Journal of AI and Data Mining
Vol 8, No 2, 2020, 189-199.
DOI: 10.22044/JADM.2019.7564.1900

A Hybrid Meta-heuristic Approach to Cope with State Space Explosion in Model Checking Technique for Deadlock Freeness

N. Rezaee and H. Momeni*

Department of Computer Engineering, Golestan University ,Gorgan, Iran.

Received 13 October 2018; Revised 31 May 2019; Accepted 01 December 2019
**Corresponding author: h.momeni@gu.ac.ir (H.Momeni).*

Abstract
Model checking is an automatic technique for software verification through which all the reachable states are generated from an initial state in order to find the errors and desirable patterns. In the model checking approach, the behavior and structure of the system should be modeled. The graph transformation system is a graphical formal modeling language used to specify and model the system. However, modeling large systems with the graph transformation system suffers from the state space explosion problem, which usually requires huge amounts of computational resources.
In this paper, we propose a hybrid meta-heuristic approach in order to deal with this searching problem in the graph transformation system because meta-heuristic algorithms are efficient solutions to traverse the graph of large systems. Using the artificial bee colony and simulated annealing, our approach replaces a full state space generation, only by producing part of it checking the safety, and finding the errors (e.g. deadlock). The experimental results obtained show that the proposed approach is more efficient and accurate compared with other approaches.

Keywords: *Software Verification, Model Checking, State Space Explosion, Meta-heuristic Approaches, Graph Transformation System.*

1. Introduction

Model checking is an automated technique used for the verification of concurrent finite state systems, through which the behavior of the system is analyzed based on the time characteristics by searching for a state graph [1-4]. The graph transformation system (GTS) is a suitable visual modeling language used to formally describe and verify the system behavior [2]. GTS describes the properties of systems including the behavioral and structural properties, and it is widely used for system specification and verification. GTS is determined as a triple (TG, HG, R), whereby TG is the type graph, HG is the host graph, and R is the rules set.
In the model checking-based verification through GTS, the temporal logic properties such as safety are examined by a comprehensive search of the whole state space. Since in a large and complex system the size of the state space expands exponentially with the number of processes, GTS encounters the *state space explosion* that consumes all the memory space that is consumed [5- 7]. When the state space explosion occurs, GTS cannot traverse through the whole graph. In this situation, it has been suggested to use a refutation technique to check the violation of a property instead of its proof so that there is no need to generate the entire state space [8- 10]. In this work, we checked the violation of safety property through deadlock detection.
In the recent years, some meta-heuristic approaches such as Genetic Algorithm (GA), Ant Colony Optimization (ACO), Artificial Bee Colony (ABC), Particle Swarm Optimization (PSO), and Simulated Annealing (SA) have been widely used to cope with state space explosion because these approaches do not search for exhaustive states that, in turn, lead to a reduction in the state space size. In this paper, a hybrid ABC-SA algorithm is proposed to cope with the state space explosion problem in the

refutation technique. As ABC may get trapped into the local optima, SA is an efficient algorithm in local search as well as in escaping from the local optima. The main contribution of our paper is to map the deadlock detection onto a search problem in the graph transformation system. Since the state space to deadlock detection is large for the complex systems, we solved this problem using the ABC-SA algorithm. The other contribution of our paper is to consider the efficiency and accuracy in the refutation technique in GTS.

The proposed approach is implemented in model checker Groove [11], and is verified in terms of deadlock-freeness and correctness.

The rest of this paper is organized as what follows. Section 2 provides a brief background. Section 3 presents some notable related works. Section 4 presents the proposed approach. Section 5 presents our model and the evaluation results. Sections 6 and 7 provide the running details and Wilcoxon signed-rank test for evaluating the results, and finally, Section 8 concludes the paper.

2. Background

2.1 Model checking

In the model checking technique, a model checker requires a model of the system and a set of properties (such as reachability [9], safety [22-27], liveness [22-24, 26-27], and fairness [5, 8]) as inputs to systematically examine whether the given model meets these characteristics or not. This is accomplished by searching for the entire state space of the system to determine whether the current behavior described by the time property is obtained from the system graph state or not [20-21]. The main challenge of the model checking technique is to cope with the state space explosion problem [28-29].

2.2 Artificial bee colony algorithm

ABC algorithm [31] simulates the behavior of real honey bees. There are three groups of bees in the colony of artificial bees including the employed, onlookers, and scouts bees. The population of ABC consists of SND-dimensional vectors of decision variables [32] as (1):

$$X_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,D}\}, \quad i = 1,2,\ldots, SN \tag{1}$$

Each X_i, which is defined by the lower and upper bounds X_{min} and X_{max}, is generated by (2):

$$x_{i,j} = x_{min,j} + rand(0,1)\left(x_{max,j} - x_{min,j}\right) \tag{2}$$

where, i = 1, 2, …, SN, and j = 1, 2, …, D. With respect to each individual X_i, a candidate V_i is produced by adding the scaling difference of two population members to the base individual as (3):

$$v_{i,j} = x_{i,j} + \Phi_{i,j}\left(x_{i,j} - x_{k,j}\right) \tag{3}$$

where, $k \in \{1, 2, \ldots, SN\}$ and $j \in \{1, 2, \ldots, D\}$ are randomly selected indices, k has to be different from i, and $\Phi_{i,j}$ is a random number in the range of [-1, 1]. Then a greedy selection is performed between the old individual X_i and the candidate V_i. The employed bees will share the information received from their searches with onlooker bees. Given the probability value P_i of the corresponding solution, each onlooker chooses a solution by (4):

$$Pi = fit_i \Big/ \sum_{j=1}^{SN} fit_j \tag{4}$$

where, fit_i is proportional to the nectar amount of the food source in the position i, is called the fitness value of the i_{th} solution, and defined by (5):

$$\text{fit}_i = \begin{cases} 1 + abs(\text{f}_i), & \text{f}_i < 0 \\ 1/(1+\text{f}_i), & \text{f}_i \geq 0 \end{cases} \tag{5}$$

where, f_i is the objective function value of solution i that can be computed by (9). Then each onlooker bee generates a new candidate solution by (3), and the greedy selection is performed again. The value of the pre-defined number of cycles is one of the important control parameters for ABC, named *limit*. If X_i is an abandoned individual, then the scout generates a new solution by (2) to replace X_i.

2.3 Simulated annealing algorithm

The simulated annealing is a meta-heuristic algorithm for approximating global optimization in large search spaces [30]. In order to escape from the local optima during exploration of the solution space, the SA algorithm lets the worse neighbor solutions in a controlled manner to be accepted [33]. In each iteration, for a current solution x, a neighbor x' is selected from the neighborhood of x. For each movement, the objective difference Δ is evaluated by (6), where $f(x)$ and $f(x')$ are the objective function values of x and x'; x' could also be accepted with a probability by (7):

$$\Delta = f(x') - f(x) \tag{6}$$

$$Ps = exp(-\Delta/T) \tag{7}$$

where, the probability of acceptance is compared with a number $r \in (0,1)$ generated randomly, and x' is accepted whenever $p > r$ [34]. T is a temperature, which is controlled by a cooling scheme. In a

typical implementation of SA, decreasing the temperature T takes place in each iteration, starting from an initial value T_0 and using an attenuation factor $\beta \grave{o}(0,1)$, which is constant, and typically, $0.75 \le \beta \le 0.95$. In the first iteration, T_0 must be high enough to let a worse solution to be accepted. In each iteration i, the current temperature value is evaluated by (8):

$$T_i = \beta^i T_0 \quad (8)$$

3. Related works

To the best of our knowledge, no work has been done yet on mapping deadlock detection in the large state space to a searching problem in GTS.

The techniques available for coping with state space explosions could be divided into several categories according to how they make and search for their state space. The classic methods first create the entire state space, and then search for all the possible states [35], e.g. depth-first search (DFS) and breadth-first search (BFS) [36]. Several methods have been proposed to reduce the size of the state space, e.g. symbolic verification, partial order reduction, symmetry checking methods, scenario-driven model checking, and abstraction. The other kinds of methods are on-the-fly techniques, which construct the state space on demand dynamically during the model checking operation [37-47, 53]. In the recent years, intelligent methods have been considered in model checking systems. In [10, 48], the use of GA to explore very large state spaces in search of error states has been examined.

In [49], a model has been presented for solving the Travel Salesman Problem (TSP) using ACO. The goal is to guarantee that most of the time the shortest paths are probabilistically taken. In [50], the use of ACO to reduce the state space explosion problem by finding the shortest counter examples has been provided. In [60], the use of ACO to refute the safety property in the concurrent systems has been presented.

In [51], a comparison has been made between five meta-heuristic algorithms for the state space explosion problem including SA, ACO, PSO, and two types of GA, and five other classic search algorithms to solve the problem of finding the properties violation in concurrent Java programs.

Teaching-learning-based optimization (TLBO) is a new type of meta-heuristic approach used in this scope, which finds better or equal solutions much faster than ES, PSO, ABC, etc. [32].

By the same token, in [35], a new ACO optimization model, called ACO-hg, has been presented to violate the safety properties in the concurrent systems. In [54], using this algorithm has been proposed to investigate the violation of the liveness property in the concurrent systems.

Furthermore, in the recent years, a combination of heuristic or meta-heuristic algorithms with one another or with other algorithms has been used to improve the process of finding errors in the model checking technique. In [55], two new algorithms have been utilized to find the deadlock in the complex software systems specified by GTS. The former is a hybrid algorithm that uses PSO and BAT (BAPSO), and the latter is a greedy algorithm to find deadlocks.

Besides, in [2], the PSO algorithm has been used to find the optimal path for a deadlock state. To increase the accuracy, a hybrid algorithm has been proposed using PSO and GSA. Similarly, in [56], symmetry reductions and heuristic search have been combined to detect safety errors in asynchronous systems.

In [61], a data mining-based approach, called EMCDM, has been proposed for detecting deadlocks in models that are specified by GTS.

In [62], a new method based on the Bayesian Optimization Algorithm (BOA) has been proposed to deadlock detection in systems specified through GTS.

In [63], a data mining-based approach has been proposed, where after getting the required knowledge, only a small portion of the state space is explored to refute the desired property. Meta-heuristic algorithms increase the performance and accuracy of search problems[64] and we use these algorithms in our proposed approach.

4. Proposed approach

We now describe our approach to cope with the state space explosion problem in the systems that are specified and modeled by GTS in this section. We used the refutation technique to check to refuse an error or property rather than providing that error or property. This goal can be reached by looking for a path that contains a property violation (called a counter-example).

In our case, we aimed to detect deadlock as a type of safety property. Safety property states that a good event must occur in all states or a bad event (for example, a deadlock) must not occur in any of the states. If we reach a deadlock in the system's state space, we can conclude that the safety property has been violated.

In addition, we used the meta-heuristic algorithms in our approach due to the efficiency of these algorithms for searching for the problems with very large state spaces.

The objective of this paper is to cope with the state space explosion problem in the complex system

specified with GTS to find errors (e.g. deadlock). To detect a deadlock, a set of paths in graph should be traversed. We employed the ABC algorithm to find an optimal path to a deadlock state but since the ABC algorithm, like many other artificial intelligence algorithms, may stick in the local optima, we combined it with the SA algorithm, which is another intelligent algorithm that is good at escaping from the local optima [31]. We present our proposed algorithm, namely ABC-SA, in this section and apply it to the searching strategy in GTS and Groove tool.

4.1 Food sources as problem solutions

In the proposed algorithm, the position of each bee in ABC is considered as a candidate solution. This position is represented by a sequence of numbers that demonstrate a path.

These numbers are generated randomly with a random function, and the values are between 0 and the maximum number of outgoing transitions in each problem. This maximum number relies on the rules of the model that we are checking. In figure 1, a path that is a representative of a food source in the ABC algorithm in a hypothetical state space is shown in a dark blue color. In this example, the position of the food source is <0, 2, 0, 1, 0>.

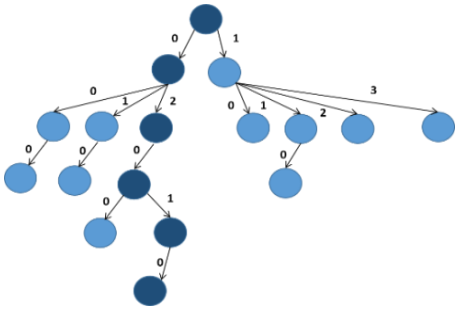

Figure 1. A path in a hypothetical state space.

4.2 Applying ABC-SA in model checking

In the ABC algorithm, by generating new candidate solutions, if the algorithm selects a new solution, provided that it definitely becomes better, it may be trapped into the local optima; therefore, we combine ABC with the SA algorithm as is shown below. SA selects the neighbor that is worse with a specific probability computed by (7). In each iteration, at first, the possibility of accepting a worse solution is higher but since the temperature cooling procedure takes place, it reduces.

The pseudo-code of the ABC-SA algorithm.

1. Create the initial population and initialize the SA parameters; T (initial temperature)
2. Iteration = 1
    3. Send the employed bees to their food source
    4. Calculate probabilities of each food source
    5. Send the onlooker bees to the food sources with higher nectar amounts
    6. Send the scout bees to the search area to find new food sources randomly instead of abandoned food sources
    7. Memorize the best food source up to now
    8. Simulated Annealing search process
    9. Reduce the temperature
    10. Record the global minimum
    11. Calculate the fitness of each food source
12. Iteration = iteration + 1
13. If termination criterion (iteration, temperature, deadlock found), met, return the result
14. Else, go to 3

4.3 Objective function

Our goal was to verify the model and find deadlocks, so we sought for paths through which the outgoing transitions of the states were decreased. We computed the sum of the outgoing transitions in a path as the fitness function. This fitness value was calculated by (9), where Sum_i is the sum of the generated states in a depth of i.

$$F(x) = \sum_{i=1}^{DepthOfSearch} \text{Sumi} \quad (9)$$

5. Modeling and evaluation

In this section, we present a formal model of our approach with GTS, and use the refutation technique to formally verify the soundness of the ABC-SA algorithm for the deadlock detection.

The results obtained are displayed as to whether the desired characteristic is satisfied or not. We considered the number of times that the path was generated to the final state (deadlock) as the "Hit Rate" and the time consumed for the verification as the "Response Time".

The control parameters of the ABC-SA algorithm are presented in table 1. Each experiment was repeated for 30 times, and the average response time (in seconds) was recorded.

In order to test our proposed approach, we used four famous problems that could not be verified by the existing tools due to the state space explosion problem. These problems were dining philosophers [57], Pacman game [58], N-queen [59], and 8-puzzle. We compared the results of the proposed approach with the classical strategies such as BFS, DFS, and A^* in the same machine. We also compared our results with the meta-heuristic-based strategies such as GA, PSO, ABC,PSO+GSA nBOA, and BAPSOS, presented in [10], [2], [61], and [55] with the same parameters and the same machine.

Table 1. Control parameters of the ABC-SA algorithm.

Control parameters of ABC-SA algorithm	
Iteration	100
Swarm size	It is different in different models
Limit	Number of onlooker bees *Dim
Number of onlookers	50% of the swarm
Number of employed bees	50% of the swarm
Number of scouts	1

5.1 GTS model

In this section, we present the model of the *dining philosopher* problem with GTS completely by showing its *Type Graph*, *Host Graph*, *Graph Transformation Rules*, and *Final State Graphs* but for the other problems, we will not describe their models here so as to avoid additional explanations.

Dining Philosophers

In this problem, a number of philosophers are sitting around a table. At first, each philosopher is thinking, and then he gets hungry; each philosopher needs two forks to eat. After eating, he starts thinking again. This process can be repeated.
Figure 2 shows the type graph of this problem in Groove tool, which is composed of philosopher and fork classes. Forks taken by the philosophers are modeled using Hold's edges. The philosopher's class has a status attribute, which specifies the current status of the philosopher. This attribute is an integer, which refers to 0 for thinking, 1 for hungry, 2 for having a left fork, and 3 for eating.

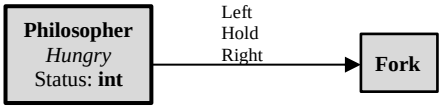

Figure 2. Type graph of the dining philosopher problem.

Figure 3 shows the Host Graph (start state) of this problem. In the initial state, all forks are on the table, so there are no hold edges in the model, and the status of all the philosophers is think = 0.

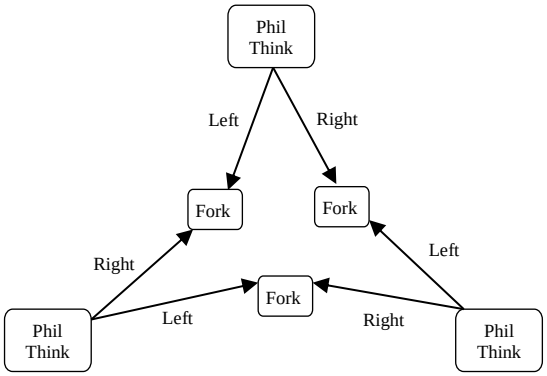

Figure 3. Host graph of the dining philosopher problem with 3 philosophers and forks.

In figure 4, the graph transformation rules are shown for the dining philosopher problem.

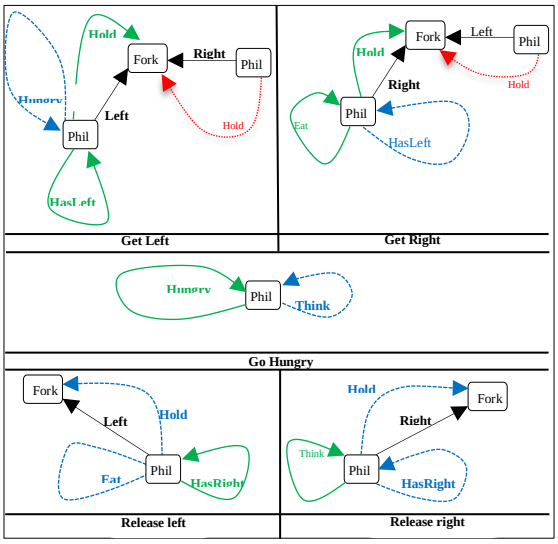

Figure 4. Dining philosopher's graph transformation rules.

If all the three philosophers take their left fork and wait for the right fork to start eating, then a *deadlock* occurs. A graph of this deadlock state (final state) is shown in figure 5.

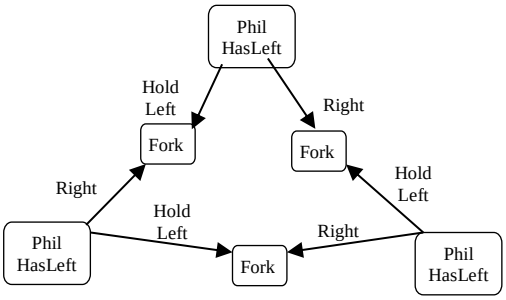

Figure 5. The final state graph for the dining philosophers problem with 3 philosophers and 3 forks.

5.2 Evaluation

The results of the experiments in these 4 problems by applying the proposed algorithm and comparing with other algorithms are presented in tables 2-5.
As shown in table 2, the speed of ABC-SA as a search strategy is much better than that of the other algorithms, and in all cases, in all 30 runs of the algorithm, it succeeds to detect the deadlock, so the "hit rate" of this algorithm is 100%.
As shown in table 3, since in the state space of this problem the number of deadlock states is high, and they can be reached from different paths, the response time is low.
Table 4 shows that the response time of the ABC-SA algorithms is lower than that of the other algorithms. However, it is worth mentioning that the hit rate of the ABC algorithm in comparison with ABC-SA is lower. For example, in the first case, the 8 × 8 dimension, in the 30 runs of these algorithms, ABC can find deadlock in 24 runs; this means that its hit rate is 80% (that happens because of trapping in the local optima problem of the ABC

algorithm) but in the same situation, the ABC-SA algorithms get response in all 30 runs, so its hit rate is 100%.

As it is shown in table 5, although the nBOA response time is lower than that of ABC and ABC-SA in the first two cases, it cannot find deadlocks in the third case; however, ABC and ABC-SA can find it in all cases and all runs, so their hit rates are 100% in this complicated problem.

Figures 6-9 show comparison of response time of the dining philosopher problem, Pacman problem, N-Queen problem, and 8-puzzle problem in our proposed approach and in the other approaches. As they show, the response time of our approaches is lower than that of the other approaches.

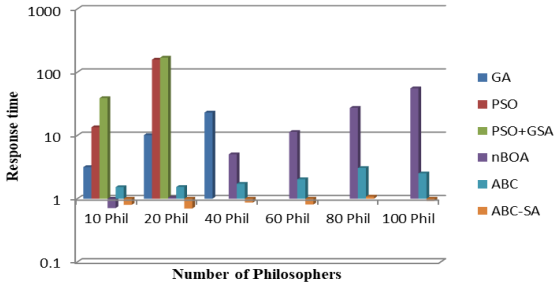

Figure 6. Comparison of response time for the dining philosophers problem.

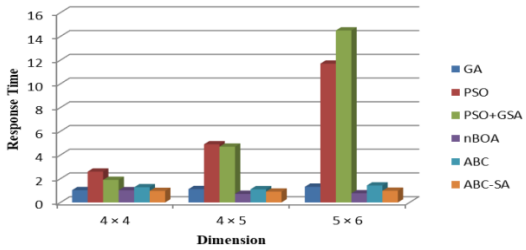

Figure 7. Comparison of response time for the pacman problem.

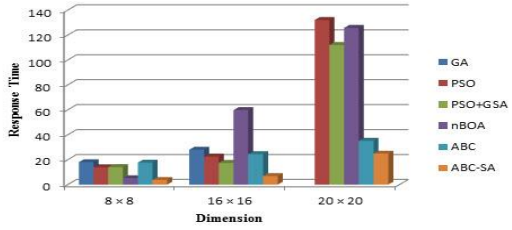

Figure 8. Comparison of response time for the N-Queen problem.

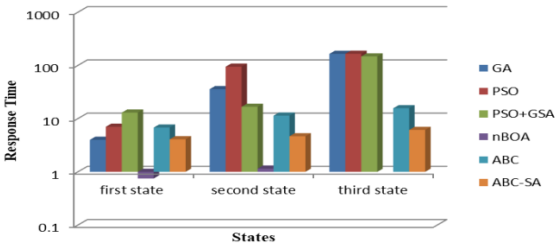

Figure 9. Comparison of response time for the 8-puzzle problem.

6. Running details

In tables 6 and 7, we describe the running details of different approaches on the dining philosophers' and N-Queen models (we only show the results of two models due to the limitation of the page numbers but the Pacman and 8-puzzle's results are the same). In these tables, we have the maximum and minimum numbers of explored states, and the transitions and runtimes in each 30 runs for every algorithm on the specific models. As shown in the results obtained, in both the maximum and minimum columns, the number of explored states and transitions in the ABC algorithm is more than that of ABC-SA in most cases. This shows that the hybrid algorithm can find deadlocks by finding the smaller counter-examples and by comparing the runtimes, so we see that the ABC-SA algorithm can also find deadlocks in a shorter time compared to the ABC algorithm. The results obtained indicate that the explored states and transitions of the ABC-SA algorithm in almost all the cases are less than those in the other algorithms. They showed that we reached our goal because we found deadlocks by exploring fewer numbers of states and shorter response times.

7. Wilcoxon signed-rank test for evaluating results

The Wilcoxon signed-rank test is a non-parametric statistical hypothesis that is used for comparing two related samples [65]. It is a test that can affirm that the results of our proposed approaches are significantly different from those of the others. This test was performed by the SPSS toolbox. In SPSS, if the output decision criterion (sig.) is less than 0.05, it can be concluded that there is a significant difference between the two groups of data. We performed this test on the results of the hybrid ABC-SA approach against the GA, PSO, PSO-GSA, BAPSO, and nBOA approaches in terms of the response times. The results of the test are shown in table 8. As shown in this table, the decision criterion (sig.) was less than 0.05 for ABC-SA against all the other approaches except for nBOA that was 0.23. Thus we can conclude that the average response time of our proposed approach is significantly different from that of the other approaches. As shown in the previous section, our proposed algorithm's hit rate was 100%, while the hit rate of the other algorithms was not 100% in all cases, and this was an important parameter in this context.

Table 2. Experimental results of different approaches to the dining philosophers problem (average response time (s)).

Count of phils.	Depth of search	Colony size	BFS /DFS/A*	GA	PSO	PSO+GSA	BAPSO	nBOA	ABC	ABC-SA
10	**25**	**15**	Out of memory	3.16	13.45	38.92	8.34	0.71	1.52	0.8
20	**100**	**20**		10.12	158	170	64.6	1.04	1.53	0.7
40	**120**	**40**		23	-	-	-	5.03	1.72	0.87
60	**140**	**60**		-	-	-	-	11.35	2.03	0.81
80	**180**	**80**		-	-	-	-	27.3	3.06	1.08
100	**220**	**100**		-	-	-	-	55.53	2.51	0.92

Table 3. Experimental results of different approaches to the Pacman problem (average response time (s)).

Dimension	Depth of search	Colony size	BFS	DFS	A*	GA	PSO	PSO+GSA	BAPSO	nBOA	ABC	ABC-SA
4 × 4	**100**	**40**	6	4	18.56	1.034	2.6	1.9	1.3	1.03	1.28	0.97
4 × 5	**100**	**60**		4.5		1.123	4.9	4.7	2.8	0.72	1.11	0.91
5 × 6	**100**	**80**	Out of memory	Out of memory	Out of memory	1.321	11.7	14.5	7.9	0.77	1.43	0.98

Table 4. Experimental results of different approaches to the N-Queen problem (average response time (s)).

Dimension	Depth of search	Colony size	BFS	DFS	A*	GA	PSO	PSO+GSA	BAPSO	nBOA	ABC	ABC-SA
8 × 8	**100**	**20**	Out of memory			18	13.8	14	19.39	5.12	17.59	3.71
16 × 16	**100**	**25**				28	22.5	17.4	Not found	59.82	24.38	6.82
20 × 20	**100**	**30**	-				132	112		125.67	35.1	24.81

Table 5. Experimental results of different approaches to the 8-puzzle problem (average response time (s)).

Start state	Depth of search	Colony size	BFS/DFS/A*	GA	PSO	PSO+GSA	BAPSO	nBOA	ABC	ABC-SA
1 2 3 4 5 6 _ 7 8	**100**	**40**	Out of memory	4	7.03	13	9.63	0.75	6.8	4.09
_ 1 3 5 2 6 4 7 8	**100**	**50**		35.81	94.7	16.7	45.53	1.15	11.39	4.66
2 3 6 1 4 8 7 _ 5	**100**	**60**		165	165.5	147.7	70.93	Not found	15.69	6.14

Table 6. Running details of the ABC and ABC-SA approaches in the dining philosophers' problem.

Number of philosophers	ABC						ABC-SA					
	Maximum #explored states /#transitions		Minimum #explored states /#transitions		Minimum runtime(s)	Maximum runtime(s)	Maximum #explored states /#transitions		Minimum #explored states /#transitions		Minimum runtime (s)	Maximum runtime (s)
10	22	25	7	6	0.64	3.45	17	19	7	6	0.31	2.82
20	22	23	7	6	0.68	6.13	17	18	7	6	0.47	2.69
40	17	20	7	6	0.75	3.87	17	18	7	6	0.4	3.01
60	22	23	7	6	0.84	5.59	17	20	7	6	0.42	3.78
80	22	27	7	6	0.92	9.017	22	27	7	6	0.35	4.67
100	22	26	7	6	1.02	6.34	17	19	7	6	0.41	4.44

Table 7. Running details of the ABC and ABC-SA approaches in the N-Queen problem.

Dimension	ABC						ABC-SA					
	Maximum #explored states /#transitions		Minimum #explored states /#transitions		Minimum runtime(s)	Maximum runtime(s)	Maximum #explored states /#transitions		Minimum #explored states /#transitions		Minimum runtime(s)	Maximum runtime(s)
8 × 8	117	133	9	8	4.66	55.51	99	100	11	10	2.02	6.69
16 × 16	111	121	13	15	5.005	85.48	89	102	11	12	2.06	7.51
20 × 20	99	112	27	26	8.97	123.55	79	81	15	16	4.60	95.77

Table 8. Results of the Wilcoxon signed-rank test on ABC-SA with other algorithms.

Ranks

		N	Mean rank	Sum of ranks
ABCSA-GA	Negative ranks	12[a]	9.25	111.00
	Positive ranks	3[b]	3.00	9.00
	Ties	0[c]		
	Total	15		
ABCSA-PSO	Negative ranks	16[a]	8.50	136.00
	Positive ranks	0[b]	.00	.00
	Ties	0[c]		
	Total	16		
ABCSA-PSOGSA	Negative ranks	16[a]	8.50	136.00
	Positive ranks	0[b]	.00	.00
	Ties	0[c]		
	Total	16		
ABCSA-BAPSO	Negative ranks	14[a]	7.50	105.00
	Positive ranks	0[b]	.00	.00
	Ties	0[c]		
	Total	14		
ABCSA-nBOA	Negative ranks	10[a]	12.45	124.50
	Positive ranks	9[b]	7.28	65.50
	Ties	0[c]		
	Total	19		
ABCSA-ABC	Negative ranks	20[a]	10.50	210.00
	Positive ranks	0[b]	.00	.00
	Ties	0[c]		
	Total	20		

a. first sample < second sample

b. first sample > second sample

c. first sample = second sample

Test statistics[a]

	ABCSA/GA
Z	-2.897[b]
Asymp. Sig. (2-tailed)	.004

	ABCSA/PSO
Z	-3.516[b]
Asymp. Sig. (2-tailed)	.000

	ABCSA/PSOGSA
Z	-3.516[b]
Asymp. Sig. (2-tailed)	.000

	ABCSA/BAPSO
Z	-3.296[b]
Asymp. Sig. (2-tailed)	.001

	ABCSA/nBOA
Z	-1.187[b]
Asymp. Sig. (2-tailed)	.235

	ABCSA/ABC
Z	-3.920[b]
Asymp. Sig. (2-tailed)	.000

a. Wilcoxon signed ranks test

b. Based on positive ranks

8. Conclusion

In this work, we mapped the deadlock detection problem onto a searching problem, and then proposed a hybrid meta-heuristic algorithm, namely the ABC-SA algorithm, to cope with the state space explosion problem in searching the graph that is specified and modeled by the graph transformation system. We used the refutation techniques to look for a path that contained a property violation, called a counter-example. We implemented our proposed approach in the Groove model checker tool, and added the ABC-SA algorithm to the existing searching strategies of this tool. The experimental results obtained showed a faster detecting deadlock under our approach compared to the other approaches. In addition, we showed that PSO, GA, PSO-GSA, nBOA, ABC, and the other mentioned algorithms could not sometimes find the deadlock but the hybrid ABC-SA algorithm can find deadlocks in all cases.

In the future research works, other properties in the model checking technique such as liveness and reachability can be considered. Moreover, another line of research work is to focus on developing a better objective function to improve the results.

References

[1] Clarke, E., Grumberg, O., Jha, S., Lu, Y., & Veith, H. (2001). Progress on the state explosion problem in model checking. Informatics, pp. 176-194.

[2] Rafe, V., Moradi, M., Yousefian, R., & Nikanjam, A. (2015). A meta-heuristic solution for automated refutation of complex software systems specified through graph transformations. Applied Soft Computing, vol. 33, pp. 136-149.

[3] Han, T., Katoen, J.-P., & Berteun, D. (2009). Counterexample generation in probabilistic model checking. IEEE Transactions on Software Engineering, vol. 35, no.2, pp. 241-257.

[4] Baier, C., Katoen, J.P., & Larsen, K. G. (2008). Principles of model checking. MIT press.

[5] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., & Hwang, L.J. (1992). Symbolic model checking: 1020 states and beyond. Information and computation, vol. 98 no.2, pp. 142-170.

[6] Groote, J. F., Kouters, T. W., & Osaiweran, A. (2015). Specification guidelines to avoid the state space explosion problem. Software Testing. Verification and Reliability, vol.25, no.1, pp. 4-33.

[7] Pelánek, R. (2008). Fighting state space explosion: Review and evaluation. International Workshop on Formal Methods for Industrial Critical Systems, L'Aquila, Italy.

[8] Alur, R., Courcoubetis, C., & Dill, D. (1990). Model-checking for real-time systems. Fifth Annual IEEE Symposium on Logic in Computer Science, Philadelphia, USA.

[9] Emerson, E. A. (2008). The beginning of model checking: a personal perspective, 25 Years of Model Checking. Springer, Berlin, Heidelberg, pp. 27-45.

[10] Yousefian, R., Rafe, V., & Rahmani, M. (2014). A heuristic solution for model checking graph transformation systems. Applied Soft Computing, vol. 24, pp. 169-180.

[11] Rensink, A. (2003). The GROOVE simulator: A tool for state space generation. International Workshop on Applications of Graph Transformations with Industrial Relevance, Springer, Berlin, Heidelberg, pp. 479-485.

[12] Kastenberg, H. & Rensink, A. (2006). Model checking dynamic states in GROOVE. International SPIN Workshop on Model Checking of Software, Springer, Berlin, Heidelberg, pp. 299-305.

[13] Engels, G., Soltenborn, C., & Wehrheim, H. (2007). Analysis of UML activities using dynamic meta modeling. International Conference on Formal Methods for Open Object-Based Distributed Systems, Springer, Berlin, Heidelberg, pp. 76-90.

[14] Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., & Yovine, S. (1998). Kronos: A model-checking tool for real-time systems. International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, Springer, Berlin, Heidelberg, pp. 298-302.

[15] Bouali, A. (1998). XEVE: an ESTEREL verification environment. International Conference on Computer Aided Verification, Springer, Berlin, Heidelberg, pp. 500-504.

[16] Song, D. X. (1999). Athena: a new efficient automatic checker for security protocol analysis. In Proceedings of the 12th IEEE Computer Security Foundations Workshop, pp. 192-202.

[17] Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., & Tacchella, A. (2002). Nusmv2: An opensource tool for symbolic model checking. In International Conference on Computer Aided Verification, Springer, Berlin, Heidelberg, pp. 359-364.

[18] Holzmann, G. J. (2004). The SPIN model checker: Primer and reference manual. Addison-Wesley Reading.

[19] Barnat, J., Brim, L., Ceska, M., & Rockai, P. (2010). Divine: Parallel distributed model checker. Parallel and Distributed Methods in Verification, In Ninth International Workshop on Parallel and Distributed Methods in Verification, and Second International Workshop on High Performance Computational Systems Biology, IEEE, pp. 4-7.

[20] D'silva, V., Kroening, D., & Weissenbacher, G. (2008). A survey of automated techniques for formal software verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 7, pp. 1165-1178.

[21] Merz, S. (2001). Model checking: A tutorial overview, Modeling and verification of parallel processes. Springer, pp. 3-38.

[22] Rozier, K. Y. (2011). Linear temporal logic symbolic model checking. Computer Science Review, vol. 5, no. 2, pp.163-203.

[23] Jhala, R. & Majumdar, R. (2009). Software model checking. ACM Computing Surveys, vol. 41, no. 4, 21.

[24] Lamport, L. (1977). Proving the correctness of multiprocess programs. IEEE Transactions on Software Engineering, vol. 2, pp.125-143.

[25] Khurshid, S., Păsăreanu, C. S., & Visser, W. (2003). Generalized symbolic execution for model checking and testing. In International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, Berlin, Heidelberg. pp. 553-568.

[26] Beneš, N., Černá, I., & Křetínský, J. (2011). Modal transition systems: Composition and LTL model checking. International Symposium on Automated Technology for Verification and Analysis, pp.228-242.

[27] Emerson, E. A. & Namjoshi, K. S. (1998). On model checking for non-deterministic infinite-state systems. Thirteenth Annual IEEE Symposium on Logic in Computer Science, pp. 70-80.

[28] Clarke, E. M., Grumberg, O., & Peled, D. (1999). Model checking: MIT press.

[29] Larsen, K. G., Pettersson, P., & Yi, W. (1995). Model-checking for real-time systems. International Symposium on Fundamentals of Computation Theory, pp. 62-88.

[30] Boussaï D, I., Lepagnot, J., & Siarry, P. (2013). A survey on optimization metaheuristics. Information Sciences, vol. 237, pp. 82-117.

[31] Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization. Technical report-tr06, Erciyes university, engineering faculty, computer engineering department.

[32] Gao, W. f., Huang, L.-l., Liu, S.y., Chan, F. T., Dai, C., & Shan, X. (2015). Artificial bee colony algorithm with multiple search strategies. Applied Mathematics and Computation, vol. 271, pp. 269-287.

[33] Abbasi, B., Niaki, S. T. A., Khalife, M. A., & Faize, Y. (2011). A hybrid variable neighborhood search and simulated annealing algorithm to estimate the three parameters of the Weibull distribution. Expert Systems with Applications, vol. 38, no. 1, pp. 700-708.

[34] Chen, S.M., Sarosh, A., & Dong, Y.F. (2012). Simulated annealing based artificial bee colony algorithm for global numerical optimization. Applied mathematics and computation, vol. 219, no. 8, pp. 3575-3589.

[35] Alba, E. & Chicano, F. (2007). Finding safety errors with ACO. In Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 1066-1073.

[36] Sivaraj, H., & Gopalakrishnan, G. (2003). Random walk based heuristic algorithms for distributed memory model checking. Electronic Notes in Theoretical Computer Science, vol. 89, no.1, pp. 51-67.

[37] Clarke, E. M., Grumberg, O., Minea, M., & Peled, D. (1999). State space reduction using partial order techniques. International Journal on Software Tools for Technology Transfer, vol. 2, no. 3, pp. 279-287.

[38] Brim, L., Cerma, I., Moravec, P., & Simsa, J. (2005). Distributed partial order reduction of state spaces. Electronic Notes in Theoretical Computer Science, vol. 128, no.3, pp. 63-74.

[39] Alur, R., Brayton, R. K., Henzinger, T. A., Qadeer, S., & Rajamani, S. K. (2001). Partial-order reduction in symbolic state-space exploration. Formal Methods in System Design, vol. 18, no. 2, pp. 97-116.

[40] P Godefroid, P., van Leeuwen, J., Hartmanis, J., Goos, G., & Wolper, P. (1996). Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem, Springer Heidelberg, vol. 1032.

[41] Barnat, J., Brim, L., & Ročkai, P. (2012). On-the-fly parallel model checking algorithm that is optimal for verification of weak LTL properties. Science of Computer Programming, vol. 77, no. 12, pp. 1272-1288.

[42] Bouajjani, A., Tripakis, S., & Yovine, S. (1997). On-the-fly symbolic model checking for real-time systems. In proceedings of the 18th IEEE Conference on Real-Time Systems Symposium, pp. 25-34.

[43] Rafe, V., Rahmani, M., & Rashidi, K. (2013). A Survey on Coping with the State Space Explosion Problem in Model Checking. International Research Journal of Applied and Basic Sciences, vol. 4, no. 6, pp. 1379-1384.

[44] Gyuris, V. & Sistla, A. P. (1997). On-the-fly model checking under fairness that exploits symmetry. International Conference on Computer Aided Verification, pp. 232-243.

[45] Edelkamp, S., Jabbar, S., & Lafuente, A. L. (2006). Heuristic search for the analysis of graph transition systems. In International Conference on Graph Transformation. Springer, Berlin, Heidelberg, pp. 414-429.

[46] Behjati, R., Sirjani, M., & Ahmadabadi, M. N. (2009). Bounded rational search for on-the-fly model checking of LTL properties. In International Conference on Fundamentals of Software Engineering, Springer, Berlin, Heidelberg, pp. 292-307.

[47] Clarke, E. M., Enders, R., Filkorn, T., & Jha, S. (1996). Exploiting symmetry in temporal logic model checking. Formal Methods in System Design, vol.9, no. 1-2, pp. 77-104.

[48] Godefroid, P. & Khurshid, S. (2002). Exploring very large state spaces using genetic algorithms. In International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, Berlin, Heidelberg, pp. 266-280.

[49] Duarte, L. M., Foss, L., Wagner, F. R., & Heimfarth, T. (2010). Model checking the ant colony optimisation, Distributed, parallel and biologically inspired systems. Springer, Berlin, Heidelberg, pp. 221-232.

[50] Francesca, G., Santone, A., Vaglini, G., & Villani, M. L. (2011). Ant colony optimization for deadlock detection in concurrent systems. In 2011 IEEE 35th Annual Computer Software and Applications Conference, pp. 108-117.

[51] Chicano, F., Ferreira, M., & Alba, E. (2011). Comparing metaheuristic algorithms for error detection in java programs. International Symposium on Search Based Software Engineering, pp. 82-96.

[52] CrepinSek, M., Liu, S.-H., & Mernik, L. (2012). A note on teaching–learning-based optimization algorithm. Information Sciences, vol. 212, pp. 79-93.

[53] Daws, C., and Tripakis, S. (1998). Model checking of real-time reachability properties using abstractions. International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 313-329.

[54] Alba, E. & Chicano, F. (2008). Searching for liveness property violations in concurrent systems with ACO. In Proceedings of the 10th annual conference on Genetic and evolutionary computation, pp. 1727-1734.

[55] Yousefian, R., Aboutorabi, S., & Rafe, V. (2016). A greedy algorithm versus metaheuristic solutions to deadlock detection in Graph Transformation Systems. Journal of Intelligent & Fuzzy Systems, vol. 31, no.1, pp. 137-149.

[56] Lluch-Lafuente, A. (2003). Symmetry reduction and heuristic search for error detection in model checking.

[57] Schmidt, A. (2004). Model checking of visual modeling languages. Budapest University of Technology, Hungary.

[58] Heckel, R. (2006). Graph transformation in a nutshell. Electronic notes in theoretical computer science, vol. 148, no. 1, pp. 187-198.

[59] Zambon, E. & Rensink, A. (2014). Solving the N-Queens problem with GROOVE-towards a compendium of best practices. Electronic Communications of the EASST, vol. 67.

[60] Alba, E. & Chicano, F. (2007). Ant colony optimization for model checking. In Processdings of International Conference on Computer Aided Systems Theory, pp. 523-530.

[61] Pira, E., Rafe, V., & Nikanjam, A. (2016). EMCDM: Efficient model checking by data mining for verification of complex software systems specified through architectural styles. Applied Soft Computing, vol. 49, pp. 1185-1201.

[62] Pira, E., Rafe, V., & Nikanjam, A. (2017). Deadlock Detection in Complex Software Systems Specified through Graph Transformation Using Bayesian Optimization Algorithm. Journal of Systems and Software, vol. 131, pp. 181-200.

[63] Pira, E., Rafe, V., & Nikanjam, A. (2018). Searching for violation of safety and liveness properties using knowledge discovery in complex systems specified through graph transformations. Information and Software Technology, vol. 97, pp. 110-134.

[64] Roustaei, R.& Yousefi Fakhr, F.(2018). A Hybrid Meta-Heuristic Algorithm based on Imperialist Competition Algorithm. Journal of AI and Data Mining, vol. 6, no. 1, pp. 59-67.

[65] Wilcoxon, F. (1945). Individual comparisons by ranking methods. Biometrics Bulletin, vol. 1, no. 6, pp. 80-83.

نشریه هوش مصنوعی و داده کاوی

ارایه راهکار فرامکاشفه‌ای برای مقابله با مشکل انفجار فضای حالت در تکنیک وارسی مدل برای آزادی بن بست

ناهید رضایی و حسین مومنی*

گروه مهندسی کامپیوتر، دانشگاه گلستان، گرگان، ایران.

ارسال ۲۰۱۸/۱۰/۱۳؛ بازنگری ۲۰۱۹/۰۵/۳۱؛ پذیرش ۲۰۱۹/۱۲/۰۱

چکیده:

وارسی مدل یک روش خودکار برای درستی یابی نرم افزار است که در آن تمام حالت‌های قابل دستیابی یک سیستم از یک حالت اولیه، به منظور پیدا کردن خطاها و الگوهای مطلوب تولید می‌شوند. در روش وارسی مدل، رفتار و ساختار یک سیستم مدل می‌شود. سیستم تبدیل گراف یک زبان رسمی گرافیکی برای مشخص‌سازی و مدلسازی سیستم است. مدلسازی سیستم‌های بزرگ با سیستم تبدیل گراف مشکل انفجار فضای حالت دارد و معمولاً به مقادیر عظیمی از منابع محاسباتی نیاز دارد. در این مقاله، ما یک روش فرامکاشفه‌ای ترکیبی، برای مقابله با مشکل جستجوی فضای حالت، در سیستم تبدیل گراف ارایه خواهیم نمود زیرا الگوریتم‌های فرامکاشفه‌ای راه‌حل کارآمدی برای جستجوی گراف در سیستم‌های بزرگ ارایه می‌نمایند. با کمک ترکیب روش اجتماع زنبورعسل مصنوعی و تبرید شبیه سازی شده بجای تولید یک فضای حالت کامل، فقط بخشی از فضای حالت، تولید و وارسی می‌شود و طی آن ویژگی ایمنی و وجود برخی خطاها (نظیر بن بست) بررسی می‌شود. نتایج آزمایش‌ها نشان می‌دهد که روش پیشنهادی ما در مقایسه با روش‌های دیگر کارآمدتر و دقیق‌تر است.

کلمات کلیدی: درستی یابی نرم افزار، وارسی مدل، انفجار فضای حالت، روش‌های فرامکاشفه‌ای، سیستم تبدیل گراف.