be helpful to achieve them. To drive a test model of SUT, MBT requires precise and detailed inference of system requirements, which makes the specified requirements more reliable. Also, expressing a system in the form of formal structures eliminates any ambiguity about the requirements. Test models have clear views of the systems, through which, some of shortcomings or incompatibilities can be detected. On the other hand, modifying the test model while changing the system covers the maintenance and modifiability properties [30].

However, test model construction is an error-prone task. That is, the quality of TCs depends on test models, and false design during the modeling leads to the generation of inapplicable TCs. In addition, the automatic generation of TCs with inadequate coverage criteria leads to an infinite number of TCs, called test case explosion.

2.4. Data Flow coverage criteria in graph transformation system

A data flow [1] is a path from a point that a variable is defined (Def) to another point that the variable is referenced (Use). In order to identify the wrong definition of variables, data flow coverage criteria focus on the relations between the variables definitions and their uses. In a GTS, a system operation is expressed through a rule; and, as mentioned above, there are dependencies between these rules. This is due to the data flows between GT rules which can be dynamically extracted. A data flow in a GTS consists of two rules, r1 and r2; the rule r1adds an object (Def) to HG, which is used by the rule r2 (use). Suppose that r1 and r2 are two GT rules and $Dep_{r_2} = LHS_{r_2} \cup R\_D_{r_2}$, if $Dep_{r_2} \cap R\_C_{r_1} \neq \emptyset$ „

so r1 and r2 are Def and Use rules, respectively. Therefore, (r1, r2) is a Def_Use pair that must be covered by a TC.

Data flow coverage criteria in GTS are defined by extracting a Dependency Graph (DG) from SUT and studying the types of dependencies between the rules. A DG is defined by DG = {G, OP, op, lab}, where G is a digraph, OP is a set of system rules, op is a function that maps the system rules to the DG nodes and lab is an edge labeling function that contains labels = {create, read, delete, update}. Suppose that n_1 and n_2 are two DG nodes denoting rule r_1 and r_2, respectively. According to this, labeling an edge is as follows:

- If $R\_C_{r_1} \cap R\_D_{r_2} \neq \emptyset$ then there is an edge from n_1 to n_2 with label <create, delete>.
- If $R\_C_{r_2} \cap LHS_{r_2} \neq \emptyset$ then there is an edge from n_1 to $n_2$with label <create, read>.
- If r_2 updates the value of an attribute of $R_C_{r_1}$then there is an edge from n_1 to n_2 with label <create, update>.

Accordingly, Create_Delete, Create_Read and Create_Update criteria are defined [31]. These criteria are described in Table 1.

Table1. Data flow criteria in GTS

Data flow criterion	Purpose
Create_Read(C_R)	Test all (C_R) rules; where r_1 adds an element to HG, that is an element of LHS_{r_2}.
Create_Delete (C_D)	Test all (C_D) rules; where $r_1$adds an element to HG, that is deleted by applying r_2.
Create_Update (C_U)	Test all (C_U) rules; where $r_1$adds an element to HG, that is updated after applying r_2.
Dependencies	Test all Def_Use rules.

3. Related works

Models are basic artifacts in MBT. These models can be either formal or informal. The Unified Modeling Language (UML) is a well-known modeling language intended to provide diagrams for describing the behavior of software systems. To generate test cases from UML diagrams, there are many approaches such as test case generation using use case diagram [32], collaboration diagram [33], UML state chart [34], and UML sequence diagram [35] are some examples. Also, the collaboration diagram has been used to determine the adequacy of test suites [36]. In [37, 38], the authors present a method using UML models for regression testing. Test case generation from data flow graphs [39] and finite state machines [40] are other approaches.

There are some other approaches trying to employ formal models. GTS is a formal language to model systems that is widely used in model checking based testing. In Model Checking based test generation technique (MCT), a model of the system and a test objective as a trap/reachability property is provided to the model checker. The model checker detects a counterexample/witnesses to violate/satisfy the property. This idea has been performed by GROOVE [41]. The most important weakness in MCT is state space explosion and test case explosion. In [42], the authors present a method using Visual Contracts (VC) to generate TCs. VCL is a graphical notation of system operations in the form of pre-conditions and post-conditions. In VCL, an operation is specified through a pair of graphs that depicts the state of the system before and after the execution of an operation. The proposed approach also has introduced several data flow coverage criteria based on dependencies and conflict relations between GT rules. Each TC indicates a random sequence of applicable rules starting from an initial state. If a rule sequence is executable, it is considered as a final TC. The proposed method is implemented in the AGG tool, a tool for modeling and analyzing the systems defined by GT. In [43], VCs are transformed to java modeling language to generate test cases. This approach uses pre-conditions for test data generation and post-conditions for test oracles. The recent work related to our proposed approach is search based testing [57]. In this work, a search based method using several heuristics like Genetic Algorithm (GA), Particle Swarm Optimization

(PSO), Bat Algorithm (BA), Gravitational Search Algorithm (GSA), and a hybrid algorithm using GA and PSO (HGAPSO) is proposed to generate test of models specified through GTS. The approach is implemented in the GROOVE. Experimental results show that this test generation method has significantly better coverage than MCT and model-based testing using visual contracts.

During the past years, in order to achieve the optimal test suite, meta-heuristic and evolutionary algorithms such as genetic algorithms, greedy algorithms, and estimation of distribution have been employed. In [44], a GA-based approach is proposed to generate test cases for a web application. In this gray-box method, transition relations are extracted from the RDG graph (Request Dependence Graph). In each generation of GA, different user sessions are mixed to cover more transition relations. Experimental results confirm that the solution can generate a high coverage test with a small size. In [45], the authors proposed a GA-based method to generate test cases for object-oriented systems using the activity diagram. In the proposed method, an activity diagram of a system is converted to an activity graph, then weights are assigned to all edges according to the number of visited nodes from the initial node. Each chromosome is a unique path between the desired source node and destination node. High weighted chromosomes are considered as TCs. Also, in [2], the same authors proposed a technique which employs other evolutionary and greedy heuristic algorithms such as Greedy, Struggle GA, Steady-state GA, evolutionary programming, and evolutionary strategies. Experimental results show that these meta-heuristics are more efficient in terms of test suite size and coverage. In [46], the authors proposed an automatic test generation approach using an evolutionary algorithm. The aim of this approach is the automatic generation of test data for structural tests.

In [47], the authors proposed two new approaches using Scatter search and a hybrid algorithm using scatter search and EDAs for branch coverage. This approach is the first application of a hybrid algorithm using EDAs to generate test cases. The proposed method uses of re-search technique to fulfill the coverage, that is, initially the EDA is used to cover the branches, and when EDA computational limit is reached, SS is employed to cover the uncovered test objectives. To compare the performance of the proposed method, several EDAs are implemented. Experimental results confirm that these algorithms and their collaboration are very promising to achieve full coverage. Moreover, the authors in [48] proposed an EDA based testing approach which generates test data for unit testing. This approach is proposed for branch coverage criteria. The initial population is a set of sequences of function calls with a specific length. High coverage sequences are selected to sample the names of the functions.

In the literature, various methods have been proposed to employ the Bayesian network in the testing process. In [49], the authors present a BN based strategy for regression testing. In [50], a BN is used for GUI testing. In this approach, BN is constructed using the prior knowledge of testers and then values are updated using the results of test cases. Also, BN has been used as a prediction tool to predict defects, failures, compatibilities, reliabilities, and qualities in the software development process. Approaches like predicting software defects by hierarchical Bayesian model [51], activity-based BN to predict software quality [52], reliability prediction [53], and failure prediction [54] are some examples.

4. The proposed approach

In this section, an automatic approach based on BOA is proposed to generate an Integration test from the system specified through GTS. Integration testing [1] is a functional testing in which the interactions between the developed units are tested to reveal the errors that may occur in the integrated units. In GTS, a system unit is expressed by a rule, and all reachable interactions between rules are expanded through graph transformation. As previously mentioned, Def_Use relations between GT rules are detectable; hence test suite is a set of paths on the state space graph starting from the initial state and leading to the states where the all def-use test objectives are satisfied. There are some hierarchical dependencies among Def_Use rules; it means that, in a Def_Use pair, the Def rule is a Use rule of another Def_Use pair or a Use/Def rule belongs to several Def/Use rules. Moreover, there are other unknown dependencies; so that, the applied rules to the current state determine the next applicable rules. BOA is able to capture these dependencies through learning a BN. Thus, by learning a BN from the fittest TCs, BOA can select a more promising rule from the matching rules to apply to the current state. In our approach, the structure of the built BN is a fixed chain and only the parameters are learned.

The architecture of the proposed approach is illustrated in Fig. 4. As seen in this figure, first all the test objectives (Def_Use rules) are extracted from an abstract model and the initial population is randomly generated. TCs are explored in the state space to determine their coverage. After computing the coverage values, a set of promising TCs is selected to learn the BN parameters using the maximum likelihood hypothesis [55]. In the sampling step, offspring are sampled from the built BN and replaced with the ineffective solutions. These steps are iterated until the termination criteria, achieving 100% coverage or reaching the time limit of 30 minutes, are met. The final step is the test suite minimization process. The rest of this section explains the process in detail.

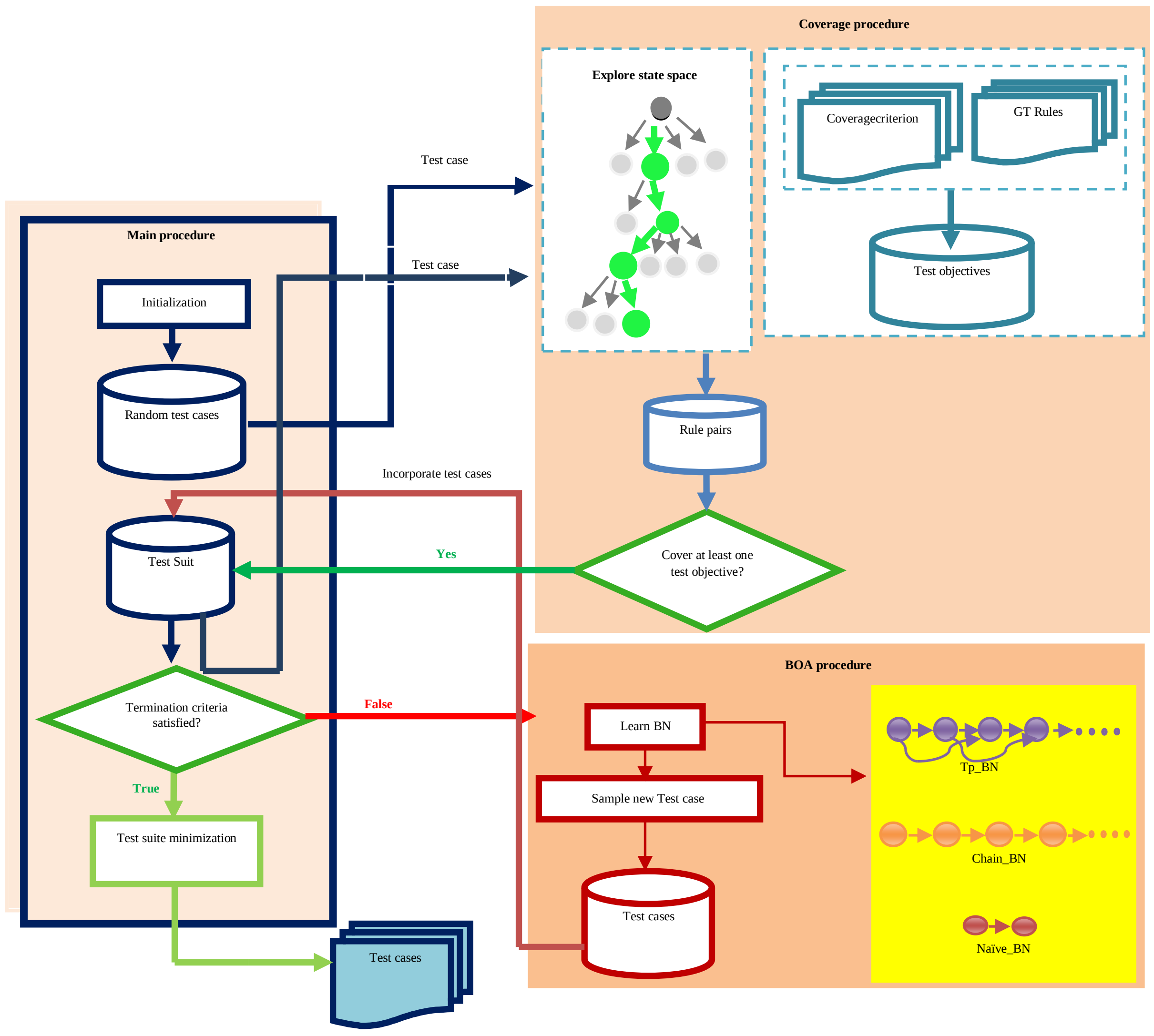

Fig4. The architecture of the proposed test case generator

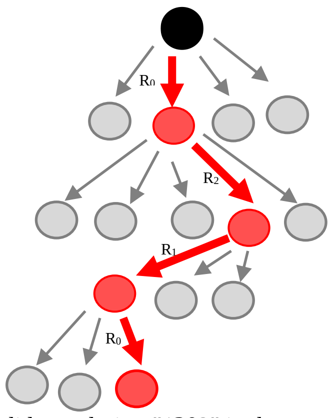

Fig5.The candidate solution "1302" in the state space graph

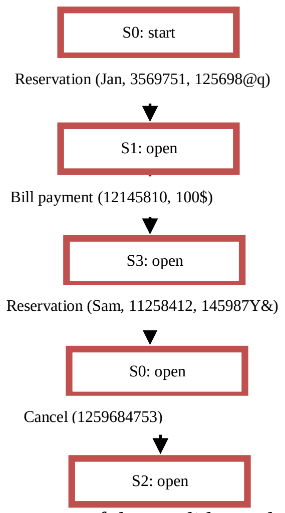
S0: start
Reservation (Jan, 3569751, 125698@q)
S1: open
Bill payment (12145810, 100$)
S3: open
Reservation (Sam, 11258412, 145987Y&)
S0: open
Cancel (1259684753)
S2: open

Fig6.The test case of the candidate solution "1302"

4.1. Test Case: In our approach, a TC is an executable sequence of system functions with the current values of their variables. Actually, a TC (also called candidate solution) is a finite path on the state space graph, which has a predetermined length and starts from the initial state. As mentioned earlier, the nodes in the state space graph are reachable states, and the outgoing transitions are applied rules. So, a candidate solution encodes a TC with a string of numbers which determine the outgoing transitions indexes. For example, assume that the online railway ticket reservation system has four transformation rules including r_0 (reservation), r_1 (cancel), r2 (bill payment), and r3 (login); and let "1302" be a candidate solution in this system. Fig. 5 illustrates a portion of the state space and the solution "1302" marked with the red colored edges. This TC is shown in Fig. 6.

4.2. Def_Use rules Coverage: A Def_Use pair is covered by a TC if there is at least one definition-clear path between Def and Use variables. A definition-clear path [1] is a path between Def and Use variables in which the Def variable is not redefined. Accordingly, a definition-clear path between Def and Use rules is defined as follows:

Definition-clear path between rules: suppose that r_1 and r_2 are Def and Use rules, respectively, $\pi =< s_5 r_1 s_7 r_8 s_1 r_5 \ldots \ldots . s_1 r_3 s_9 r_2 >$ is a path between r_1 and r_2, and DU_element = $((LHS_{r_2} \cup R\_D_{r_2}) \cap R\_C_{r_1})$ is Def_Use elements. Therefore, π is a definition-clear path between r_1 and r_2 if the applied rules from r_1 to r_2 do not make $DU\_element = \emptyset$. To determine the Def_Use rules, a pseudocode is given in Algorithm 2.

Algorithm2. The process of determining Def_Use rules

```
BEGINE
    Input: rules;
    Output: Def_Use rules;
    (C_R) _set: a set of Create_Read rules;
    (C_D) _set: a set of Create_Delete rules;
    (C_U) _set: a set of Create_Update rules;

    For all rules do
        1.    extract the Produced_set (added nodes and edges by rule);
        2.    extract the Updated_set (updated nodes and edges by rule);
        3.    extract the Consume_set (removed nodes and edges by rule);
        4.    extract the Preserved_set (nodes and edges in the precondition);
    End for

    For all $P_2^{rules}$ do
        1.   if (first rule Produced_set $\cap$ second rule Preserved_set $\neq \emptyset$)
```

```
                 Add rule pair into (C_R) _set;
         2.  if  (first rule Produced_set ∩ second rule Consume _set ≠ ∅)
                 Add rule pair into (C_D) _set;
         3.  if  (first rule Produced_set ∩ second rule Updated_set ≠ ∅)
                 Add rule pair into (C_U) _set;
    End for
END
```

4.3. Fitness function: to confirm the coverage of the test objectives, the all applied Def_Use rules in a TC path must be achieved. Therefore, the encoded solution is explored in the state space, and a sequence of rule calls is extracted. Then, all Definition-clear paths between these pairs are identified. The pseudocode for determining the TC coverage is shown in Algorithm 3. Our fitness function, $Fitness(TS)$, is the number of uncovered test objectives for a given test suite which should be minimized. The fitness function is defined using the following equation (2), where TS is the given test suite, n is the number of TCs, test_obj is a set of all the test objectives, and covered_obj is a set of covered Def_Use rules by the TC$_i$.

$$Fitness(TS) = \{test\_Obj\} - \sum_{i=1}^{n}\{Coverd\_Obj\ \ of\ TC_i\} \qquad (2)$$

Algorithm3. The fitness function

```
BEGINE
    Input: test case, M: a specified model;
    Output: test case coverage;
    Test_obj: a set of Def_Use rules;
    TC_path: explored TC path in the state space;
    Covered_obj: a list of covered Def_Use rules for given TC;
    Test suite: a set of test cases;
    Let a D_U_path be a path that starts with a Def rule and ends with a Use rule;

    While (there is a D_U_path in the TC_path)
        If (criteria== Create_Read)
                If ($R\_C_{def} \cap LHS_{use} \neq \emptyset$)
                    Add Def_Use rule in covered_obj;
        Return;
        If (criteria == Create_Delete)
                If ($R\_C_{def} \cap R\_D_{use} \neq \emptyset$)
                    Add Def_Use rule in covered_obj;
        Return;
        If (criteria == Create_Update)
                If ($R\_C_{def} \cap R\_U_{use} \neq \emptyset$)
                    Add Def_Use rule in covered_obj;
        Return;
    End while
    If (covered_obj≠ ∅)
        Add test case in test suite;
        Test_obj= Test_obj - covered_obj;
End if
END
```

4.4. Learning: After computing the coverage, promising TCs are selected to learn a BN. To select promising solutions, the truncation selection [56] is utilized. In our approach, the BN structure is a fixed chain and only the parameters are learned [19]. So, the three different structures are used for the BN, as follows:

nBOA: In nBOA, BN structure is a two-node chain that preserves the applied rules over the previous and current states. The parameters of nBOA are represented by two probability distribution tables.

The first table determines the probability distribution p (X_0= r_i) and the second one determines the conditional probability distribution p (X_1= r_j|X_0= r_i).
tpBOA: In tpBOA, BN structure is an n-node chain where n is the candidate solution length, and two predecessor nodes of each node are its parents. In this chain, X_k is the corresponding node to the gene_k that preserves the applied rules over the state_k. The parameters of tpBOA are represented by n probability distribution tables. The first table determines the probability distribution p (X_0= r_i), the second one determines conditional probability distribution p (X_1= r_j|X_0= r_i), and the others determine conditional probability distribution p (X_k= r_j|X_{k-1}= r_i, X_{k-2}= r_l).
cBOA**:** The BN structure in cBOA is an n-node chain similar to tpBOA, and the predecessor node of each node is its parent. The parameters of cBOA are represented by n probability distribution tables. The first table determines probability p (X_0= r_i), and the others determine conditional probability p (X_k= r_j|X_{k-1}= r_i).

The pseudocode for parameter learning in the cBOA is shown in Algorithm 4. As can be seen in this algorithm, first all the applied rules to the explored promising solutions are preserved as the default values of the corresponding node. Afterward, for each state corresponding to that node, the relative frequency of each item is computed.

Algorithm 4. The learning process in the cBOA

```
BEGINE
    Input: promising solutions;
    Output: BN;
    Let Slo_path be a set of explored path of solutions in the state space;

    While (solution_ length>0)
        Get new node_i ();
        While (Slo_path≠∅)
            Add applied rules to the Slo_path (state_i) as the node_i items;
        End while
    End while
    While (there is a node)
        If (node_i is the first node)
            Get initial node ();
                While (Slo_path≠∅)
                    Get the frequency of each item in the Slo_path (state_0);
                    Item probability= item frequency / the number of selected solutions;
                End while
        Else
            Get node_i ();
            Get node_{i-1}();
            While (there is an item in node_{i-1})
                While (there is an item in node_i)
                    While (Slo_path≠∅)
                        Get the frequency of (item_{i-1}, item_i) in the Slo_path (state_{i-1}, state_i);
                        (item_{i-1}, item_i) probability= frequency of (item_{i-1}, item_i) / frequency of item_{i-1}
                        in the Slo_path (state_{i-1});
                    End while
                End while
            End while
        End if
    End while
END
```

4.5. Sampling: In this step, offspring are sampled from the built BN and replaced with the worthless solutions. That is, the state space graph is explored from the initial state for a maximum length (or until a null state is reached), and in each move, a rule that has the most frequency compared with the other applicable rules is applied to the current state. This heuristic leads to the detection of the most promising states. The pseudocode for sampling the new solutions in the cBOA is shown in Algorithm 5. According to this algorithm, first all the applicable rules to the current state are obtained. Then, considering the parent value, a matched rule which maximizes the p (X1= r |X0= previous rule name) value is selected.

Algorithm 5. The sampling process in the cBOA

```
BEGINE
   Input: population;
   Output: new population;
   Let Slo_index be solution count replaced by sampling;
   Let individual be offspring;
   Let max_level be max length of test case;
   Let current rules be a set of all applicable rules to the current state;
     While (Slo_index< count of the population)
       Clear individual;
       Gene_index=0;
       Current state = initial state;
       Current rule name=null;
       Previous rule name=null;
             While (current state! = null && Gene_index< max_level)
               Current rules= current state. Get matches ();
              If (current rules is empty)
                Return;
              End if
             If (previous rule name is null)
   Best rule= rule r of current rules by which the value of p(X0 = r) is maximized;
             Else
                Best rule = rule r of current rules by which the value of p (X1= r |X0= previous rule name) is
                maximized;
             End if
              Individual. Add (the index of Best rule transition);
              Apply Best rule over the current state;
              Do next ();
              Gene_ index++;
              Previous rule name = current state rule name;
       End while
   Population. Add (Individual);
   End while
END
```

4.6. Test reduction: The presence of redundant test cases in the generated test suite may occur due to the overlapping of TCs. That is, there exists a subset of TCs which covers all covered test objectives of TS. This is a contradictory event to the effectiveness of a test suite in terms of resources and time. As a result, the redundant TCs should be eliminated which leads to the test case reduction (also called test suite minimization). In this paper, the test suite minimization process is employed to achieve an effective test suite. Our minimization process is done in two steps, redundant TC elimination and TC length reduction. These steps are described by the following definitions:

Redundant TC elimination: suppose that ts is a test suite, tc_coveredobj is the covered objectives of a TC with the highest coverage, and tc'_coveredobj is the covered objectives of another TC. Therefore, if $\mathrm{tc}'\_\mathrm{coveredobj} - \mathrm{tc}\_\mathrm{coveredobj} = \emptyset$ then tc' is redundant.

TC length reduction: suppose that tc is a TC, $\pi =< s_5 r_1 s_7 r_8 s_1 r_5 \dots\dots s_1 r_3 s_9 r_2 >$ is the explored path of tc, and tc_coveredobj is the covered objectives of tc. In order to reduce the length of tc, π is shortened to a path from the initial state to the state where all Def_Use rules of tc_coveredobj are visited; the remainder is ignored.

5. Evaluation

The BOA-based test generation technique is implemented by java in the GROOVE toolset, and to evaluate its efficiency, it is compared with the search-based testing [57]. As mentioned in the previous sections, our proposed approach explores the state space of the model and checks the selected paths to cover the test objectives. Since, one of the state-of-the-art approaches is the model checking based testing (MCT) [58], the proposed method is compared with this approach. These approaches already are implemented in the GROOVE toolset. To make the comparison fair enough, the same fitness function that is reported for these approaches is considered. To show that the results of the BOA based approach are significantly different from the others, the results are evaluated by the Wilcoxon signed-rank test. Wilcoxon signed-rank test [63] is a non-parametric statistical hypothesis test used to compare two related samples. In this test, if the sig. is less than 0.05, it can be concluded that there is a significant difference between the two related samples.

Table 2 shows the initial parameters along with their values to execute the BOA-based approach. The learning and sampling rate determines the percentage of selected promising solutions to learn BN and sample candidate solutions in each iteration, respectively. The maximum length of test case is L = 50. The time limit is set to 30 minutes and results are computed as an average of 10 independent runs of each approach. The experiments were run on a PC with an Intel® Core ™ 2 Duo 2 GHz CPU and 2 GB Memory.

Table 2. The initial parameters for executing the BOA

Approach	Parameters	Value
BOA	Learning rate	0.4
	Sampling rate	0.5
	Population size	20

5.1. Case studies

To evaluate the efficiency, the obtained results on three case studies are compared. These case studies are the models of the online shopping system [59], the bug tracker system [60], and the travel agency system [61]. These systems are modeled through graph transformation using the GROOVE toolset.

Online shopping system (OSS): An online shopping system is a system for purchasing the products electronically over the internet. This system allows a customer to directly view products and select goods, submit an order to the seller, and pay a bill using a credit card.

Travel agency system (TAS): A Travel agency system (TAS) is the process of selling services to the customers. This system provides services related to airlines, hotels, railways, tours, and timetables.

Bug tracking system (BTS): A bug tracking system (BTS) is a software application which tracks the history of software bugs in the software development lifecycle.

Table 3 shows the specifications of the selected case studies including the number of rules and the number of test objectives for each criterion. We refer to the Create-Read, Create-Delete, Create-Update, and dependencies as C1, C2, C3, and C4, respectively.

Table 3. Specifications of the selected case studies

Case study	Rules#	Test objectives#			
		Create_Read C1	Create_Delete C2	Create_Update C3	Dependencies C4
Online shopping system	19	28	7	12	47
Bug tracker system	32	73	13	5	91
Travel agency system	43	66	10	10	86

5.2. Experimental results

Table 4 shows the results of the BOA-based test generation approach in the OSS, BTS, and TAS. In this table, the first column is the used BOA, the second column is the selected criterion, and the others are dedicated to the median, mean and standard deviation of the achieved coverage in the case studies. To show that the BOA-based approach is more efficient than the others, it is compared to the Random Testing (RT). Random Testing is a technique that uses Random Search to select test cases. To study the effectiveness of the proposed approach, the Random Search from [62] is used. The best results are shown in bold.

Figure 7 shows the achieved coverage in the dependencies criterion for the OSS, TAS, and BTS. According to the reported results in Table 4, cBOA is able to completely cover the test objectives for the OSS, TAS, and BTS. tpBOA obtains 100% coverage in the Create_Update criterion for the OSS and BTS, and in all criteria for TAS. nBOA obtains 100% coverage in the Create_Update and Create_Read criterion for TAS, and in Create_Update criterion for BTS. Also, RT obtains 100% coverage in the Create_Update criterion for BTS. For the TAS case study, the other results can be sorted as follows: tpBOA (100%), nBOA (75%), and RT (69%). For the OSS case study, tpBOA (94%) is the second, RT (85%) is the third, and nBOA (70%) is the last. Also, for the BTS case study, other results is as follows: tpBOA (92%), nBOA (91%), and RT (75%).

Table 4 demonstrates that the cBOA is better than the RT and two others in terms of coverage. Therefore, to show that the cBOA outperforms the others, the achieved average coverage is evaluated by the Wilcoxon signed-rank test. The results of this test are given in Table 5. The ineffective or equal cases are shown in bold. As can be seen in this table, the sig. is less than 0.05 in the 25 cases, and in the remaining ones, it equals 1; and there is no statistical difference between the cBOA and others in terms of coverage. So, it can be concluded that the average coverage of the cBOA is significantly different from the others. It is obvious that in all case studies, cBOA generates better results than RT. In other words, cBOA improves the coverage of RT per case study as follows: OSS (15%), BTS (25%), and TAS (31%).

Table 6 shows the running results of the cBOA such as the average coverage and test suite size in the Dependencies criterion for the large models of OSS, TAS, and BTS. For these large models, we performed Wilcoxon signed-rank test on the results of the cBOA against RT. The results are shown in Table 7. As can be seen, the sig for the coverage is less than 0.05 in all cases. Also, the cBOA generates better results than RT in every case.

Table 4. Comparison of the coverage achieved by the BOA-based approach and RT

Approach	Criterion	Case I: OSS			Case II: BTS			Case III: TAS		
		Median	Mean	Deviation	Median	Mean	Deviation	Median	Mean	Deviation
cBOA	C1	**100**	**100**	**0**	**100**	**100**	**0**	**100**	**100**	**0**
	C2	**100**	**100**	**0**	**100**	**100**	**0**	**100**	**100**	**0**
	C3	**100**	**100**	**0**	**100**	**100**	**0**	**100**	**100**	**0**
	C4	**100**	**100**	**0**	**100**	**100**	**0**	**100**	**100**	**0**
tpBOA	C1	96	95.4	1.2649	93	94.8	3.7058	**100**	**100**	**0**
	C2	77	77	0	92	94.4	3.8643	**100**	**100**	**0**
	C3	**100**	**100**	**0**	**100**	**100**	**0**	**100**	**100**	**0**
	C4	94	92.8	1.9321	92	92.5	1.7159	**100**	**100**	**0**
nBOA	C1	72	73.4	14.9383	93	94.4	2.9514	**100**	**100**	**0**
	C2	55	54	24.0159	84	85	7.0047	60	70	14.1421
	C3	70.5	75.3	11.4119	**100**	**100**	**0**	**100**	**100**	**0**
	C4	70	66.4	9.0823	91	89	4.7714	75	76	8.7787
RT	C1	85	86.3	8.6545	78.5	80.5	8.6827	91	92	7.7316
	C2	77	73.9	5.6853	78	73.5	14.9759	86.5	85.7	9.5574
	C3	92	85.2	30.0178	**100**	**100**	**0**	87.5	85.7	14.4918
	C4	85	83.5	9.2286	75	75.5	12.6037	69	67.3	2.5841

Table 5. The results of the Wilcoxon signed-rank test
(**Z:** z-score is the signed number of standard deviations by which the value of an observation is above the mean value of what is being observed or measured. [a] Based on negative ranks.)

Approach	Criterion	Case I: OSS		Case II: BTS		Case III: TAS	
		z	Asymp. Sig. (2-tailed)	z	Asymp. Sig. (2-tailed)	z	Asymp. Sig. (2-tailed)
cBOA- tpBOA	C1	-2.972[a]	0.003	-2.530[a]	0.011	**0.000[a]**	**1.000**
	C2	-3.162[a]	0.002	-2.646[a]	0.008	**0.000[a]**	**1.000**
	C3	**0.000[a]**	**1.000**	**0.000[a]**	**1.000**	**0.000[a]**	**1.000**
	C4	-2.919[a]	0.004	-2.844[a]	0.004	**0.000[a]**	**1.000**
cBOA - nBOA	C1	-2.810[a]	0.005	-2.828[a]	0.005	**0.000[a]**	**1.000**
	C2	-2.836[a]	0.005	-2.754[a]	0.006	-2.762[a]	0.006
	C3	-2.850[a]	0.004	**0.000[a]**	**1.000**	**0.000[a]**	**1.000**
	C4	-2.840[a]	0.005	-2.820[a]	0.005	-2.803[a]	0.005
cBOA - RT	C1	-2.818[a]	0.005	-2.805[a]	0.005	-2.207[a]	0.027
	C2	-2.913[a]	0.004	-2.807[a]	0.005	-2.810[a]	0.005
	C3	-2.214[a]	0.027	**0.000[a]**	**1.000**	-2.201[a]	0.028
	C4	-2.814[a]	0.005	-2.809[a]	0.005	-2.871[a]	0.004

Table 6. Running details of the cBOA approach in the selected case studies

Case study	Host#	Coverage			Test case#			Test suite length		
		Median	Mean	Deviation	Median	Mean	Deviation	Median	Mean	Deviation
OSS	5	90	90.4	3.5024	5	4.8	1.0328	124	125.9	24.3011
	10	86	86	3.6515	5	4.8	1.1353	122	115.4	26.6091
	20	68	65.4	11.3940	5	4.9	0.7379	119	114.8	15.2447
	25	66	67.5	6.4507	5	5.4	0.5164	121	123.7	11.4120
BTS	5	91	87.7	5.4171	7.5	7.8	1.3984	298.5	308.3	54.2198
	7	87.5	86.5	5.6618	8.5	8.9	1.1005	366.5	367.4	53.1961
	10	81.5	80.6	3.5653	9	8.9	1.1972	353	350	46.0156
	15	78	77.8	1.9322	9.5	9.9	1.9692	354	375.3	73.2424
TAS	4	100	99.2	1.9322	7	6.6	0.6992	277.5	265.7	26.7542
	9	90.5	88.9	5.5066	7.5	8	1.3333	296	313.7	53.6926
	12	86	86.4	5.0596	10	10	0.6667	391	386.6	23.3057
	20	73	71.9	4.6774	9.5	9.4	0.9661	350	353.5	31.1956

Table 7. Comparison of running results of the cBOA approach against RT for generating test suite

Case study	Host#	Coverage				Test suite length			
		cBOA (Median)	RS (Median)	z	Asymp. Sig. (2-tailed)	cBOA (Median)	RS (Median)	z	Asymp. Sig. (2-tailed)
OSS	5	90	82	-2.347[a]	0.019	124	304	-2.803[a]	0.005
	10	86	80	-2.148[a]	0.032	122	298.5	-2.807[a]	0.005
	20	68	60	-2.431[a]	0.015	119	317	-2.805[a]	0.005
	25	67	58	-2.040[a]	0.041	121	263.5	-2.803[a]	0.005
BTS	5	91	70	-2.666[a]	0.008	298.5	722	-2.805[a]	0.005
	7	87.5	68.5	-2.499[a]	0.012	366.5	796	-2.803[a]	0.005
	10	81.5	71	-2.524[a]	0.012	353	826	-2.805[a]	0.005
	15	78	69	-2.809[a]	0.005	354	867.5	-2.803[a]	0.005
TAS	4	100	67	-2.803[a]	0.005	277.5	679.5	-2.805[a]	0.005
	9	90.5	67	-2.805[a]	0.005	296	781	-2.803[a]	0.005
	12	86	68	-2.701[a]	0.007	391	709.5	-2.803[a]	0.005
	20	73	65	-2.809[a]	0.005	350	805.5	-2.803[a]	0.005

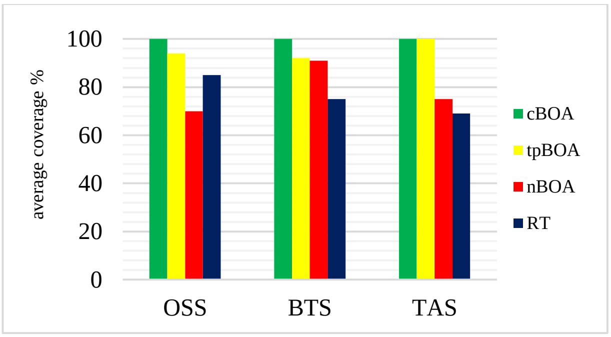

Figure 7. The coverage achieved by the BOA-based approach and RT

Tables 8, 9, and 10 display the average test suite size of the RT and BOA-based approaches. In these tables, the first column is the selected approach, and the second one is the coverage criterion. The third, fourth, and fifth columns provide the number of test cases, test case length, and test suite size, respectively. According to Table 4, the cBOA is the best proposed approach, because it is able to completely cover the test objectives for the case studies OSS, TAS, and BTS. Therefore, to find a significant difference between the cBOA against RT and two others for the cases in which there is not a considerable difference in coverage (based on Table 5, the sig. equals 1.000 and z equals 0.000[a]), the running results of the test suite size are evaluated by Wilcoxon signed-rank test. The outcomes of this test are given in Table 11. The ineffective or equal cases are shown in bold. According to Table 11, it can be concluded that there is no significant difference between the cBOA and others in terms of test suite size. The results are reported by the box-plot in Figure 8. The BOA-based approach generates test suites with a smaller size than RT, in accordance with tables 8 to 10 (the same method is exploited to minimize the RT test suite).

Table 8. Comparison of the size of the generated test suite by the BOA-based and RT approaches for the OSS case study

Approach	Criterion	Test case#			Test case length			Test suite length		
		Median	Mean	Deviation	Median	Mean	Deviation	Median	Mean	Deviation
cBOA	C1	3	3.1	0.5676	25	25	0	75	77.5	14.1912
	C2	2	2.4	0.5164	23	22.6	2.1705	50	54	11.4988
	C3	2	1.8	0.6325	30	29.9	0.3162	60	53.7	18.3548
	C4	3	3	0.6667	25	25	0	75	75	16.6667
tpBOA	C1	3	2.9	0.5676	25	25	0	75	72.5	14.1912
	C2	2	2.4	0.5164	23	22.75	1.3794	49	54.4	11.0172
	C3	2	1.7	0.4830	29.75	29.25	1.0341	58	49.7	14.1739
	C4	3	3.1	0.8756	25	25	0	75	77.5	21.8899
nBOA	C1	2.5	2.3	0.8233	25	24.7	0.9487	62.5	57.2	21.1229
	C2	2	1.8	0.6325	23.5	23.1	1.9692	45	41.7	15.5995
	C3	1	1.2	0.4216	29	28.9	0.8756	29	34.7	12.3112
	C4	2	2.1	0.7397	25	25	0	50	52.5	18.4466
RT	C1	8	8	1.2472	21.8	21.4	1.9406	169.5	167.6	20.3208
	C2	4	3.7	0.4830	18.3	18.7	1.9381	69.5	68.5	5.8737
	C3	5.5	5.6	1.2649	27.2	26.9	1.8848	144	149.1	29.4711
	C4	7	7.5	1.2693	20.7	20.9	1.2429	143.5	156.7	28.7211

Table 9. Comparison of the size of the generated test suite by the BOA-based and RT approaches for the BTS case study

Approach	Criterion	Test case#			Test case length			Test suite length		
		Median	Mean	Deviation	Median	Mean	Deviation	Median	Mean	Deviation
cBOA	C1	4	5	1.6330	50	50	0	200	250	81.6497
	C2	4.5	4.4	0.9661	41	39.2	5.8080	166	169	35.5084
	C3	3	3.2	0.6325	11.5	11.4	1.6465	36	36	6.5490
	C4	6	5.8	0.4216	40	40	0	240	232	16.8655
tpBOA	C1	4	3.9	0.8756	50	50	0	200	195	43.7798
	C2	4	3.8	0.6325	41.25	41.34	1.5890	161	157.05	27.1364
	C3	3	3.3	0.4830	11	11	1.7638	34.5	35.8	4.8944
	C4	7	6.4	1.2649	40	40	0	280	256	50.5964
nBOA	C1	4	3.8	0.4216	50	50	0	200	190	21.0819
	C2	4	3.9	0.3162	41	41.3	1.8288	164	161.1	15.2494
	C3	3	2.7	0.4830	12.5	12.2	0.9189	33	32.7	5.1218
	C4	5.5	5.7	1.0593	40	40	0	220	228	42.3740
RT	C1	11	10.8	2.4404	42	41.6	3.4774	457.5	450.4	93.7517
	C2	7.5	7.4	0.6992	35.1	34.7	3.8846	253.5	255.9	31.0821
	C3	4	4.2	0.6325	10	9.7	1.8949	39.5	40.4	9.3714
	C4	13	14.2	1.9322	34.1	33.5	2.1834	452.5	473.5	65.5680

Table 10. Comparison of the size of the generated test suite by the BOA-based and RT approaches for the TAS case study

Approach	Criterion	Test case#			Test case length			Test suite length		
		Median	Mean	Deviation	Median	Mean	Deviation	Median	Mean	Deviation
cBOA	C1	6	5.6	1.3499	40	40	0	240	224	53.9959
	C2	3	2.9	0.7379	27	26.3	1.9465	79.5	75.9	18.5200
	C3	4	3.7	0.8233	27.5	27.3	1.9465	110	100.7	22.4700
	C4	6	5.9	0.7379	40	40	0	240	236	29.5146
tpBOA	C1	6	5.5	0.7071	40	40	0	240	220	28.2843
	C2	3	2.9	0.5676	26	26.73	1.9195	78	77.29	14.5693
	C3	3	3	0.6667	27.45	26.98	1.8091	79.95	80.81	17.1942
	C4	6	5.9	1.1005	40	40	0	240	236	44.0202
nBOA	C1	6	5.6	0.8433	40	40	0	240	224	33.7310
	C2	2	2.1	0.5676	27	25.6	3.1693	54	54.3	18.0373
	C3	3.5	3.4	0.6992	27.5	27.4	1.4298	94	93.6	21.7419
	C4	3	2.9	0.8756	40	40	0	120	116	35.0238
RT	C1	12	11.9	2.2336	30	29.7	2.5382	352.5	351.3	58.6308
	C2	5	5.4	0.5164	20.4	20.5	1.8667	111.5	110.6	14.8189
	C3	8	7.8	1.0328	21.8	21.9	2.0339	168	170.4	24.8202
	C4	14	14.3	1.5670	25.4	24.9	3.1282	342	359	66.0858

Table 11. The results of the Wilcoxon signed-rank test
(**Z:** z-score is the signed number of standard deviations by which the value of an observation is above the mean value of what is being observed or measured. [a] Based on negative ranks.)

Case study	Criterion	Approach	Test suite length	
			z	Asymp. Sig. (2-tailed)
OSS	C3	tpBOA	-0.658[a]	0.511
BTS	C3	tpBOA	-0.422[a]	0.673
	C3	nBOA	-0.611[a]	0.541
	C3	RT	-0.918[a]	0.359
TAS	C1	tpBOA	-0.345[a]	0.730
	C2	tpBOA	-0.459[a]	0.646
	C3	tpBOA	-1.383[a]	0.066
	C4	tpBOA	**0.000[a]**	**1.000**
	C1	nBOA	**0.000[a]**	**1.000**
	C3	nBOA	-1.067[a]	0.286

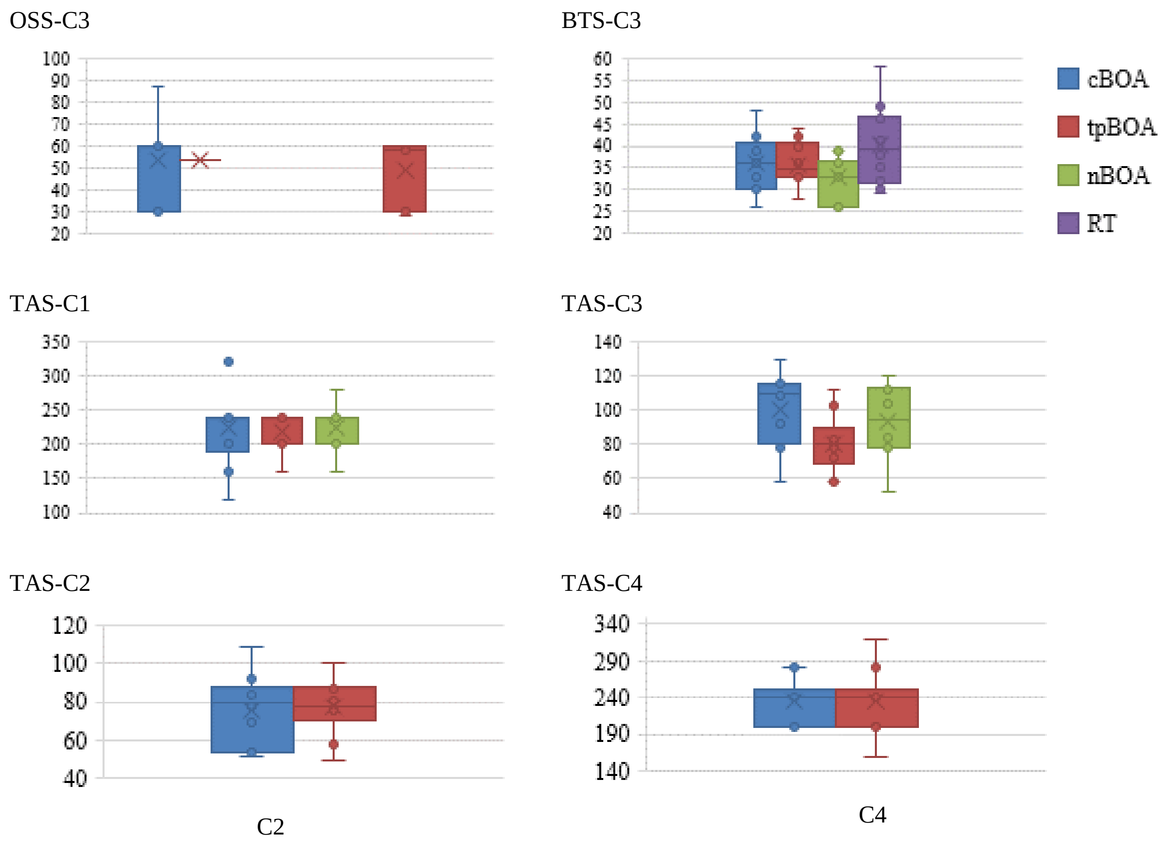

Figure 8. The box plots of the average test suite size for the cases with the same coverage

5.3. Comparison with other testing techniques

In this section, to evaluate the efficiency of the BOA-based approach, it is compared with the Model checking- based test generation and search-based testing in terms of the achieved coverage.

5.3.1. Model checking-based test generation approach (MCT): In the MCT technique, a model of the system and a set of test objectives as trap/reachability properties is provided to the model checker. The model checker detects a counterexample/witness path to violate/satisfy the property. A counterexample/witness is a path from the initial state to a state that the property is refused/verified. These paths can be used as TCs.

In [58], the authors proposed an automatic test case generation approach for data flow testing using model checking. In this approach, test objectives are extracted from the program source code. To compare our approach with the MCT, the MCT is implemented in the GROOVE to make the comparison fair enough. In this implemented MCT, test objectives are extracted dynamically from the model and they are expressed as a set of trap properties. For every test objective that is not covered, equivalent trap property is provided to the model checker, and the state space is verified for it. If the given property is satisfied, the model checker generates a counterexample. Afterward, this counterexample is added to the test suite if it is a def-clear path. On the other hand, if the property is not satisfied or state space explosion occurs, the current test objective is marked as untestable. The strategy to search counterexample is the Breadth-first Search (BFS). This process is repeated until the termination criteria such as the satisfaction of the properties, state space explosion, or reaching the time limit of 30 minutes occurs.

Table 12 reports the results of comparing the cBOA with MCT for the host graphs of different sizes. The coverage criterion is C4, and the other execution settings are the same as the cBOA. As shown in this table, in all cases, MCT fails to generate the test suite due to the running out of memory. The results confirm that the proposed method is more scalable than MCT because as the state space is getting larger, MCT fails to explore it due to the exponential memory usage.

Table 12. Comparison of the coverage achieved by the cBOA and MCT

Case study	Host#	Test generation strategies			
		MCT	cBOA		
			Median	Mean	Deviation
OSS	1	Out of memory	100	100	0
	5	Out of memory	90	90.4	3.5024
	10	Out of memory	86	86	3.6515
	20	Out of memory	68	65.4	11.3940
	25	Out of memory	66	67.5	6.4507
BTS	1	Out of memory	100	100	0
	5	Out of memory	91	87.7	5.4171
	7	Out of memory	87.5	86.5	5.6618
	10	Out of memory	81.5	80.6	3.5653
	15	Out of memory	78	77.8	1.9322
TAS	1	Out of memory	100	100	0
	4	Out of memory	100	99.2	1.9322
	9	Out of memory	90.5	88.9	5.5066
	12	Out of memory	86	86.4	5.0596
	20	Out of memory	73	71.9	4.6774

5.3.2. Search-based testing: The recent work related to our proposed approach is the search-based testing [57]. In this work, a search-based method is proposed for integration testing of systems specified through GTS. To meet the challenges of the MCT approach, the proposed method employs several heuristics like Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Bat Algorithm (BA), Gravitational Search Algorithm (GSA), and a hybrid algorithm using GA and PSO (HGAPSO) to generate TCs using the GROOVE toolset. Table 13 shows the initial parameters along with their values to execute the search algorithms. It should be noted that these values are suitable value for these parameters which are reported in [57]. In this work, test objectives are extracted dynamically from the model, and each TC is a path within the state space. In different generations, if a path is a def-clear path, the sequence is added to the test suite. Experimental results show that this test generation method has significantly higher coverage than MCT and model-based testing using visual contracts. The results confirm that the hybrid algorithm is faster and more scalable than the others.
Table 14 compares the achieved coverage of the proposed approach and search-based testing for the three case studies. Time is considered 30 minutes and the coverage criterion is C4. According to the reported results, the cBOA is able to completely cover the test objectives in the case studies OSS, TAS, and BTS. GA and HGAPSO achieve full coverage in the OSS case study. As shown in Table 14, in other cases, the cBOA has better coverage than others. So, it can be concluded that if a fewer test budget is set, the cBOA achieves better results.

Table 13. The initial parameters for executing search-based testing

Algorithm	parameter	value
	Population size	30
	Maximum of iteration	100
	Maximum length of test	50
	Maximum number of test	15
Genetic Algorithm (GA)	Mutation rate	0.01
	Cross over rate	0.6
Particle Swarm Optimization (PSO)	W	0.8
	C, C ’	0.2
	Maximum velocity	0.2
Bat Algorithm (BA)	Min frequency	0
	Max frequency	100
	Loudness	25
	Plus rate	0.5
Gravitational Search Algorithm (GSA)	G_0	100
	Alpha	2

Table 14. Comparison of coverage achieved by the cBOA and the search-based testing

Approach	Case I: OSS			Case II: BTS			Case III: TAS		
	Median	Mean	Variance	Median	Mean	Variance	Median	Mean	Variance
cBOA	**100**	**100**	**0**	**100**	**100**	**0**	**100**	**100**	**0**
HGAPSO	**100**	**100**	**0**	85.71	87.68	33.75	98.25	96.5	21.65
GA	**100**	**100**	**0**	84.61	85.15	30.89	100	98.25	10
PSO	91.48	91.9	9.85	81.31	82.081	17.7	84.3	84.99	20.85
BA	93.61	93.39	6.49	77.47	77.87	23.22	85.46	85.11	23.38
GSA	91.48	92.33	11.28	77.47	78.56	16.97	87.2	87.2	9.01

Here, for the cases in which no significant difference is found in coverage, a comparison of the convergence speed between the search algorithms and cBOA is presented. The time limit is set to 30 minutes and the coverage criterion is C4. According to Figure 9, the convergence speed for the GA is better than HGAPSO and cBOA; there is not much difference between them. However, if a fewer test budget (time) were set, cBOA would reach to better coverage.

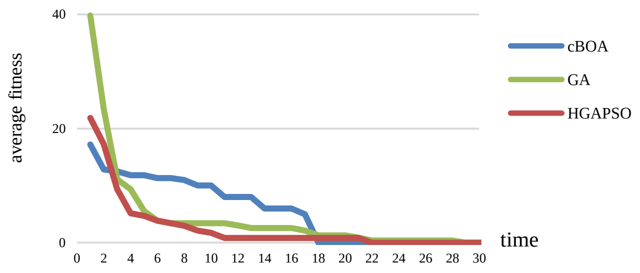

Figure 9. Comparison of the convergence speed

6. Conclusion and future works

The model checking-based test generation approach is a proper technique to automatically generate the executable test cases from the models by exploring the state space. However, the limitation of this technique in exploring paths to satisfy a set of test objectives is the state space and test case explosion. Recently, methods have been proposed using meta-heuristic and evolutionary approaches to manage the state space explosion. Consequently, a portion of the model state space can be explored by the model checker tools to achieve the test objectives.

In this paper, a novel approach is proposed using the Bayesian optimization algorithm and model checker to automatically generate the test cases for the systems specified through graph transformation. The BOA-based test generation technique is implemented by java programming language in the GROOVE toolset, and to evaluate its efficiency, it is compared with the search-based testing, random testing, and model checking based testing. In this approach, the structure of the built BN is a fixed chain and only the parameters are learned.

The experiments show that the cBOA structure outperforms the other two suggested structures in terms of coverage and convergences speed. The advantages of the cBOA over the other methods can be listed as follows:

- if a fewer test budget is set, the cBOA achieves better results than the others.
- it is faster than search-based testing.
- it obtains better results than random testing in terms of coverage and test suite size.
- it obtains better results than search-based testing in terms of coverage.
- it obtains better results than the others in complex systems.
- it explores a fewer number of states than the model checking-based test generation approach

The proposed approach has a limitation. In a complex system maintaining the conditional probability tables requires more memory space.

In this paper, we have assumed that the structure of learning network is fixed. In the literature, there are algorithms can be used to searching for a good Network. As a future research, searching over the networks can be considered in order to maximize the value of a scoring metric.

References

[1] P. Ammann and J. Offutt, "introduction to software testing", Cambridge university press, New York, ISBN-13 978-0-511-39330-3, 2008.

[2] V.M Sumalatha and G.S.V.P Raju, "Model Based Test Case Optimization of UML Activity Diagram using Evolutionary Algorithms", International Journal of Computer Science and Mobile Application, Vol.2, Issue.11, 2014, pp.131- 142.

[3] M. Utting, B. Legerad, F. Bouquet, E. Fourneret, F. Peureux and A. Vernotte, "Recent Advances in Model-Based Testing," in Advances in Computers, Vol. 101, A. Memon, Ed., Elsevier, 2016, pp. 53-120.

[4] M. Utting, A. Pretschner and B. Legeard, "A taxonomy of model-based testing approaches", Software Testing, Verification and Reliability, 2011, published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/stvr.456.

[5] S. Colin, B. Legeard, F. Peureux, "Preamble computation in automated test case generation using constraint logic programming", Journal of Software Testing, Verification and Reliability, Vol.14, Issue.3, 2004, pp. 213–235.

[6] J. Dick and A. Faivre, " Automating the generation and sequencing of test cases from model-based specifications", Proceedings of the 1st International Symposium of Formal Methods Europe, Odense, Denmark, vol. 670, 1993, pp. 268–284.

[7] S. Shamshiri, J. Rojas, G. Fraser and P. McMinn, "Random or Genetic Algorithm Search for Object-Oriented Test Suite Generation?", in Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, New York, NY, USA, 2015.

[8] P. McMinn," Search-based software test data generation: A survey", Journal of Software Testing, Verification and Reliability, vol.14, Issue. 2, 2004, pp. 105–156.

[9] S. Ali, M. Iqbal, A. Arcuri and L. Briand, "A search-based OCL constraint solver for model-based test data generation," in Proceedings of International Conference on Quality Software, Madrid, 2011.

[10] D. Clarke, T. Jéron, V. Rusu, E. Zinovieva, "A symbolic test generation tool. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)", Lecture Notes Computer Science, vol. 2280, Springer: Berlin,2002, pp. 470–475.

[11] M. Schnelte and B. Güldali, "Test Case Generation for Visual Contracts Using AI Planning", conformance Informatik 2010: Service Science – Neue Perspektiven für die Informatik, Beiträge der 40. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Band 2, 27.09. 2010, Leipzig.

[12] S. Mohalik, A. Gadkari, A. Yeolekar, K. Shashidhar and S. Ramesh, "Automatic test case generation from Simulink/State flow models using model checking," Journal of Software Testing, Verification & Reliability, vol. 24, no. 2, 2014, pp. 155–180.
[13] A. Offutt, S. Liu, A. Abdurazik and P. Ammann, "Generating test data from state-based specifications", Journal of Software Testing, Verification and Reliability, vol.13, Issue.1, 2003, pp. 25-53.
[14] C. Baier, J.P Katoen, "Principles of Model Checking", The MIT Press, Cambridge, ISBN-978-0-262-02649-9, 2008.
[15] H. Hong, I. Lee, O Sokolsky and H. Ural, " A temporal logic based theory of test coverage and generation", Proceedings of the TACAS’02, 2002, pp. 327–341.
[16] R. Yousefian, V. Rafe and M Rahmani, " A heuristic solution for model checking graph transformation systems”, Applied Soft Computing, Vol.24, Issue. C, 2014, pp. 169-180.
[17] V. Rafe, M. Moradi, R. Yousefian, A. Nikanjam, " A meta-heuristic approach for automated refutation of complex software systems specified through graph transformations", Applied Soft Computing, Vol. 33, 2015, pp. 136–149.
[18] G. Francesca, A. Santone, G. Vaglini and M.L. Villani, "Ant colony optimization for deadlock detection in concurrent systems", Journal of Computer Software and Applications Conference (COMPSAC). IEEE, 2011, pp. 108–117.
[19] E. Pira, V. Rafe, A. Nikanjam," Deadlock detection in complex software systems specified through graph transformation using Bayesian optimization algorithm”, The Journal of Systems and Software, No.131, 2017, pp.181-200.
[20] M. Pelikan, "Probabilistic Model-Building Genetic Algorithms", Hierarchical Bayesian Optimization Algorithm, Studies in Fuzziness and Soft Computing, 170, Springer Berlin Heidelberg, 2005, pp. 13–30.
[21] Pelikan, Martin, David E. Goldberg, and Erick Cantú-Paz, "BOA: The Bayesian optimization algorithm." Proceedings of the genetic and evolutionary computation conference GECCO-99. Vol. 1. 1999.
[22] J. Pearl, "Probabilistic Reasoning in Intelligent systems: Networks of Plausible Inference", 2014. Morgan Kaufmann.
[23] R. Heckel, "Graph transformation in a Nutshell", Electr. Notes Theor. Comput.Sci. (ENTCS), vol. 148, issue. 1, 2006, pp. 187–198.
[24] H. Kastenberg and A Rensink, "Model Checking Dynamic States in GROOVE", International SPIN Workshop on Model Checking of Software, Springer, Berlin Heidelberg, 2006, pg. 299–305.
[25] H. Ehrig, G. Engels, F. Presicce and G. Rozenberg, "Graph Transformations," in Second International Conference on Graph Transformation, Rome, Italy, 2004.
[26] G. Taentzer, 'AGG: a graph transformation environment for modeling and validation of software", In: International Workshop on Applications of Graph Transformations with Industrial Relevance. Springer, Berlin Heidelberg, 2003, pp. 446–453.
[27] D. Varro and A. Balogh, "The model transformation language of the VIATRA2framework", Sci. Comput. Program, Vol. 68, Issue. 3, 2007, pp. 214–234.
[28] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri, "NUSMV: A New Symbolic Model Verifier", In CAV ’99: Proceedings of the 11th International Conference on Computer Aided Verification, London, UK,1999, pp. 495–499.
[29] P. Larranaga, J.A. Lozano, "Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation", Springer Science & Business Media, 2001.
[30] M. Mlynarski, "Model-Based Testing: Achievements and Future Challenges", University of Paderborn, s-lab – Software Quality Lab, Paderborn, Berlin, Germany.
[31] R. Heckel, T. Ahmed Khan and R. Machado, "Towards Test Coverage Criteria for Visual Contracts," in Proceedings of the Tenth International Workshop on Graph Transformation and Visual Modeling Techniques, Berlin, 2011.
[32] M. Badri, L. Badri and M. Naha, "A use case driven testing process: towards a formal approach based on UML collaboration diagrams", Proceedings of the 3rd International Workshop on Formal Approaches to Testing of Software FATES 2003, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Heidelberg, pp. 223–235.

[33] S. Ali, L.C. Briand, M.J Rehman, H. Asghar, M. Zohaib Z. Iqbal and A. Nadeem, "A state-based approach to integration testing based on UML models", Information and Software Technology, vol.49, 2007, pp.1087–1106.
[34] M.E. Vieira, M.S. Dias and D.J. Richardson, "Object-oriented specification-based testing using UML state-chart diagrams", in: Proceedings of the Workshop on Automated Program Analysis, Testing, and Verification (at ICSE'2000), June 2000.
[35] N. Khurana, R.S Chillar, " test Case Generation and Optimization using UML Models and Genetic Algorithm", Procedia Computer Science, vol. 57, 2015, pp. 996 – 1004.
[36] A. Abdurazik and J. Offutt, "Using UML collaboration diagrams for static checking and test generation", in: Proceedings of the Third International Conference on the Unified Modeling Language (UML'00), York, UK, October 2000, pp. 383–395.
[37] S. Gnesi, D. Latella and M. Massink, "Formal test-case generation for UML state charts", Proceedings of the 9th IEEE International Conference on Engineering Complex Computer Systems (ICECCS'04), 2004, pp. 75–84.
[38] A.S.M. Sajeev and B. Wibowo, " UML modeling for regression testing of component based systems", Electronic Notes Theoretical Computer Science, vol. 82, issue.6, 2003, pp. 1–9.
[39] L. Gallagher and J. Offutt, "Integration Testing of Object-oriented Components Using FSMS: Theory and Experimental Details", GMU Technical Report ISE-TR-04-04, July 2004.
[40] L. Gallagher, J. Offutt and A. Cincotta, "Integration testing of object-oriented components using finite state machines", software testing, verification and reliability, Test. Verif. Reliab. Vol. 16, 2006, pp. 215–266.
[41] L. Gönczy, R. Heckel and D. Varró, " Model-Based Testing of Service Infrastructure Components", In Testing of Software and Communicating Systems, 19th IFIPTC6/WG6.1 International Conference, TestCom 2007, 7th International Workshop, FATES 2007, Tallinn, Estonia, June 26-29, 2007, Proceedings. Lecture Notes in Computer Science 4581, pp. 155–170.
[42] O. Runge, T. Ahmed Khan and R. Heckel, "Test Case Generation Using Visual Contracts", Electronic Communications of the EASST, Vol.58, 2013, Proceedings of the 12th International Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT 2013).
[43] B. Güldali, M. Mlynarski, A. Wübbeke and G. Engels, " Model-Based System Testing Using Visual Contracts", In 35th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2009, Patras, Greece, August 27-29, 2009, Proceedings. Pp. 121–124. IEEE Computer Society, 2009.
[44] X. Peng and L. Lu," A New Approach for Session-based Test Case Generation by GA", IEEE 3rd International Conference on Communication Software and Networks, 2011.
[45] V.M. Sumalatha and G.S.V.P. Raju, "An Model Based Test Case Generation Technique Using Genetic Algorithms", The International Journal of Computer Science & Applications (TIJCSA), Vol. 1, No. 9, 2012, pp.46-57.
[46] J Wegener, A Baresel, "Evolutionary test environment for automatic structural testing", Information and Software Technology, Vol.43, Issue.14, 2001, pp.841-854.
[47] R. Sagarna, J.A. Lozano," Scatter Search in software testing, comparison and collaboration with Estimation of Distribution Algorithms", European Journal of Operational Research, Vol. 169, 2006, pp. 392–412.
[48] R. Sagarna, A. Arcuri and X. Yao, "Estimation of Distribution Algorithms for Testing Object Oriented Software", IEEE Congress on Evolutionary Computation,2007.
[49] Z. Fang and H. Sun. "A software regression testing strategy based on Bayesian network", International Conference on Computational Intelligence and Software Engineering (CiSE), IEEE, 2010.
[50] Z. Yang, Z. Yu, and C. Bai, "The approach of graphical user interface testing guided by Bayesian model", Lecture Notes in Electrical Engineering 2014, pp.385–393.
[51] A.S. Andreou and S.P. Chatzis, "software defect prediction using doubly stochastic Poisson processes driven by stochastic belief networks", The Journal of Systems and Software, Vol.122,2016, pp. 72-82.
[52] S. Wagner, " A Bayesian network approach to assess and predict software quality using activity-based quality models", Information and Software Technology, vol. 52, 2010, pp. 1230–1241.
[53] C.G Bai, "Bayesian network based software reliability prediction with an operational profile", The Journal of Systems and Software, vol.77, 2005, pp. 103–112.

[54] C.G. Bai, Q.P. Hu, M. Xie and S.H. Ng, "Software failure prediction based on a Markov Bayesian network mode", The Journal of Systems and Software, vol. 74, 2005, pp. 275–282.

[55] A. Nikanjam, A. Rahmani, "Exploiting bivariate dependencies to speedup structure learning in Bayesian optimization algorithm", J. Comput. Sci. Tech. vol. 27, no. 5, 2012, pp. 1077–1090.

[56] K. Jebari, M. Madiafi," Selection Methods for Genetic Algorithms", International Journal of Emerging Sciences, vol. 3, no. 4, 2013, pp. 333-344.

[57] A. Kalaee, V. Rafe, "Model-based test suite generation for graph transformation system using model simulation and search-based techniques", Information and Software Technology, vol.108, 2019, pp.1-29.

[58] S. Ting, K. Wu, M. Weikai, P. Geguang, H. Jifeng, C. Yuting, S. Zhendong, "A survey on data-flow testing", ACM Comput. Surv. vol. 50 (1) (2017) 1–35.

[59] G. Engels, B. G¨uldali, M. Lohmann, "Towards model-driven unit testing", International Conference on Models in Software Engineering Berlin, Heidelberg, 2006.

[60] O. Runge, T. Khan, R. Heckel, "Test case generation using visual contracts", in: Proceedings of the 12th International Workshop on Graph Transformation and Visual Modeling Techniques (ECEASST), 2013.

[61] V. Rafe, "Scenario-driven analysis of systems specified through graph transformations", J. Visual Lang. Comput., vol. 24 (2) (2013) 136–145.

[62] A. Arcuri, L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering", in: Proceedings of the 33rd International Conference on Software Engineering, New York, NY, USA, 2011.

[63] Wilcoxon. F, "Individual comparisons by ranking methods", Biom. Bull. Vol. 1 (6), 1945, pp. 80–83.