 doi:10.20944/preprints202010.0288.v1

Fuzzy particle swarm optimization algorithm (NFPSO) for reachability analysis of complex software systems

Nahid Salimi
Department of Computer Engineering, Faculty of Engineering, Arak University
Arak, Iran
n-salimi@phd.araku.ac.ir

Vahid Rafe
Department of Computer Engineering, Faculty of Engineering, Arak University
Arak, Iran
v-rafe@araku.ac.ir

Hamed Tabrizchi
Department of Computer Science, Shahid Bahonar University
Kerman, Iran
0000-0001-9250-2232

Amir Mosavi
Kalman Kando Faculty of Electrical Engineering, Obuda University
Budapest, Hungary
amir.mosavi@kvk.uni-obuda.hu

***Abstract*— Nowadays, model checking is applied as an accurate technique to verify software systems. The main problem of model checking techniques is the state space explosion. This problem occurs due to the exponential memory usage by the model checker. In this situation, using meta-heuristic and evolutionary algorithms to search for a state in which a property is satisfied/violated is a promising solution. Recently, different evolutionary algorithms like GA, PSO, etc. are applied to find deadlock state. Even though useful, most of them are concentrated on finding deadlock. This paper proposes a fuzzy algorithm in order to analyze reachability properties in systems specified through GTS with enormous state space. To do so, we first extend the existing PSO algorithm (for checking deadlocks) to analyze reachability properties. Then, to increase the accuracy, we employ a Fuzzy adaptive PSO algorithm to determine which state and path should be explored in each step to find the corresponding reachable state. These two approaches are implemented in an open-source toolset for designing and model checking GTS called GROOVE. Moreover, the experimental results indicate that the hybrid fuzzy approach improves speed and accuracy in comparison with other techniques based on meta-heuristic algorithms such as GA and the hybrid of PSO-GSA in analyzing reachability properties.**

Keywords*— *Fuzzy Adaptive Particle Swarm Optimization, Graph Transformation System, Model Checking, Reachability Property, State Space Explosion

I. INTRODUCTION

During recent decades, software systems play an essential role in developing software systems and security considerations. Software development techniques have evolved to make more complex software systems over the years. Model-Driven engineering takes models in order to describe complex systems at multiple levels of abstraction [1]. Modeling systems make it possible to use model checking as a formal analysis technique to verify software and detect system errors in the design phase that is easier and cheaper than after the implementation. It is vital to use a modeling language to model systems and verifying them by model checking. GTS is a visual graph-based formal language that is used to model software systems with dynamic structures [2]. GTS uses graphs to describe and model the structure of complex systems [3]. For large systems, GTS tends to be huge, so the state space explosion is the main drawback of model checking [4]. Some classic methods such as symbolic verification [5, 6], partial order reduction [7, 8], symmetry model checking methods [4, 9-11], scenario-driven model checking [12], and several heuristic approaches including depth-first search [13], best-first search [13, 14], a version of the A* algorithm [15], Coverage First Search [16] and an A* search algorithm to check liveness property in the explicit-state model checking [7] is proposed to resolve the state space explosion problem. However, the use of exhaustive search in the state space also causes a lack of memory and low speed for classical and heuristic methods. In recent years, several methods based on meta-heuristic and evolutionary algorithms have been proposed in this context concerning their efficiency, comparing classical and heuristic approaches. These approaches search a subset of the state space of systems instead of all state space to achieve the specified properties, so they are more practical to resolve the state space explosion problem. In this context, some meta-heuristic approaches have been proposed to find deadlocks in the systems and refute the security property, such as frameworks based on reinforcement learning [17] and ant colony algorithm [18-20]. A new approach has been proposed using the genetic algorithm for verifying the correctness of communication protocols in [21]. This genetic validation has been tested on a hand-made protocol and on the Transmission Control Protocol. In [22] two different learning algorithms have been proposed to verify safety, reachability and liveness properties of systems whose state space is expressible by regular expressions. In the other paper, an ACO-based approach is presented to prevent the state explosion problem for finding deadlocks in complex networks described by using Calculus of Communicating Systems [23]. Two other papers [24, 25] proposed using a type

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 October 2020 doi:10.20944/preprints202010.0288.v1

of ACO model to refute safety and liveness properties in concurrent systems. This Approach applies the GA algorithm on several random paths with a specific length starting from the initial state as the initial population and finds the first path, which leads to a deadlock state. In order to evaluate the efficiency of this solution, it is implemented in GROOVE toolset. Another proposed approach for this problem uses the PSO algorithm to detect deadlocks in graph transformation systems [26]. The authors have also suggested a hybrid algorithm based on PSO and GSA to avoid the local optima problem. Another solution using a simple greedy algorithm called BFA is also proposed in this paper in which a state with the least outgoing transitions must be selected in each exploration step. These two approaches are implemented in the GROOVE. In [28], an efficient approach is proposed using data mining techniques called EMCDM to check the model of complex software systems that are designed according to an architectural style and modeled by GTS formally. The EMCDM approach is applied to verify the reachability property to refute the safety property. In other research [29], a BOA-based approach is proposed to detect deadlocks in systems specified through graph transformations. The results of this algorithm indicate more improvement in terms of speed and accuracy. One of the properties which can be verified in model checking process is reachability property. It can also be used instead of refuting a similar safety property. For checking reachability property, a specified goal state must be generated by graph transformation rules from an initial configuration of the system; however, there is no meta-heuristic approach working on reachability check. This aspect of the model checking is very similar to rule-based DSE frameworks [30, 31].

In this paper, two meta-heuristic solutions based on PSO approach are used to verify the reachability property in the systems specified through GTS. Despite the high speed of convergence of the PSO algorithm, estimating the appropriate value for C1 and C2 is a critical challenge. As well a Fuzzy Adaptive Particle Swarm Optimization Algorithm is proposed, which uses Fuzzy systems to estimate the best values of C1 and C2 in each particle swarm optimization Iteration. These proposed approaches for checking the reachability property can also be used as a search-based model transformation approach to search and produce a set of target models from a given initial model.

This paper is organized as follows: We present our proposed approaches based on PSO and Fuzzy PSO algorithms in Section 2. Section 3 briefly describes the necessary background, such as model checking, GTS formalism, PSO algorithm, and Fuzzy inference systems. Section 4 includes obtained the experimental results based on several well-known case studies. Moreover, a discussion on the observations is presented. The superiority of the proposed approaches have been discussed in section 5. Finally, Section 6 concludes the paper and highlights the future works.

II. BACKGROUND

A. Model Checking

As a completely automatic technique, model checking attempts to verify the correctness properties of different systems. This technique determines if a typical correctness property suits a given system by exploring the possible transitions in different states. Some of the properties which can be verified in the model checking process are safety, reachability, liveness, and fairness. A safety property defines that "no undesired situation should occur" or a "desired event" must always happen in a system. This property is satisfied when all finite and infinite paths in the model satisfy it. Finding a finite path into a goal state violating the safety property can decline the property. In the case of the reachability property, the model checks whether there will be a particular configuration in the system or not. Finding a finite path into the goal state satisfying the property leads to detecting the state that leads to verifying the reachability property. The two above-mentioned properties are dual; in fact, verifying the reachability property as the negation of a safety property can be a counterexample violating the safety property [32].

B. Graph Transformation System

Using model checking to verify a system requires that the system is described by a formal language. To model systems with dynamic structures, graph transformation can be used as a graph-based visual formal language [33]. The formal, accurate mathematical basis of the graph transformation system is one of its basic features [34]. An attributed GTS is a triple: AGT = (TG, HG, R) in which the type graph, host graph, and ruleset are shown by TG, HG, and R, respectively. The system's overview and Meta-model can be determined by TG. The initial configuration of a system can be determined by HG, as an instance of TG. A system's different configurations can be created by applying transformation rules on a host graph. A triple like (LHS, RHS, NAC) can define the graph transformation rule set R on a TG. The left and right-hand sides are represented by LHS and RHS, determining the rule's pre-conditions and post-conditions, respectively. NAC (harmful application condition) is a particular configuration that is used to verify that there is not any subgraph in the rule.

C. Particle Swarm Optimization Algorithm

PSO algorithm, as one of the most widely used optimization algorithms to solve multi-dimensional problems. At first, this algorithm randomly creates an initial population of candidate solutions as particles. The position of the i^th the particle can be described by the vector xi. Next step, in each iteration, the algorithm calculates each particle's fitness. Fitness is the degree of the optimality of a particle to be a desirable solution. Until a termination criterion occurs, g_best and p_best are updated for all particles as the "personal best" and "global best" particles, respectively. The velocity and the position of particles are updated, as shown by Eq. (1) and Eq. (2):

$$vi(t+1) = W * vi(t) + C1 * (pbesti - xi(t)) * R1 + C2 * (gbest - xi(t)) * R2 \quad (1)$$

$$xi(t+1) = xi(t) + vi(t+1) \quad (1)$$

C1 and C2 represent "cognitive coefficient" and "social coefficient," specifying the weight of the personal experience and collective experience effect on particle behavior, respectively. These two coefficients are real-valued and usually in the range of $0 \leqslant C1 + C2 \leqslant 4$. The momentum weight represented by W determines to what extent a particle's velocity in the current step influences its velocity in the subsequent step. Also, R1 and R2 are two random factors

Identify applicable funding agency here. If none, delete this text box.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 October 2020 doi:10.20944/preprints202010.0288.v1

containing two diagonal matrices of random real numbers lying the range of (0,1)[35]. The iterative process of calculating finesse and updating g_best, p_best, position and velocity for all particles continues until a termination condition is realized.

The pseudo-code related to the algorithm can be seen in Algorithm 1.

Algorithm 1. Pseudo code of the PSO algorithm

1. Initialization
   a. Initialize **the particle's position** $x_i(t)$ **(*t=0* , *i:1…N*)**
   b. Initialize **the particle's best position to its initial position**

$p_i(t) = x_i(t)$ **(*t=0* , *i:1…N*)**

   c. Calculate **the fitness of each particle** $f(x_i(t))$ **(t=0 , i:1…N)**
   d. if $f(x_i(0)) \leq f(x_j(0))$ **(for each *i:1…N*, $i \neq j$)** then **initialize the global best as** $g_{best} = x_j(t)$ **(*j:1…N*)**
2. while **a stopping criterion is not met,**

repeat **the following steps:**

   a. Update **the velocity** v_i **for each particle:**

$v_i(t+1) = W*v_i(t) + C_1(p_{besti} - x_i(t))R_1 + C_2(g_{best} - x_i(t))R_2$

   b. Update **the position** x_i **for each particle:**

$x_i(t+1) = x_i(t) + v_i(t+1)$

   c. Evaluate **the fitness** $f(x_i(t+1))$ **for each particle**
   d. if $f(x_i(t+1)) \geq f(p_{besti})$ then
      Update **personal best:** $p_{besti} = x_i(t+1)$
   e. if $f(x_i(t+1)) \geq f(g_{best})$ then
      Update **personal best:** $g_{best} = x_i(t+1)$

At the end of iterative process the best solution is represented by g_{best}.

D. Fuzzy inference system

A fuzzy system consisted of three main parts namely, the fuzzification, FIS, and defuzzification.

- In the fuzzification process, the linguistic variables (of inputs and outputs) are defined.
- The FIS defines the rules describing how the system works and map inputs into outputs.
- In the defuzzification process, outputs are computed.

To design a fuzzy model, the following should be defined: the input and output variables, the fuzzy membership functions, the fuzzy rules, and the parameters used in equations (2) and (3).

III. The Proposed Approaches

Here, two PSO and fuzzy Adaptive PSO-based approaches are proposed to manage the state space explosion in the systems defined formally by GTS to verify reachability property. Although meta-heuristic algorithms such as PSO are widely employed to deal with optimization problems, we use these algorithms to search the reachability property in a potentially huge state space which may even be infinite.

A. Particle Encoding

A reachability property *p* can be verified by finding a state where *p* occurs through exploring the system's reachable state space. The output in our proposed algorithms is the path that starts from an initial state and ends at the state holding the reachability property. As mentioned, particles are candidate solutions, and their positions can be specified by a sequence of the numbers representing a path. Each number represents a transition in each stage, and its minimum value is 0, whereas its maximum value is the maximum number of the state space's possible outgoing transitions. For example, the path '1 0 2 1' in Figure 1 shows a particle's position. This position can be encoded to the path "$r_1 r_0 r_2 r_1$" that r_i implies the applied rule.

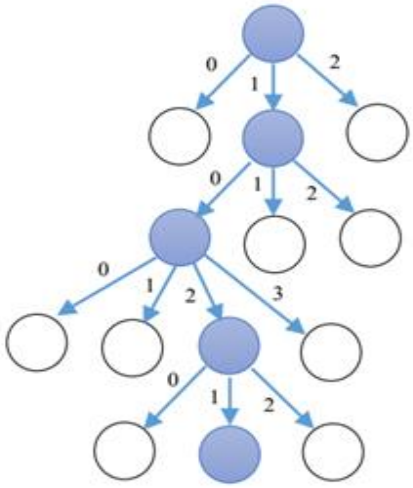

Figure 1. A Solution Encoded by the Path <1,0,2,1>

B. Fitness Function

- Fitness is the degree that specifies how much a particle is good to be the goal solution. In our approach, each particle is a path with a specific length that starts from the initial state and ends at another state belonging to the state space and our goal is to find a state that is the same as defined reachability property. One can guess that the more similar the last state of the path to the reachability property the more likely the path is to be a promising one. So, we use the similarity between the path's last state and the specified reachability property for defining fitness function. In this paper, the system is modeled by GTS and the associated states and properties are represented by graphs using GROOVE toolset. As mentioned earlier, a GTS is a triple (TG, HG, R) where TG, HG, and R represent type graph, host graph, and graph transformation rule set respectively. Also, R can be specified by the triple (LHS, RHS, NAC) where LHS and RHS are left and right-hand sides describing the pre and post-conditions of the rules respectively. Also, NAC stands for Negative Application Condition and specifies a configuration which should not occur to apply the rule.

  In GROOVE toolset, LHS, RHS, and NAC are represented together as an individual graph using colors to recognize the original LHS, RHS, and NAC graphs. If the blue dashed thin edges and nodes are present in LHS, the rule could be

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 October 2020 doi:10.20944/preprints202010.0288.v1

applied to the host graph, and they can be removed after applying the rule. The bolded green solid edges and nodes belong to RHS, which should be created after rule application [36]. In this graph, NACs also displayed by red bold dashed nodes and edges. To apply the rule, they should not occur in the graph. Each node and edge in the graph can have their labels that can be defined by a self-loop edge named by the node's label.

The two inputs of the fitness function are a particle and the under-study reachability property. The fitness value can be calculated as follows:

1. Finding the pairs of two nodes in which the first node belongs to the given property graph (except NAC nodes) and the second belongs to the last state of the path specified by the particle so that they have the same labels.

2. Calculating the total number of pairs found in the first step.

3. Calculating the total number of each NAC's nodes and edges for given properties occurring in the graph specifies the path's last state which is encoded using the given particle.

4. The difference between the value calculated in step 2 and the value achieved in step 3 is considered as the fitness value.

The pseudo-code related to the fitness function can be seen in Algorithm 2.

Algorithm 2. Fitness Function of PSO and FAPSO

1. Input & Output:
   a. **Input: h: a particle and**
      p: a given reachability property to be checked;
   b. **Output: the fitness value of h;**

2. Initialization:
   a. **Initialize NodeList member Np_i with node *i*th of G_p;**
      (i: 0 to Number of nodes G_p)
   b. **Initialize EdgeList member Ep_i with edge *i*th of G_p;**
      (i: 0 to Number of edge G_p)
   c. **Initialize NodeList member Nh_i with node *i*th of G_h;**
      (i: 0 to Number of nodes G_h)
   d. **Initialize EdgeList member Eh_i with edge *i*th of G_h;**
      (i: 0 to Number of edge G_h)
   e. **Initialize BooleanList member $hVisited_{ij}$ with *false*;**
   f. **Initialize BooleanList member $pVisited_{ij}$ with *false*;**
   g. **Initialize BooleanList member $Visited_{ij}$ with *false*;**
      (For part e, f and g: i: 0 to Number of nodes G_h and j: 0 to Number of nodes G_p)
3. for each **Nh_i**
   for each **Np_j**
   EdgeList ENP = all edges of Ep whose source node is Np_j;
   EdgeList ENH = all edges of Eh whose source node is Nh_i;
   E-Count$_{ij}$ = The number of pairs (p,h) which (p) is from ENP and (h) is from ENH as p's label is equal to h's label;
   PE-Count$_{ij}$ = size of ENP;
   DE-Count$_{ij}$ = E-Count$_{ij}$ – PE-Count$_{ij}$;
   end for
   end for

4. **EQ-Count = 0;**
   while **all $Visited_{ij}$ is not *true*** do
   Find the smallest DE-Count$_{ij}$ that $Visited_{ij}$ = *false*;
   $Visited_{ij}$ = *true;*
   if **!$pVisited_{ij}$ && !$hVisited_{ij}$** then
   EQ-Count += E-Count$_{ij}$;
   $pVisited_{ij}$ = *true;*
   $hVisited_{ij}$ = *true;*
   end if
   end while

5. Find **all NACs of G_p and store in ArrayList of NACs allNAC**
   NEQ-Count = 0;
   for each **NAC_i in allNAC do**
   NEQ-Count += The number of nodes and edges of NAC_i occurring in G_h;
   end for
- return EQ-Count – NEQ-Count;

C. PSO-based Approach

The first proposed approach applies the PSO algorithm to search the state space to find the path starting from an initial state and ending at the state satisfying the given reachability property. As already mentioned in this algorithm each particle is encoded into a path of transitions. The algorithm starts with an initial random population of particles. Then, the fitness value of each particle is calculated, g_{best} and p_{best} are updated based on the fitness values of the particles. Next, the termination condition is checked. If updated g_{best} is a perfect solution or current iteration number of the algorithm is not less than the predefined maximum number of generations, the algorithm will be finished otherwise Equations 1 and 2 will update each particle's velocity and position respectively. The algorithm will run with the phase of calculating fitness until one of the termination conditions is met.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 October 2020 doi:10.20944/preprints202010.0288.v1

D. FAPSO-based Approach

Another approach proposed in this paper applies the Fuzzy Adaptive PSO algorithm called FAPSO to verify the GTS-based defined systems' reachability property. In traditional PSO, parameters C_1 and C_2 are two constant values which cannot be varied through all generations. It should be noted that better results can be obtained through dynamic changes in C_1 and C_2 during algorithm execution. In FAPSO, fuzzy inference systems can be used to adjust C_1 and C_2 in each generation. As a PSO algorithm, the proposed FAPSO starts with an initial population of particles produced randomly. Next, each particle's fitness value is obtained and g_{best} and p_{best} are updated based on the fitness values of particles. If the termination condition is not met, two input parameters of the fuzzy system called Diversity and Iteration will be calculated. The diversity measure can be defined as the dispersion degree of particles, i.e. the more the particles are separated the higher the diversity. According to Eq. (3), the diversity measure can be the average Euclidean distance between each particle and the best one in the related generation [37].

$$Diversity\,(S(t)) = \frac{1}{n_s}\sum_{i=1}^{n_s}\sqrt{\sum_{j=1}^{n_x}(x_{ij}(t)-\bar{x}_j(t))^2} \quad (3)$$

The fuzzy system's second input is a percentage of iterations calculated by Eq. (4). At the start of the algorithm, Iteration is "Low" and gets higher while the number of algorithm iterations is getting close to the maximum iteration number [37].

$$Iteration = \frac{Current\ Iteration}{Maximum\ of\ Iteration} \quad (4)$$

The two above-mentioned measures are used as the inputs of the fuzzy system adjusting C_1 and C_2. Of course, C_1 and C_2 will be the outputs of this fuzzy system. It is worth noting that the fuzzy system's inputs lie in the range of [0, 1]. The Iteration variable can be defined in the acceptable range of values, but it is necessary to perform normalization on the Diversity to convert it into a value between 0 and 1. The normalization performed on Diversity is shown in Eq. (5) and (6). Eq. (5) indicates that when the maximum and minimum Euclidean distances are equal, normalized Diversity is 0 because particles' positions have not changed. If the maximum and minimum Euclidean distances are different, normalized Diversity can be calculated by Eq. (6) [37].

$$Normal\ Diversity = \begin{cases} 0 & MinDiversity = MaxDiversity \\ Norm & MinDiversity \neq MaxDiversity \end{cases} \quad (5)$$

$$Norm = \frac{Diversity - MinDiversity}{MaxDiversity - MinDiversity} \quad (6)$$

In this approach, a Mamdani's fuzzy system inference method with two input variables of Iteration and Diversity and two output variables of C_1 and C_2 is proposed. Figure 2 presents the proposed fuzzy system.

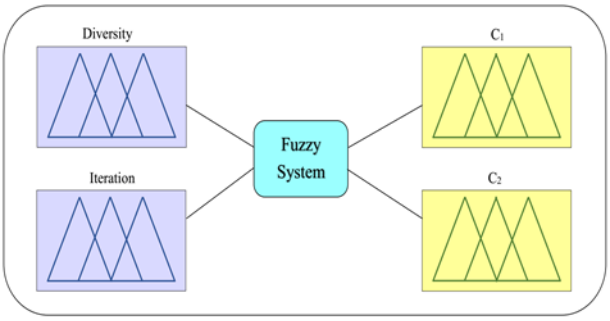

Figure 2. The Fuzzy System of the Proposed Approach

For each input of the fuzzy system, Three triangular membership functions are designed [37]. Figures 3 and 4 represent the membership functions employed for the input variables of Iteration and Diversity respectively.

According to the fact that it is recommended to select C1 and C2 from the range of [0.5 2.5] [38], the output variables should be adjusted in this range. As seen in Figures 5 and 6, the output variables of C1 and C2 are granulated in five triangular membership functions. Tables 1 and 2 show the fuzzy system's rule sets. It should be noted that considering the following two points is necessary to define the fuzzy system's rules.

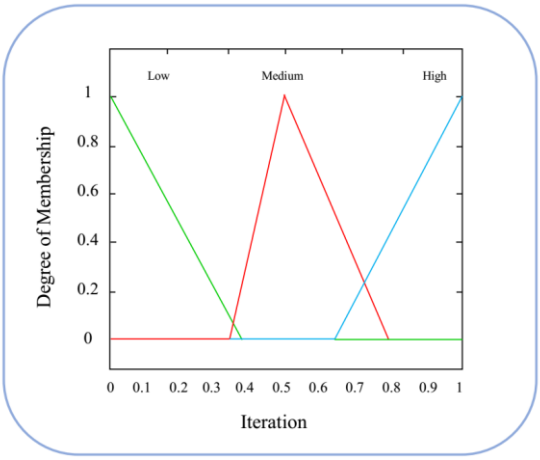

Figure 3. Input 1: Iteration

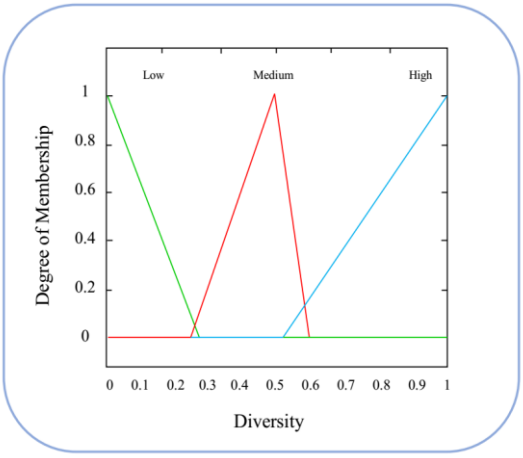

Figure 4. Input 2: Diversity

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 October 2020 doi:10.20944/preprints202010.0288.v1

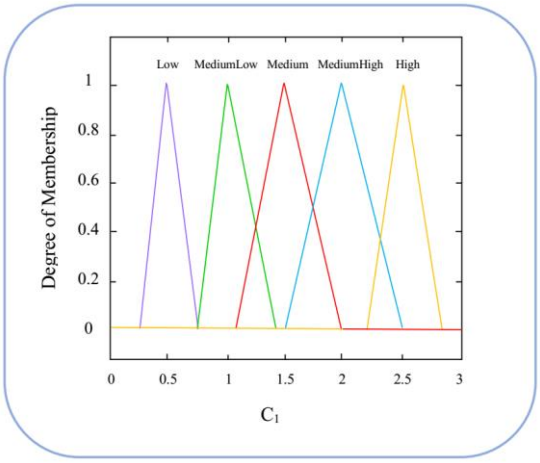

Figure 5. Output 1: C1

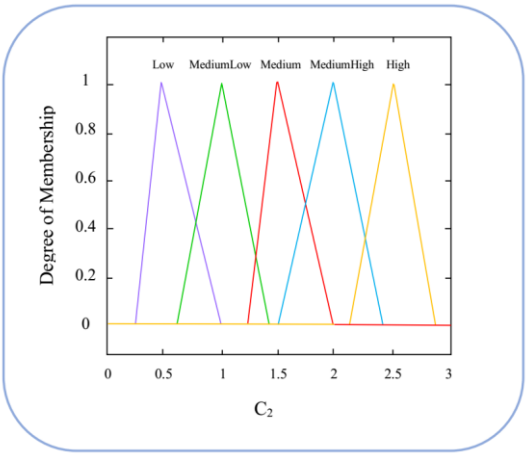

Figure 6. Output 2: C2

The first point is that the exploration process must be done in the early iterations of the PSO algorithm to exploit desired solutions eventually. The second one is that exploration must be done in low diversities whereas the exploitation must be done in high diversities i.e. when particles are far apart [37, 43].

TABLE I RULE SET OF FUZZY SYSTEM TO CALCULATE C_1

Iteation		**Low**	**Medium**	**High**
Diversity	Low	High	Medium High	Medium
	Medium	Medium High	Medium	Medium Low
	High	Medium High	Medium Low	Low

TABLE II. RULE SET OF FUZZY SYSTEM TO CALCULATE C_2

Iteration		**Low**	**Medium**	**High**
Diversity	Low	Low	Medium Low	High
	Medium	Medium	Medium	Medium High

When the fuzzy system's input variables namely, Diversity and Iteration are calculated via Eq. 3 and Eq. 4, in the fuzzification phase, the percentage of their membership can be obtained using the membership functions shown in Figures 5 and 6. Each membership function has three categories of high, medium, and low. In the next step, the rules defined by the inference system which is shown in Tables 1 and 2 are applied to calculate C_1 and C_2. The variables acquired from the inference system are the percentage of the membership of C_1 and C_2 to five categories high, medium-high, medium, medium-low, and low. The used defuzzification method is the centroid method that calculating the ultimate values of C_1 and C_2 through output membership functions presented in Figures 7 and 8 respectively. As seen in Figure 7, the proposed approach attempts to dynamically adjust the PSO algorithm's parameters in each iteration by taking advantage of a fuzzy system.

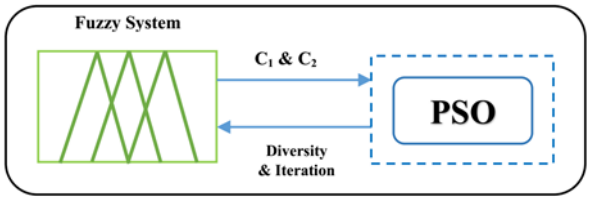

Figure 7. The main idea of the approach

When C_1 and C_2 values are achieved form fuzzy system, the PSO algorithm will update each particle's velocity and by equations 1 and 2. Then, the phase of fitness calculation will be performed until one of the termination conditions is met.

IV. Experimental Results

We implemented our approaches in GROOVE toolset using Java programming language to evaluate and compare their performance. We manipulate some existing classes of the GROOVE and create some new classes to better implement our approaches. For evaluation purposes, the dining philosophers [39], shopping [41], Pac-Man [40], process life cycle, N-Queen, and 8-puzzle [42] models were considered. These models can be downloaded through the web[1]. The initial parameters used in PSO are listed in Table 3. The value of *W* and Iteration used in the FAPSO algorithm is the same as the PSO method. A PC with an Intel CORE i5 processor and 3 GB of memory was employed for our experiments. it should be noted that the values of parameters like depth limit and population size is defined based on the problems and their model size so that the lager the size of

[1] https://sourceforge.net/projects/groove-and-ga-boa-reachability/files/

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 October 2020 doi:10.20944/preprints202010.0288.v1

the model, the more expanded the state space and the harder the goal state finding; so, the higher the depth limit and population size should be defined.

TABLE III. INITIAL PARAMETERS OF THE PSO-BASED APPROACH

Iteration	**100**
C_1	2
C_2	2
W	0.8

It is worth noting that it is necessary to change the GA proposed in [2] and the PSO-GSA approach in [26] for comparing the proposed methods' efficiencies. These two approaches were proposed to decline the safety property of the systems specified through GTS by detecting a deadlock state. We just replace their fitness function with the one presented in 4.2 to use them to verify the reachability properties.

Experimental results obtained from an average of 20 independent runs and were presented in two tables for each case study. Average running times were reported in the first table and the detailed results were presented in the second one.

A. Dinning philosopher's problem

This problem was first introduced by E.W. Dijkstra. In this problem, several philosophers are sitting around a table. There is a fork between each pair of adjacent philosophers. After thinking, philosophers get hungry. Each philosopher picks up the left and right forks to use them for eating. A hungry philosopher should have both the left and right forks to start eating. after eating, the philosopher puts the left and the right forks on the table and thinks again. This process continues until the deadlock situation happens when all philosophers are waiting for their right fork as they have picked up their left forks [39]. This state is the reachability property checked in different kinds of this problem. The results obtained by performing the proposed approaches for verifying this problem's reachability property can be found in Table 4.

B. Pac-Man game problem

In the Pac-Man game, three types of objects are defined: Pac-Man, marbles, and ghosts [40]. According to the rules of the game, the Pac-Man and the Ghost can move to an adjoining box in each stage.

TABLE IV. THE COMPARISON OF RUNNING TIMES OF ALL APPROACHES TO VERIFY THE REACHABILITY PROPERTY IN THE DINING PHILOSOPHERS' PROBLEM

Number of philosophers	Depth Limit	Population	FAPSO (sec)	GA (sec)	PSO (sec)	PSO-GSA (sec)
10	50	15	2.81	6.27	8.12	7.06
20	100	20	29.2	22	85	68
25	150	40	38.52	41	112	90
30	200	60	49.86	91	137	109

If Pac-Man moves to a new box and there is a marble in, he can eat that marble. However, if the ghost moves to an adjoining box when Pac-Man is in the box, the ghost kills Pac-Man. The game ends when all marbles are eaten or Pac-Man is killed by a ghost. In this problem, the following reachability property should be checked: Pac-Man is the winner and eats all apples.

The results obtained by different approaches for the Pac-Man game problem can be found in Table 5.

TABLE V. THE COMPARISON OF RUNNING TIMES OF ALL APPROACHES TO VERIFY THE REACHABILITY PROPERTY IN THE PAC-MAN PROBLEM

Dimension of Pac-man Game	**Depth Limit**	**Population**	**FAPSO (sec)**	**GA (sec)**	**PSO (sec)**	**PSO-GSA (sec)**
4×4	100	40	4.13	4.88	15.07	12.49
4×5	100	60	7.96	11.15	36.79	27.31
5×6	100	80	17.59	72.03	59.1	60.26

The results related to this problem demonstrate that the FAPSO approach takes a shorter time to find the given reachability property and decreases the number of explored states significantly.

C. Process Life Cycle problem

Process Life Cycle describes the stages related to the life cycle of a process traversing in an OS. This cycle starts with creating a new process. Then, it is loaded into the memory providing that enough free memory is available. Afterward, it waits for I/O devices or CPU. After the complete execution of the process, all allocated resources are released and the process stops. The reachability property that should be checked in the models of this problem is: All existing processes have been completed. The results obtained by different approaches for the Process life cycle problem can be seen in Table 6. As seen in this table, FAPSO Approach, unlike other proposed Approaches can find the given reachability property even with increased dimensions of this problem.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 October 2020 doi:10.20944/preprints202010.0288.v1

TABLE VI. THE COMPARISON OF RUNNING TIMES OF ALL APPROACHES TO VERIFY THE REACHABILITY PROPERTY IN THE PROCESS LIFE CYCLE PROBLEM

Process Life Cycle	Depth Limit	Population	FAPSO (sec)	GA (sec)	PSO (sec)	PSO-GSA (sec)
20: process 8: memory	180	20	7.08	6.58	37.09	17.66
30: process 8: memory	280	40	7.28	8.16	37.61	16.92
40: process 8:memory	350	60	19.52	125.4	80.28	54.13
50: process 8: memory	450	80	40.95	Out of Memory		

D. *Shopping problem*

This problem is related to the purchase process of the customers in a store originally presented in [41]. The reachability property considered here is: all customers have successfully finished their shopping. Tables 7 represents the results obtained by different approaches for the Shopping problem.

TABLE VII. THE COMPARISON OF RUNNING TIMES OF ALL APPROACHES TO VERIFY THE REACHABILITY PROPERTY IN THE SHOPPING PROBLEM

Shopping	Depth Limit	Population	FAPSO (sec)	GA (sec)	PSO (sec)	PSO-GSA (sec)
Shop-10-cus30-good	160	20	2.45	2.01	3.89	4.14
Shop-15-cus30-good	170	30	10.53	32.74	34.5	27.62
Shop-20-cus30-good	180	40	33.58	Out of Memory		

E. *N-Queen Problem*

An N×N chessboard and N queens are the elements of this problem. These queens should be placed on the chessboard so that no queen can guard another one. In the chess game, each queen can move horizontally, vertically, or diagonally as far as she wants, and two queens can guard each other as long as their rows, columns, or diameters are the same. In this regard, the acceptable arrangement is the situation that the columns, rows, and diameters of all queens are different. The reachability property considered in the different models of this problem is: *All queens have been placed in the locations where none of them can guard others*. Table 8 represents the results obtained by different approaches for the N-queen problem.

TABLE 8. THE COMPARISON OF RUNNING TIMES OF ALL APPROACHES TO VERIFY THE REACHABILITY PROPERTY IN THE N-QUEEN PROBLEM

N-Queen Dimension	Depth Limit	Population	FAPSO (sec)	GA (sec)	PSO (sec)	PSO-GSA (sec)
8 × 8	100	20	3.0 7	1.45	6.83	2.17
16 × 16	120	30	24.17	Out of Memory		

F. *8-Puzzel Problem*

In this problem, there is a nine-box board where eight boxes are filled by numbered tiles (from 1 to 8) and one is empty [42]. A tile can move to the empty box when it is adjoining to the empty cell. I this game, one should begin with an arbitrary configuration of tiles and try to arrange the numbers in ascending order.

Table 9 shows the results obtained by different approaches for this problem.

TABLE IX**.** THE COMPARISON OF RUNNING TIMES OF ALL APPROACHES TO VERIFY THE REACHABILITY PROPERTY IN THE 8-PUZZEL PROBLEM

Initial arrangement	Depth Limit	Population	FAPSO (sec)	GA (sec)	PSO (sec)	PSO-GSA (sec)
	100	40	2.77	1.79	11.37	12.45
	100	50	6.57	5.91	47.13	61.9
	100	60	34.93	26.3	126.03	276.35
	100	70	102.5	116.51	209.23	380.52

V. DISCUSSION

The proposed approaches' advantages and limitations are described as follows.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 October 2020 doi:10.20944/preprints202010.0288.v1

As explained in the literature, the previous approaches proposed detects a deadlock state for declining safety property of the GTS-specified systems. We applied the FAPSO approach for declining Safety property by verifying the right reachability property in dining philosopher's, process life cycle, and 8-puzzle problems and comparing the results with Evolutionary approaches proposed to refute safety properties through deadlock detection. Table 10 presents the comparison results.

TABLE X. COMPARING THE PERFORMANCE OF FAPSO APPROACH TO REFUTE SAFETY BY REACHABILITY PROPERTY WITH PROPOSED APPROACHES TO SAFETY REFUTATION BY DETECTING A DEADLOCK STATE

Approach / problem	Depth limit	Population	FAPSO	nBOA	GA	PSO	PSO-GSA	BAPSO	BFA	BS
10 dining philosophers	25	15	4.41	0.71 ± 0.15	10.12	13.45	38.92	8.34	0.94	3.85
20 dining philosophers	100	20	29.2	1.04 ± 0.15	23	158	170	64.6	1.9	4.12
process life cycle (40-process-8-memory)	350	60	19.52	1.44 ± 0.8	Not found	Not found	939.45	854	Not found	Not found
process life cycle 50:process8 :memory	450	80	40.95	1.81 ± 0.39	Not found	Not found	Not found	Not found	Not found	Not found
8-puzzel (Second Arrangement)	100	50	6.57	1.15 ± 0.33	35.81	94.72	16.7	45.53	0.16	1.33
8-puzzel (Third Arrangement)	100	60	34.93	Not found	165	165.51	147.7	70.93	3.5	2.33

Another important advantage of the FAPSO approach is its accuracy. In fact, the higher the number of successful runs, the higher the accuracy. The proposed approaches' accuracy in terms of the reachability verification of the given properties in all mentioned problems has been compared through the chart in Figure. 8.

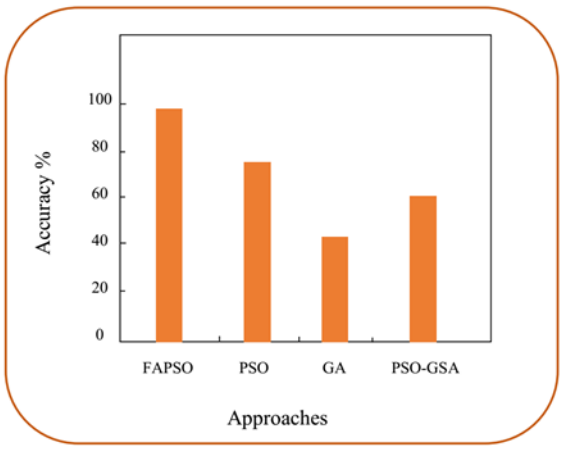

Figure 8. Comparing the accuracy of the proposed approaches

As the previously presented results indicate, the execution speed of these approaches, especially the FAPSO, is more than that of others.

Additionally, the proposed approaches generate shorter counterexamples/witnesses in comparison with other approaches. The chart in Fig. 9 compares the length of the witnesses created by different approaches to verify reachability in the dining philosopher's problem—in the case of 10 philosophers.

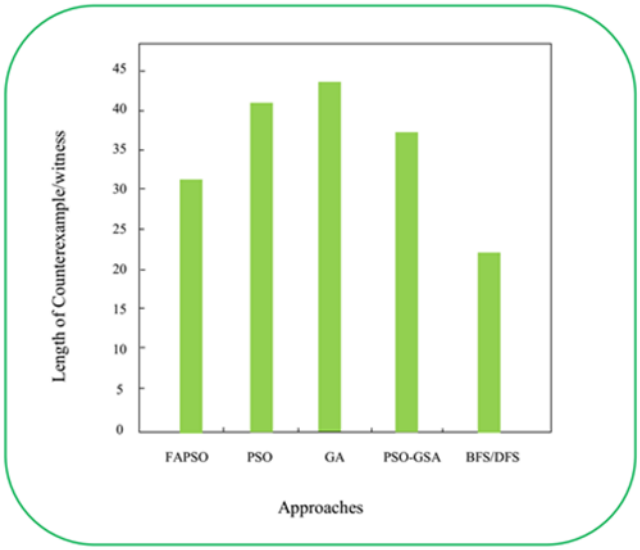

Figure 9. Comparing the length of counterexample/witness of the proposed approaches to reachability verification in the dining philosophers' problem with 10 philosophers

It should be noted that generating the shorter counterexamples/witnesses is very important in the process of model checking. So, the generated counterexamples/witnesses' length in the proposed approaches were controlled using the depth limit parameter.

The proposed approaches, especially the FAPSO, outperform other approaches in terms of the number of explored

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 14 October 2020 doi:10.20944/preprints202010.0288.v1

states. The number of states explored by different approaches to verify the reachability property considered for the dining philosopher's problem—in the case of 10 philosophers—can be compared through the chart in Figure 10.

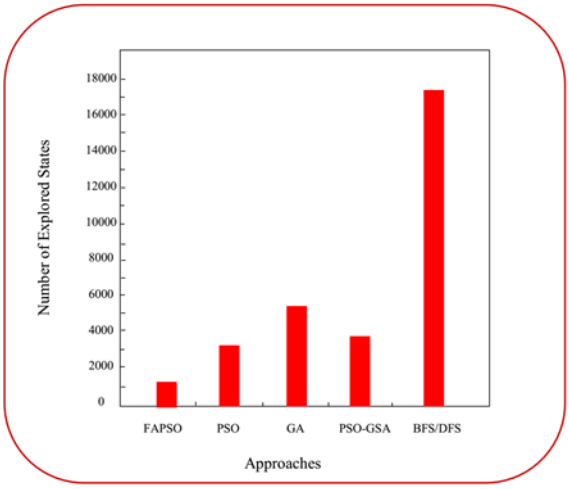

Figure 10. Comparing the number of explored states of the proposed approaches to reachability verification in the dining philosophers' problem with 10 philosophers

There are some limitations to employ the PSO-based Approach proposed in this paper. One of the essential limitations is to choose the best value for C1 and C2. To deal with this limitation, we proposed the FAPSO approach taking advantage of a fuzzy system to calculate C1 and C2 effectively. However, the FAPSO approach also has some limitations. The time spent to evaluate the population and calculate the inputs of the fuzzy system increases in the large complex systems having numerous transformation rules.

VI. CONCLUSIONS

This paper presents two practical approaches in order to verify a reachability property aiming to find a configuration that its occurrence refutes the safety property. The primary purpose of these approaches is to manage problem of the state space explosion in model checking of complex software systems that are specified GTS. In this solution, a PSO-based algorithm and a Fuzzy Adaptive PSO-based algorithm are proposed to explore the state space intelligently and find a target state in which the reachability property is satisfied. These two algorithms are implemented in the GROOVE toolset, and some parts of the source code are modified by Java programming language. We also implemented two other techniques based on GA and PSO-GSA algorithms to evaluate the efficiency of our approaches. The experimental results indicate the fact that the FAPSO approach is generally faster and more accurate than GA, PSO, PSO-GSA approaches, especially when the size of the problem increases. Also, fewer numbers of explored states and shorter lengths of counterexample/witness are other advantages of the FAPSO solution in comparison with the PSO-based approach.

Further researches can be done by changing the fuzzy system to improve the efficiency of the algorithm. Applying the proposed fitness function to verify reachability in other approaches proposed to detect deadlock can be considered as future works.

TABLE XI. GLOSSARY OF ACRONYMS

Genetic algorithm	GA
Particle swarm optimization	PSO
Graph ttransformation system	GTS
Gravitational search algorithm	GSA
Design space exploration	DSE
Operating System	OS
Ant Colony Optimization	ACO
Fuzzy inference system	FIS

REFERENCES

[1] J. Denil, M. Jukss, C. Verbrugge, and H. Vangheluwe, "Search-Based Model Optimization Using Model Transformations," in System Analysis and Modeling: Models and Reusability, Springer International Publishing, pp. 80–95, 2014.

[2] R. Yousefian, V. Rafe, and M. Rahmani, "A heuristic solution for model checking graph transformation systems," Applied Soft Computing, vol. 24, pp. 169–180, Nov. 2014.

[3] L. Baresi and R. Heckel, Tutorial introduction to graph transformation: A software engineering perspective, In Graph Transformation (ICGT), 2002.

[4] A.L. Lafuente, Symmetry reduction and heuristic search for error detection in model checking, In Workshop on Model Checking and Artificial Intelligence, 2003.

[5] E.M. Clarke, K.L. McMillan, S.V.A. Campos and V.I. Hartonas-Garmhausen, Symbolic model checking, In Computer Aided Verification 1102 , pp. 419-422, 1996.

[6] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill and L.J. Hwang, Symbolic model checking: 1020 states and beyond. Information and Computation 98(2), pp. 142-170, 1992

[7] S. Edelkamp, S. Leue and A.L. Lafuente, Directed explicit-state model checking in the validation of communication protocols, International Journal on Software Tools for Technology Transfer (STTT) 5, pp. 247–267, 2004.

[8] D. Boˇsnaˇcki, S. Leue and A.L. Lafuente, Partial-order reduction for general state exploring algorithms, International Journal on Software Tools for Technology Transfer (STTT) 11 , pp. 39–51, 2009.

[9] E.A. Emerson and A.P. Sistla, Symmetry and model checking, formal methods in system design 9 , pp. 105–131, 1996.

[10] E.M. Clarke, R. Enders, T. Filkorn and S. Jha, Exploiting symmetry in temporal logic model checking, Formal Methods in System Design 9 , pp. 77–104, 1996.

[11] V. Gyuris and A.P. Sistla, On-the-fly model checking under fairness that exploits symmetry, Formal Methods in System Design 15 , pp. 217–238, 1999.

[12] V. Rafe, Scenario-driven analysis of systems specified through graph transformations, Visual Languages and Computing 24(2), pp. 136–145, 2013.

[13] F.J. Lin, P.M. Chu and M.T. Liu, Protocol verification using reachability analysis: the state space explosion problem and relief strategies, In ACM Workshop on Frontiers in Computer Communications Technology, ACM New York, pp. 126–135, 1987.

[14] C.H. Yang and D.L. Dill, Validation with guided search of the state space, In DAC'98 Proceedings of the 35th Annual Design Automation Conference. ACM New York, pp. 599-604, 1998.

[15] S. Edelkamp and F. Reffel, OBDDs in Heuristic Search, In Lecture Notes in Computer Science. Springer-Verlag, pp. 81–92, 1998.

[16] G. Friedman, A. Hartman, K. Nagin and T. Shiran, Projected state machine coverage for software testing, In ACM SIGSOFT International Symposium on Software Testing and Analysis. ACM New York, pp. 134–143, 2002.

[17] V. Rafe, A.T. Rahmani, L. Baresi and P. Spoletini, Towards automated verification of layered graph transformation specifications. IET Software 3 , pp. 276–291, 2009.

[18] G. Francesca, A. Santone, G. Vaglini, M.L. Villani, Ant Colony Optimization for Deadlock Detection in Concurrent Systems, In 35th IEEE Annual Computer Software and Applications Conference, Munich, Germany, pp. 108-117, 2011.

[19] E. Alba and F. Chicano, Ant Colony Optimization in Model Checking, In 11th international conference on Computer Aided Systems Theory, Gran Canaria, España, pp. 523-530, 2007.

[20] L.M. Duarte, L. Foss, R. Wagner and T. Heimfarth, Model Checking the Ant Colony Optimisation, In Distributed, Parallel and Biologically Inspired Systems, IFIP Advances in Information and Communication Technology, Springer, pp. 221-232, 2010.

[21] E. Alba and J.M. Troya, Genetic Algorithms for Protocol Validation, In International Conference on Parallel Problem Solving from Nature PPSN IV, Springer, Berlin, Heidelberg, pp. 869-879, 1996.

[22] A. Vardhan, Learning to Verify Systems, University of Illinois at Urbana-Champaign, 2006.

[23] G. Francesca, A. Santone, G. Vaglini and M.L. Villani, Ant Colony Optimization for Deadlock Detection in Concurrent Systems, In 35th Annual Computer Software and Applications Conference, IEEE, Munich, Germany, pp. 108-117, 2001.

[24] E. Alba and F. Chicano, Finding safety errors with ACO, In 9th annual conference on Genetic and evolutionary computation, pp. 1066-1073, 2007.

[25] E. Alba and F. Chicano, Searching for Liveness Property Violations in Concurrent Systems with ACO, In 10th Annual Conference on Genetic and Evolutionary Computation. ACM, Atlanta, Georgia, USA, 2008.

[26] M. Moradi, V. Rafe, R. Yousefian and A. Nikanjam, A Meta-Heuristic Solution for Automated Refutation of Complex Software Systems Specified through Graph Transformations. Applied Soft Computing 33, pp. 136-149, 2015.

[27] R. Yousefian, S. Aboutorabi and V. Rafe, A greedy algorithm versus metaheuristic solutions to deadlock detection in Graph Transformation Systems, Journal of Intelligent & Fuzzy Systems 31(1), pp. 137–149, 2016.

[28] E. Pira, V. Rafe and A. Nikanjam, EMCDM: Efficient Model Checking by Data Mining for Verification of Complex Software Systems Specified through Architectural Styles, Applied Soft Computing 44 , pp. 1185-1201, 2016.

[29] E. Pira, V. Rafe and A. Nikanjam, Deadlock detection in complex software systems specified through graph transformation using Bayesian optimization algorithm, Journal of Systems and Software 131, pp. 181-200, 2017.

[30] H. Abdeen, D. Varró, H. Sahraoui, A.S. Nagy, Á. Hegedüs and Á Horváth, Multi-Objective Optimization in Rule-Based Design Space Exploration, In 29th ACM/IEEE international conference on Automated software engineering. Vasteras, Sweden, pp. 289-300, 2014.

[31] M. Fleck, J. Troya, and M. Wimmer, Search-based model transformations, Journal of Software: Evolution and Process 28(12), pp. 1081-1117, 2016.

[32] A. Rensink, Á. Schmidt and D. Varró, Model Checking Graph Transformations: A Comparison of Two Approaches, In International Conference on Graph Transformation. Springer, Berlin, Heidelberg, pp. 226-241, 2004.

[33] R. Heckel, Graph Transformation in a Nutshell, Electronic Notes in Theoretical Computer Science (ENTCS) 148(1), pp. 187-198, 2006.

[34] L. Baresi and R. Heckel, Tutorial introduction to graph transformation: A software engineering perspective, In The First International Conference on Graph Transformation (ICGT), Springer, pp. 402-429, 2002.

[35] J. Kennedy, Particle Swarm Optimization. Encyclopedia of Machine Learning. Springer, pp. 760-766, 2010.

[36] H. Kastenberg and A. Rensink, Model Checking Dynamic States in GROOVE, In International SPIN Workshop on Model Checking of Software. Springer, Berlin, Heidelberg, pp. 299-305, 2006.

[37] P. Melin, F. Olivas, O. Castillo, F. Valdez, J. Soria, and M. Valdez, Optimal design of fuzzy classification systems using PSO with dynamic parameter adaptation through fuzzy logic, Expert Systems with Applications 40(8), pp. 3196-3206, 2013.

[38] F. Olivas and O. Castillo, Particle Swarm Optimization with Dynamic Parameter Adaptation Using Fuzzy Logic for Benchmark Mathematical Functions, In Recent Advances on Hybrid Intelligent Systems. Springer, Berlin, Heidelberg, pp. 247-258, 2013.

[39] A. Schmidt, Model Checking of Visual Modeling Languages, Master's Thesis, Budapest University of Technology, Hungary, 2004.

[40] R. Heckel, Graph Transformation in a Nutshell, Electronic Notes in Theoretical Computer Science (ENTCS) 148(1), pp. 187-198, 2006.

[41] J.H. Hausmann, Dynamic Meta Modeling: A Semantics Description, Technique for Visual Modeling Techniques, Ph.D. Dissertation University of Paderborn, Germany, 2005.

[42] J. Gaschnig, Performance measurement and analysis of certain search algorithms, Carnegie Mellon University technical report, Pittsburgh, PA, USA, 1979.

[43] Salimi et al. Fuzzy Genetic Algorithm Approach for Verification of Reachability and Detection of Deadlock in Graph Transformation Systems, CANDO Conference, Budapest , Hungary, 2020.