IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 11 May 2023, accepted 22 May 2023, date of publication 3 July 2023, date of current version 26 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3291412

RESEARCH ARTICLE

An Incremental Optimization Algorithm for Efficient Verification of Graph Transformation Systems

FARANAK NEJATI[1], NOR ASILAH WATI ABDUL HAMID[2,3], (Senior Member, IEEE), SINA ZANGBARI KOOHI[2,3], AND ZAHRA RAHMANI ZADEH[4]

[1]Lee Kong Chian Faculty of Engineering and Science (LKC FES), Department of Internet Engineering and Computer Science (DIECS), Universiti Tunku Abdul Rahman, Sungai Long, Selangor 43000, Malaysia
[2]Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM), Serdang, Selangor 43400, Malaysia
[3]Institute for Mathematical Research, Universiti Putra Malaysia, Serdang, Selangor 43400, Malaysia
[4]Department of Software Engineering, University of Applied Sciences and Technology (UAST), Arak 14188-64511, Iran

Corresponding authors: Nor Asilah Wati Abdul Hamid (asila@upm.edu.my) and Sina Zangbari Koohi (zangbari@gmail.com)

This research was supported by Universiti Putra Malaysia (UPM) under Geran Putra GP/2020/9693400.

ABSTRACT This paper proposes an incremental optimization framework for verifying graph transformation systems to overcome the state space explosion (SSE). SSE refers to the exponential growth of the number of possible states in a system during its verification. The framework maps the verification problem to a search problem and incrementally generates the state space. The generated increments can still be significant in size, thus we use the Raccoon Optimization Algorithm (ROA), to non-exhaustively search through the state space. ROA selects sequences of states with a higher potential of having deadlock in increments, which prevents SSE and ensures that memory capacity is not exceeded. However, there is possibility that the migration method of ROA lead to a loss of diversity in the population, reducing the algorithm's ability to explore new regions of the search space. To address this issue, we propose a new migration method for ROA, called Improved ROA (IROA), which preserves diversity in the population and reduces execution time and the risk of getting stuck in local optima. Our approach is evaluated using the Groove simulation tool and compared with other relevant meta-heuristic algorithms in terms of computation time and memory consumption. The experimental results show that IROA outperforms both ROA and other relevant meta-heuristic algorithms that we compared in terms of computation time and memory consumption, with total efficiency of 1.043 and 1.02, respectively, demonstrating its effectiveness in verifying massive state spaces without facing state space explosion in a reasonable time.

INDEX TERMS Artificial intelligence, raccoon optimization algorithm, software verification, model checking, graph transformation systems, state space explosion.

I. INTRODUCTION

Graph transformation systems (GTS) have emerged as a useful mathematical language for software development. GTS are employed for various stages of software development, such as meta-modeling [1], software architecture [2], goal modeling [2], [3], model-based development [2], programming languages [4], performance analysis [5]. To improve the reliability in GTS, model checking [6] is an effective technique. However, the verification of GTS by model checking is a challenging task, as it involves exploring the state space of the system, which can be prohibitively large due to the state space explosion (SSE) problem. The challenge arises when the size of the state space, i.e., the set of all possible states of a system, grows exponentially as a function of the system's size or the level of abstraction in the model. This exponential growth makes it computationally infeasible to explore the entire state space, even for small-sized systems. Thus, it is

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

quite valuable to propose an efficient method to mitigate the SSE problem and successfully verify GTS.

To address the SSE problem, this paper proposes an incremental optimization framework that utilizes the Raccoon Optimization Algorithm (ROA) [7] to iteratively generate and verify the search space. In this approach, the algorithm incrementally generate and selects the states to explore, thus avoiding the exhaustive verification of the entire state space.

The reason we utilized ROA is that this algorithm, in each iteration, snatches a subset of SS to be explored and checked which is essential for the incremental approach. Furthermore, ROA has two different populations in two different zones of the state solution, which makes it more efficient than other optimization algorithms in terms of skipping local optima. However, sticking to local optima is inevitable and may occur. We also present an improved version of the ROA, named IROA, to further reduce the probability of getting stuck in local optima.

The motivation for proposing this method is to provide an approach to successfully verify GTS without facing the SSE problem. Traditional model checking techniques require exhaustive exploration of the entire state space, which is infeasible for large and complex systems. By utilizing an incremental approach along with IROA, our method can overcome the SSE problem and verify GTS models. This will lead to higher quality and more reliable systems, making our method a crucial tool for system development.

The paper's key contributions are twofold. First of all, we use GTS to map the verification process into a search problem, apply a novel meta-heuristic algorithm to search non-exhaustively and incrementally into the problem, and compare the results with a set of different meta-heuristic algorithms. In addition, we devise a strategy to improve ROA and find deadlock-freedom with efficiency and accuracy. The proposed method has been implemented in GROOVE [8] which is a simulation tool for the GST specification and generating its state space [9]. It has been applied to non-trivial case studies to demonstrate the feasibility, efficiency, and accuracy. The verification properties that is address in this paper is deadlock freedom.

To provide a roadmap for the paper, in Section II, we provide a background and preliminary findings related to the research. Section III presents related work and compares our approach with existing methods. In Section IV, we explain our proposed method in detail, including the extension of ROA and the mapping of the verification problem into a search problem. Section V describes the evaluation of our method through case studies. In Section VI, we discuss the results, limitations, and future work. Finally, in Section 7, we conclude the paper.

II. PRELIMINARIES

A. MODEL-CHECKING

Model checking is a renowned method for ensuring the correctness of a system. It clarifies the system specification and ensures accuracy. Model-checking can be defined as the following.

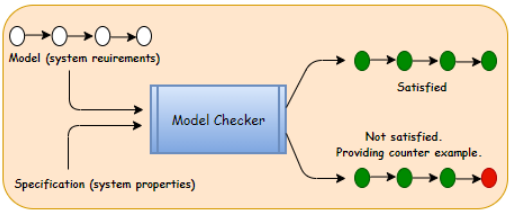


FIGURE 1. Model checking approach.

Definition 1: Model checking is a tuple (M, m, S), where:

- M is a conceptual model for expressing and representing a system requirement with m states;
- S is formal specification of a system properties;

Figure 1 represents the model checking approach. The system's models are represented by a Kripke structure [10], however, they can be represented by any graph-like visualization, such as GTS [11] or Petri Net [12]. The formal specification of the system is formulated by propositional temporal logic. This kind of logic is for describing and reasoning about the system's properties in terms of time. Some common system properties are reachability, safety, liveness, fairness [13]. In this paper, we aim to check deadlock freedom, which is a kind of safety property. The safety property guarantees that under particular circumstances, a violation (e.g.deadlock state) will not happen in any way. To check safety or other properties through model checking, all the SS should be generated and checked. However, the generation of the entire SS of the systems faces the state space explosion (SSE). To explain the severity of SSE, let's consider a system with n processes, where each process has m states. Then, the calculation of the size of SS can reach up to m^n. It obtains that by a large number of n and m the amount of state space in the system increases exponentially, therefore surplusing the memory volume. For more information about the methods of model checking, interested researchers can refer to [13].

B. GRAPH TRANSFORMATION SYSTEM

Graph transformation system is a graphical representation for system modeling with a formal and accurate mathematical foundation [14]. GTS has been advanced to deal with non-linear structures since the first protocols were made in the late 1960s and early 1970s [15]. Since GTS can comprehensively define the behavior of the system and describe the concepts of all of its components along with their relationships with graphs, it can also be used to check the system's correctness through model checking. A formal definition of GTS is provided below.

Definition 2: Graph transformation is a triple $GTS = (TG, HG, R)$, which are examining each other.

- HG is a host graph;
- TG is a type graph;
- R is a set of rules.

The primary purpose of HG is to display the system's initial configuration. The reason is that in the initial phase, a uniform graph from HG to TG is required so that every edge and node in the host graph are mapped to single edge types and single node types in the TG. The type graph TG can be defined formally as follows:

Definition 3: Type graph is a tuple $AGT = (TN, TE, src, des)$ in which:

- TN is the set of node types;
- TE is the set of edge types;
- src is the source of a node;
- des is the destination of a node.

Each TE links a TN as a source to another TN as the destination by two different functions src and des. R is a set of rules which each rule can be define formally as the following.

Definition 4: Rule in GTS is a triple $AGT = (LHS, RHS, NAC)$ in which:

- LHS referred the left hand side of the rule which served as the prerequisite for the rule to be valid;
- RHS referred to the right hand side of the rule;
- NAC referred to the Negative Application Condition.

Numerous tools have been created to manipulate and model GTS. Among the existing tools, the only one that supports model checking is Groove [16]. Additionally, the Groove is also an open-source tool, which enables researchers to easily add new features.

C. RACCOON OPTIMIZATION ALGORITHM

Meta-heuristic algorithms have found widespread applications across various fields [17], [18]. ROA is one such algorithm that distinguishes itself by having two distinct populations in two different zones of the state solution. This unique feature makes it more efficient than other optimization algorithms in terms of skipping local optima. Given this advantage, we have selected ROA as the basis for our proposed method to improve verification in GTS.

The remarkable hunting ability of raccoons, coupled with their sharp claws and keen eyesight, has served as the inspiration behind a new approach to achieving the global optimum in optimization problems, known as the ROA method [7]. In this method, the fitness function domain is modeled after the natural habitat of the raccoon, where food sources are scattered throughout, offering a variety of potential options to explore. The goal is to identify the most optimal solution among all the possibilities available.

The ROA process comprises three distinct phases: parameter definition, initialization, and the main loop. To simplify the overall computational procedure of the ROA, a pseudocode has been developed (see Pseudocode 1), outlining each of the individual stages of the algorithm.

To ensure optimal performance, the parameters of an algorithm should be tailored to the specific problem at hand. In the case of ROA [7], there are six distinct parameters, which are summarized in Algorithm 1. Once these parameters have been established, the next step is initialization,

Algorithm 1 ROA

Parameters Definition:
$RZR \leftarrow$ Radius of Reachable Zone (RZ)
$N_{reachable} \leftarrow$ No. of RZ Candidates
$VZR \leftarrow$ Radius of Visible Zone (VZ)
$N_{visible} \leftarrow$ No. of VZ Candidates
$NI \leftarrow$ No. of Iterations
$MF \leftarrow$ Factor of Migration
Initialization:
$pos_0 \leftarrow$ Random Initial Location
$G_{opt} \leftarrow pos_0$
$RZP_0 \leftarrow$ Initial Reachable Population (RP)
$R_{best_0} \leftarrow$ Best Candidate in RZP_0
$VZP_0 \leftarrow$ Initial Visible Population (VP)
$V_{best_0} \leftarrow$ Best Candidate in RZP_0
$n_{persev} \leftarrow 0$
Main Loop:
if $f(pos_{NI}) > f(G_{opt})$ **then**
    Return pos_{NI}
else
    Return G_{opt}
end if
for $i = 1$ to NI **do**
    $pos_i \leftarrow$best position in pos_{i-1}, $RP_{best_{i-1}}$, and $VP_{best_{i-1}}$
    if $f(pos_i) > f(G_{opt})$ **then**
        $G_{opt} \leftarrow pos_i$
    end if
    if $pos_i = pos_{i-1}$ **then**
        $n_{pereserv}++$
    else
        $n_{pereserv} = 0$
    end if
    if $n_{pereserv} = MF$ **then**
        $pos_i \leftarrow$new position (random) not inside of VZP_{i-1}
        $n_{pereserv} = 0$
    end if
    $RZP_i \leftarrow$ RP closed pos_i
    $R_{best_i} \leftarrow$ Best Candidate in RZP_i
    $VZP_i \leftarrow$ VP closed pos_i
    $V_{best_i} \leftarrow$ Best Candidate in VZP
end for
Return V_{best_i}

which can have a significant impact on the algorithm's overall performance. Algorithm 1 provides an overview of the initialization phase in ROA.

ROA involves two distinct populations: the zone of reachable population (RZP), which is accessible from the raccoon's paws, and the zone of visible population (VZP), which is accessible from the raccoon's eyes. The final phase of ROA is the main loop, which consists of several processes that work together to achieve the optimal solution. These processes

include changing the raccoon's position, migration, and generating the next generation. During each iteration of the main loop, the following values are selected:

- pos_{i-1}: the raccoon's current position;
- $RP_{best_{i-1}}$: refers to the reachable population (the preceding iteration);
- $VP_{best_{i-1}}$: refers the best value in the visible population (the preceding iteration).

Then, the raccoon changed its position to the best position among the aforementioned values.

In particular circumstances, preservation of the result may happen. The preservation occurs when the raccoon gets stuck in a local optima. After a user defined number of iterations, called the ''Factor of Migration'', the condition for preservation will be checked.

In the event that preservation occurs, the raccoon migrates to a new position. For this reason, new pos_i and G_{opt} are computed, and the one with best fitness will be the next position.

$$G_{opt} = (f(G_{opt}) > f(pos_i)) \rightarrow (G_{opt}) \wedge \neg(f(G_{opt}) > f(pos_i)) \rightarrow (pos_i) \quad (1)$$

III. RELATED WORK

Model checking has proven to be an effective method for ensuring the correctness of complex systems. However, the state space explosion problem (SSE) remains a major challenge, limiting the scalability and applicability of model checking to verification of models such as GTS. To overcome this challenge, researchers have developed a range of methods, include scaling down the state space [19], compositional verification [20], memory handling [21], bottom-up verification [22], and heuristics and probabilistic [23], [24], [25], [26], [27] approaches. In this section, we review the strengths and drawbacks of some of the improved optimization algorithms in improving the performance of graph transform systems (GTS).

Probabilistic algorithms can be considered a type of optimization algorithm because they use probability or randomness to explore the search space and find the optimal solution or an approximation of it. In probabilistic algorithms, the solution is not found with certainty, but instead, the algorithm tries to find the solution with a high probability. This approach can be useful when the search space is large or complex, and deterministic algorithms are not feasible or efficient. Therefore, probabilistic algorithms can be useful for optimization problems, including those in model checking and verification. One of the basic probabilistic algorithms is random walk [28], which specifies a path in the state space that contains a sequence for finding defects in model-checking. The algorithm has minimal memory requirements, but its complexity is not superior to existing methods for solving SSE [29]. Another basic probabilistic approach is the breadth-first search algorithm, which is utilized for protocol validation and makes significant improvements over depth-first search [29]. However, for larger systems, it may still not be practical to exhaustively search the entire state space using this approach. Additionally, the use of other techniques may still be necessary to make the verification process more efficient.

Bloom filter is another probabilistic approach for verification purposes [30], employed in two basic schemes for probabilistic verification: hash compaction and bit-state hashing. Bloom filtering saves compressed values in a data hash table instead of maintaining complete state descriptors. In this method, states with a non-zero probability will be eliminated throughout the verification process. As a result, some attainable states may get unchecked during the verification process, which may lead to false-positive results.

The continuous-time Markov chain (CTMC) [31], [32] is also used in model checking as a probabilistic approach to deal with SSE. CTMC models are an extension of Markov chains that enable the modeling of continuous-time events. These models are widely used to model and analyze the performance and reliability of systems with random behavior. CTMCs have been applied in various domains, such as communication networks, manufacturing systems, and biological systems. However, CTMCs have some limitations that should be considered when using them for verification and model checking. For example, the analysis of CTMCs can be computationally expensive, especially when the system has a large number of states. Therefore, researchers need to develop efficient algorithms and techniques to mitigate these limitations when using CTMCs for verification and model checking.

One class of methods used in probabilistic approaches are meta-heuristic algorithms, which are designed to provide approximate solutions to optimization problems. Among the various meta-heuristic algorithms, the Genetic Algorithm (GA) has been widely applied in the field of model-checking [33], [34]. However,GA can suffer from premature convergence, where the algorithm terminates before finding the optimal solution. Additionally, the computational complexity of the genetic algorithm can be high for large-scale problems, making it less suitable for massive verification tasks. In an effort to improve the efficiency of GA, a hybrid approach combining Genetic Algorithms with Assume-Guarantee reasoning, a subset of compositional verification, is proposed in [35].

Another popular meta-heuristic technique is the Ant Colony Algorithm, which has been utilized for identifying optimal pathways in graph-like models [36]. This method has also been successfully applied to verification and model checking problems. One advantage is its ability to identify optimal pathways in graph-like models, which makes it suitable for solving problems in various domains. It has been successfully applied to verification and model checking problems. However, one of its drawbacks is the lack of diversity in its solutions, which can result in premature convergence and suboptimal solutions. Another disadvantage is its sensitivity to parameter settings, which can lead to different results for different parameter values. In a similar vein, to improve

the efficiency of Ant Colony Algorithm, a study presented in [37] introduces the use of randomized selection mechanisms through an ant colony algorithm to diversify solutions.

The meta-heuristic algorithm is also used in the verification of GTS [38]. A framework has been devised in [39] for applying a non-exhaustive search to investigate system properties modeled by GTS. They present a heuristic method for reducing the verification effort and providing faster results. R. Yousefian et al. [40] investigated the application of GA for model checking through GTS. They claim that the problem of SSE can be mitigated by creating an incomplete SS each time and searching for deadlock in that partial path. The Bayesian optimization algorithm (BOA) [41], [42] is another of the successful algorithms that has been used to verify GST-specified systems for deadlock freedom. Although the studies mentioned above demonstrate the potential of meta-heuristic methods to tackle the SSE problem, it is important to note that such methods may not always provide the optimal solution, and the quality of the obtained solution may be influenced by the choice of algorithm and its parameters. Furthermore, the efficiency and effectiveness of these methods are highly dependent on the characteristics of the problem instance. Hence, there is always scope for further improvement in the design and application of heuristic algorithms.

In light of these limitations, our study introduces an incremental optimization algorithm that generates and verifies the search space iteratively, rather than all at once. It strategically choosing the states to explore by using an improved version of raccoon optimization algorithm named IROA. The IROA can efficiently verify the state space without encountering the SSE. Therefore, our study contributes to the ongoing efforts to design and apply meta-heuristic algorithms that can overcome the limitations of existing methods and provide more accurate and efficient solutions.

IV. PROPOSED APPROACH

Software verification can be limited due to the state space explosion (SSE) problem, where the software's state space grows exponentially and surpasses the memory capacity during the verification process. Traditionally, a model checker generates the entire state space from a unique initial state s_0 and starts checking from s_0. Every state is checked, and the model checker determines whether the state satisfies the desired system properties or not. However, generating the whole state space at once leads to the state space explosion problem.

To overcome this problem, an incremental optimization framework can be used. This section presents such a framework that uses an improved version of the Raccoon Optimization Algorithm (ROA) called IROA to iteratively generate and verify the search space. The framework employs a non-exhaustive search on the state space using the Groove tool. The main aim of the method is to generate and check a partial state space in multiple iterations until the entire state space is verified, instead of generating the entire state space at once. This is known as incremental verification. The partial

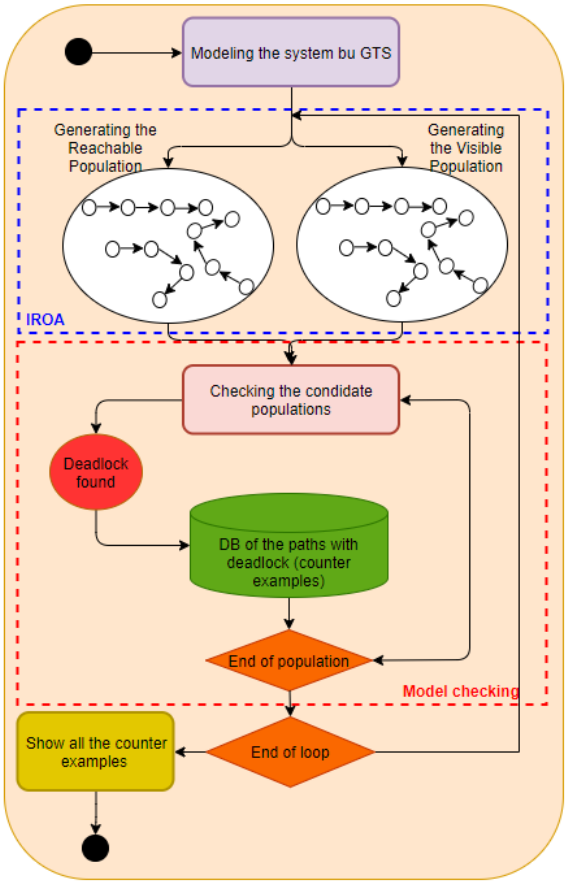


FIGURE 2. Proposed model checking based on IROA in Groove.

state space, as it still is massive, then will be checked by IROA to determine whether there is a property violation on any of the paths. In our scenario, the property violation is considered a deadlock.

By using IROA in the verification process, we can efficiently skip local optima, which makes it more efficient than other optimization algorithms. This leads to faster verification of the entire state space, which would not have been possible using traditional methods. Therefore, the proposed incremental optimization framework provides an efficient solution to the SSE problem in software verification.

The verification process of improved ROA with the Groove tool has several steps which are described in the subsections. The overview of the proposed verification process is displayed in 2. The figure does not contain the migration part. The following subsections explain the proposed framework in details.

A. MODELING THE SYSTEM USING GTS IN GROOVE TOOL

To verify a system based on GTS, firstly, the system under test should be modeled based on the syntax and semantics of the GTS and the Groove tool. A graph transform is used to model the system's behavior and the modeled system graphs are used to represent the SS of the systems.

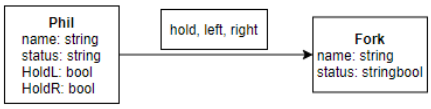


FIGURE 3. Attribute type.

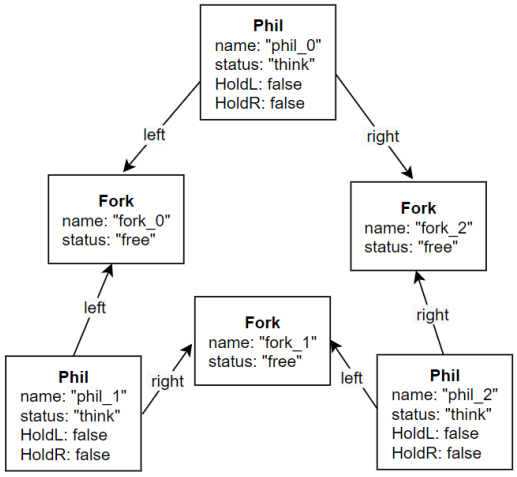


FIGURE 4. Host graph for dining philosopher in Groove.

Throughout this section, we use the dining philosopher problem as a running example to describe the proposed method clearly and demonstrate its suitability. The dining philosophers provide an example of how the SS might grow enormously as the number of philosophers grows. This conundrum features a group of philosophers (two or more) sitting at a table, who each think and become hungry for a while. To start, one of them may use the left fork first, then the right fork, and eat with both. The philosopher then enters deep thinking mode while placing the left and right forks on the table. This method can be repeated numerous times. Because there are several philosophers seated around the table and each fork is shared by the two philosophers next to them, they struggle to pick up the forks. The system will grind to a deadlock in some situations, such as when every philosopher chooses the left fork.

Figure 3, shows the Type graph of this problem in Groove tool with two type node $TN = \{Phil, Fork\}$ and three different type edge $TE = \{left, right, hold\}$. Figure 4, presents the host graph of this problem in the Groove tool. Philosophers in thinking status are in the initial state of the host graph. The next in GTS is defining rules for example the right-hand side of rules is $RHS = \{left, right, release\}$.

The SS of a system model represents its possible states during running the system, and in order to generate it, all the defined rules must be applied repeatedly to the initial state. In this research, a transition system is used to display the SS.

Definition 5 (transition system) A transition system (TS) can be defined as a tuple $S = (b_0, B, E, P)$, where:

- B is a set of system's states (b_0 is initial state);
- E indicates a set of edges ($e \in E$) for changing system's state which labeled by the set of rules R;
- Δ indicates a set of steps from one state to another by an edge, $\Delta \subseteq S \times E \times S$.

B. GENERATING POPULATION

In this paper, we propose a novel approach to verify graph transformation systems (GTS) by mapping the verification process into a search problem. Specifically, we represent the state space of the GTS model as a set of individuals in the population, with each individual generated by a trace function that can explore all possible states of the GTS model. We consider each trace as an individual in the population, and input this population into an optimization algorithm for searching for deadlock. To address the exponential growth of state space, our approach proposes generating the population incrementally. This involves generating a limited number of individuals in successive increments based on the memory capacity of the system by using the trace function. The generated population is then checked by IROA, and all unwanted population before generating the next increment will be deleted from memory. The next population is then generated and checked until the entire state space is covered.

To generate the initial trace, we first define the initial position of the Raccoon pos_0 based on the GTS model. Then, by tracing the TS at a fixed depth, we generate individuals for the population. Our trace function is defined as a process of exploring the TS by visiting each node and generating all possible states.

Definition 6 (trace) A *trace* of a TS which indicated by ζ is a sequence of steps attached with its edges. $\zeta = (b_i, (e_i), b_{i+1}), (b_{i+1}, (e_{i+1}), b_{i+2}), \ldots, (b_n, (e_n), b_{n+1}) \in \Delta$. In this research we use the notation $\zeta = b_i \xrightarrow{e_i} b_{i+1}, b_{i+1} \xrightarrow{e_{i+1}} b_{i+2}, \ldots, s_n \xrightarrow{e_n} b_{n+1}$. Also, we may use the notation $\zeta = b \xrightarrow{e_1, e_2, \ldots, e_n} b_{n+1}$ to show the trace path from b to b'.

Based on the algorithm strategy in this phase, the global optimum is the initial position, $G_{opt} = pos_0$. A set of random candidate solutions around the pos_0 that are considered the initial reachable population will be assigned to the RZP_0. This set is an increment from the whole state space. The candidate solutions are selected based on the sequence of transitions starting from s_0 generated by the trace function defined in definition 6. Every potential solution is regarded as an individual within the reachable population. The depth (length) of each trace are problem-dependent, and the user should define it based on the problem. The depth is measured by the number of states (NS) in each trace. The process of generating an initial population for reachable zone is used to provide an initial population for the visible zone. In Figure 5, the sequence of "b_0, b_6, b_{25}, b_{23}, b_5" can be considered an individual in the initial population. For the next iterations, the same strategy will be used to provide the random candidate solutions (increments).

C. FITNESS FUNCTION

For every individual candidate solution in RZP_0, the number of potential outgoing transitions (POT) will be calculated.

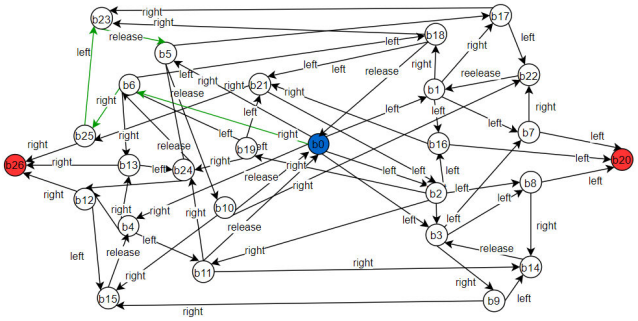


FIGURE 5. State space of dining philosophers.

By using POT, the model checker can evaluate whether the current path is a deadlock or not. As POT is a problem-dependent number, it is not trivial to find its maximum. For example, in the dining philosophers, each state could have a maximum up to N number of POT where N refers to the number of philosophers. The maximum POT of each problem is called the MPOT.

The aim is to find deadlock in the SS. A deadlock occurs when there is no POT in a given individual. In this circumstance, the system is stuck in one of the states and will not progress. Thus, an obvious fitness function can be the paths with a lower total POT in comparison with the MPOT of the current path $POT_{current\ path} < MPOT$. Thus, we have a minimization problem in which the algorithm should minimize the total POT of the path.

$$F(x) = \Sigma_{i=1}^{NS}\ POT_i$$

where NS is the number of allowed states in a trace. The best individuals in the population will be the ones with a lower POT number, named $POT\,risk$. Once a population is created, if its total fitness function is not satisfying, then some of the individuals with higher POT will be replaced by new $POT\,risks$.

Once evaluating the population is completed, the $POT\,risk$ inside the population will be selected as the initial best solution, $R_{best_0} = POT\,risk$. Then, a visible zone based on s_0 will be calculated. An initial visible population will be assigned to VZP_0. The method of population evaluation for the VZP_0 is the same as that RZP_0. After completing the evaluation of both populations, the next generation will be produced. This process will be repeated until the desired number of iterations is reached.

D. IMPROVED MIGRATION PROCESS

While ROA is effective at avoiding local optima due to its use of two different populations, migration can still be an issue. If migrated solutions are too similar to existing solutions in the target subpopulation, it can limit the algorithm's ability to explore new regions of the search space and find the global optimum. To address this issue, we propose a new method for escaping local optima that preserves diversity in the population. We discuss our proposed method in detail and show that it outperforms other methods in escaping local optima without sacrificing diversity.

In the proposed algorithm, the candidate solutions are generated incrementally. However, to monitor if a local optima has occurred, a number of increments will be buffered. If the similarity between the increments is greater than a predefined amount, then it will be determined as a local optima.

Once it has been determined that the Raccoon needs to migrate, a set of new individuals is given to a small group of randomly selected individuals in the population based on their outgoing transitions.

Three outcomes during raccoon migration should be followed. (1) to avoid choosing random states for the sequence. It is so that the selected random states are not a sequence of states with transitions to each other and (2) move in the same direction as the current individual. It means that using the transition of the provided individual to migrate will result in a series of states connected by transitions and the proper edge; (3) calculate the migration individuals for both reachable and visible zones. The migration will be decided based on the best option for both populations.

To start the migration process, we select a set of best individuals from visible zone and update its position according to the transitions in its states. The new individual can be calculated using the following formula.

$$CS = CS + \Sigma_{i=0}^{n}(\epsilon.(b_i) - b_{i,CS}) \tag{2}$$

where CS is the current position of the individual, ϵ is a random number produced by a function controlled by the Gaussian distribution to choose the transition for each state b_i, $b_{i,Cs}$ is the current position of the ith state, and b_i is the new position of the ith state in the new individual CS. A small set of individuals for migration will be provided, and the best one based on the fitness function will be selected to migrate.

E. THE PSEUDO-CODE AND COMPUTATIONAL COMPLEXITY

The basic structure of the Improved ROA algorithm is identical to the ROA presented in Algorithm 1. However, the migration phase has been modified. The pseudo-code for IROA, including the updated migration method, is provided in Algorithm 2. It should be noted that the input to this algorithm is not the entire state space. Instead, the state space will be created incrementally based on the method defined in section (B). The number of increments required depends heavily on the problem and memory capacity. Therefore, if the problem has a vast state space and limited memory, the number of increments containing candidate solutions will be higher.

The computational complexity of the Improved ROA algorithm has been analyzed and found to be $O(NS \cdot NP \log NP)$, where NS is the number of allowed states in a trace and NP is the population size. The analysis considered the initialization, evaluation, selection, migration, elitism, and termination phases of the algorithm. The initialization phase involves creating the initial populations and offspring,

Algorithm 2 IROA

Require:
RZR: Radius of Reachable Zone (RZ)
Nreachable: No. of RZ Candidates
VZR: Radius of Visible Zone (VZ)
Nvisible: No. of VZ Candidates
NI: No. of Iterations
MF: Factor of Migration
K: Number of Migration candidate
Ensure:
Gopt: Best Solution found

Randomly initialize $pos0$ and set $Gopt \leftarrow pos0$
Initialize $RZP0$, $Rbest0$, $VZP0$, $Vbest0$, and $npersev$
for $i = 1$ to NI **do**
  $posi \leftarrow$ best position in pos_{i-1}, $RP_{best,i-1}$, and $VP_{best,i-1}$
  if $f(posi) > f(Gopt)$ **then**
    $Gopt \leftarrow posi$
  end if
  if $posi = pos_{i-1}$ **then**
    $npersev \leftarrow npersev + 1$
  else
    $npersev \leftarrow 0$
  end if
  if $npersev = MF$ **then**
    Migration :
    Choose a set of individuals S from VZP_i
    for each s in S **do**
      Choose a random state b_i from s
      Calculate the new individual CS using:
      $CS = CS + \sum_{i=0}^{n}(\epsilon.(b_i) - b_{i,CS})$
      Evaluate the fitness of CS
      if CS is better than s **then**
        Replace s with CS
      end if
    end for
  end if
  $RZPi \leftarrow RP$ closed to $posi$
  $Rbesti \leftarrow$ Best Candidate in $RZPi$
  $VZPi \leftarrow VP$ closed to $posi$
  $Vbesti \leftarrow$ Best Candidate in $VZPi$
end for
return *Gopt*

which takes $O(NP)$ time. The evaluation phase calculates the POT and fitness function for each individual in the population, requiring $O(NS \cdot NP)$ time. The selection phase chooses NP individuals from the population based on fitness function, which takes $O(NP \log NP)$ time. The evaluation phase is repeated for the selected individuals, taking another $O(NS \cdot NP)$ time. The migration phase selects NP individuals from the selected individuals in VZP and requires $O(NP \log NP)$ time. The elitism phase keeps the best individual from the initial population, taking $O(1)$ time. The termination phase checks the stopping criteria, which takes $O(1)$ time. Overall, the Improved ROA algorithm has a computational complexity of $O(NS \cdot NP \log NP)$, which is dominated by the evaluation and selection phases. However, the number of increments in the state space creation process and the memory capacity may impact the number of increments and candidates, affecting the performance of the algorithm.

V. IMPLEMENTATION AND EVALUATION

This section aims to comprehensively evaluate the performance of the proposed meta-algorithms on four varied and well-known case studies, namely the dining philosophers [43], PacMan Game evaluation [44], N-Queen [39], Vehicle Platoon Evaluation [45]. Moreover, this section provides a detailed comparison of our proposed method with five other popular meta-heuristics methods. The evaluation is based on an average of different runs, and the summary of the parameters for each algorithm, including the improved ROA, is presented in the following section.

In order to verify each model, we have computed the average execution time for verifying the state space in the total execution runs. Additionally, we have analyzed the memory usage for each algorithm and monitored the occurrence of the SSE problem.

The execution time, memory usage, and occurrence of SSE problem are the primary evaluation metrics of the solutions generated by the proposed meta-algorithms on the four case studies. Therefore, we have not included a convergence diagram in this section as it is not necessary for the evaluation of the proposed meta-algorithms. The incremental deadlock identification process is the main focus of our study.

We also calculate the out-performance of IROA compared to ROA to demonstrate the effectiveness of the proposed approach. To find the average out-performance of IROA over ROA for all case studies, we use the speedup metric. The speedup metric measures the relative performance improvement of a new algorithm (in this case, IROA) compared to an existing algorithm (standard ROA) on the same problem. The formula for calculating speedup is:

$$speedup = P(R)/P(I) \tag{3}$$

where T(R) is the execution time of standard ROA, and T(I) is the execution time of IROA.

After calculating the speedup for all case studies, the efficiency of the IROA is found by adding up all the speedup values and dividing by the number of case studies. The formula for calculating speedup is:

$$efficiency = \Sigma speedup/No.\ of\ case\ studies \tag{4}$$

A. BENCHMARK

To evaluate the suggested algorithm, we examined four distinct, non-trivial known case studies. We used these case studies to examine other proposed optimization algorithms for solving SSE in GTS verification. There are multiple

optimization algorithms that provide a range of advantages and applications. However, most of them are highly problem-dependent and difficult to apply for GTS verification. As a result, we selected the generalized version of a few well-known algorithms to apply them in our incremental framework. These algorithms are ROA, GA [46], PSO [47], ABC [48], BOA [49]. We also compare the results with one of the famous model checkers named NuSMV [6].

The comparison of IROA has been conducted with the original versions of other popular algorithms. This serves as a baseline to demonstrate the effectiveness of the improvements made in IROA, showing how it outperforms the original ROA, GA, PSO, ABC, and BOA in terms of convergence rate, accuracy, and robustness. Additionally, the original versions of these algorithms have been widely used and bench-marked in various optimization problems, making a comparison with them a fair assessment of IROA's performance. Furthermore, the recent improvement on the meta-heuristic algorithms may have been for specific problem or data types. It may not provide a precise comparison as it may not perform well on other types of data or problems.

In our experimental approach, we utilized an initial parameter set to explore the state space, which was carefully selected based on the commonly used values in the literature. Given the crucial role of parameter selection in meta-heuristic algorithms and its potential impact on the accuracy and effectiveness of our proposed method, we conducted extensive testing to investigate the sensitivity of our results to different parameter configurations. Specifically, we tested by systematically varying one parameter at a time while keeping the others fixed, and measuring the resulting impact on the algorithm's performance. Our findings demonstrate that modifying these parameters had an insignificant impact on the experiment's results. To ensure consistency and fairness across all benchmarks, we fixed these parameter values for all evaluations. This approach allowed us to eliminate potential confounding factors and obtain reliable results.

The final parameters are listed below.

- ROA
  - RZR: 5
  - No. RZ Candidate: 10
  - VZR: 10
  - No. VZ Candidate: 3
  - No. Iterations: 30
  - Migration Factor: 3
- Improved ROA
  - RZR: 5
  - No. RZ Candidate: 10
  - VZR: 10
  - No. VZ Candidate: 3
  - No. Iterations: 30
  - Migration Factor: 3
- GA
  - Population Size: 100
  - Crossover Percentage: 0:7
  - Mutation Percentage: 0:2
  - Mutation Rate: 0:03
  - Tournament Size: 5
- PSO
  - Swarm Size: 100
  - Inertia Weight (IW): 0.99
  - IW Damping Ratio: 1.0
  - Global Learning Coefficient: 3:0
  - Personal Learning Coefficient: 2:0
- ABC
  - Colony Size: 100
  - Number of Onlooker Bees: 100
  - Abandonment Limit Parameter: 200
  - Acceleration Coefficient Upper Bound: 1
- BOA
  - Beam Size: 10
  - Selection Percentage: 40%
  - Replacement Percentage: 50%

1) THE DINING PHILOSOPHERS CASE STUDY EVALUATION

The present paper employs the Dinning philosopher problem as a reference case to evaluate our proposed enhanced ROA against other meta-heuristic algorithms. In order to evaluate the performance of each algorithm, we have summarized and compared the results in Tables 1 and 2. To verify each case study, we have calculated the maximum memory usage and the average time required to verify the whole state space. The findings from our analysis demonstrate that our proposed algorithm achieves a better results among the compared algorithms in the best-case scenario. This is primarily attributed to the state-space explosion issue, which occurs in NuSMV and other meta-heuristics algorithms when dealing with a larger number of philosophers. These algorithms need to generate and explore the entire state space, which poses a challenge to their efficiency. The results for the standard ROA is also acceptable as it is able to verify the state space in a reasonable time without facing the memory overflow. However, we tested a strategical migration method and the results improved. The reason is likely due to the fact that IROA's migration method helps to preserve diversity in the population, which reduces the risk of getting stuck in local optima. In contrast, ROA's migration method can sometimes lead to a loss of diversity in the population, which reduces the algorithm's ability to explore new regions of the search space. By improving the migration method, IROA is able to explore the search space more effectively, specially when the number os state space is high, leading to better results. The time and memory usage improve 1.09% and 1.03%, respectively.

2) PacMan GAME CASE STUDY EVALUATION

Our model, in particular, differs and begins with a field dimension of 4. The game features two agents: Pac-Man and Ghost.In every stage of the game, both entities have the ability to move to adjacent boxes while adhering to a predetermined set of guidelines. In the event that Pac-Man moves to a box

TABLE 1. Execution time of running dining philosophers.

	Time (min)						
Scale	Improved ROA	ROA	GA	PSO	ABC	BOA	NuSMV
40	14	14	722	31	146	130	32
100	46	49	1089	659	646	235	538
500	338	374	7040	NA	2081	NA	NA
1000	506	572	NA	NA	NA	NA	NA
1500	757	884	NA	NA	NA	NA	NA

TABLE 2. Memory consumption of dining philosophers.

	Memory (MB)						
Scale	Improved ROA	ROA	GA	PSO	ABC	BOA	NuSMV
40	16	16	38	76	49	44	90
100	42	43	79	138	926	83	506
500	77	80	266	NA	251	NA	NA
1000	208	219	NA	NA	NA	NA	NA
1500	559	606	NA	NA	NA	NA	NA

containing a marble, it will consume the marble. However, if Ghost moves to a box that is already occupied by Pac-Man, the latter will be killed. The game concludes when either Pac-Man devours all the marbles, resulting in a win, or when the Ghost kills Pac-Man, resulting in a loss. In both scenarios, a deadlock situation may arise. The state space of the problem under investigation is not particularly extensive due to the presence of multiple error states. This is a result of the significant number of potential error states that may arise, which constrains the size of the state space. Despite the relatively smaller state space, the problem presents unique challenges and complexities that require an effective and efficient solution. We deliberately selected the Pac-Man game as our case study to evaluate the performance of our proposed algorithm in comparison to other methods, as it presents a less complex state space compared to other benchmark problems, such as the dining philosopher's problem. This strategic selection allowed us to better isolate the key factors that influence the performance of our algorithm, while providing a meaningful comparison with existing methods. Through this approach, we were able to gain deeper insights into the strengths and limitations of our proposed approach and assess its potential for real-world applications.

Our experimental results, as shown in Tables 3 and 4, indicate that all the algorithms were able to successfully verify the case study. However, IROA demonstrated superior performance compared to the other methods. Despite the relatively small state space, all the methods were able to complete the verification process within the memory capacity constraints. Nevertheless, the incremental approach and verification of two different sets of candidate solutions (visible and reachable zones) in IROA allowed for faster verification with lower memory consumption. The improvement in time

TABLE 3. Execution time of running Pac-Man.

	Time (min)						
Scale	Improved ROA	ROA	GA	PSO	ABC	BOA	NuSMV
4×4	7.35	7.44	13.1	14.14	11.11	12.53	7.38
4×5	14.36	14.54	178.1	30.14	26.58	21.23	16.21
6×5	30.43	31.12	610.54	91.57	60.54	50.31	39.44

TABLE 4. Memory consumption of Pac-Man.

	Memory (MB)						
Scale	Improved ROA	ROA	GA	PSO	ABC	BOA	NuSMV
4×4	47	47	76	86	81	93	56
4×5	78	78	135	172	129	115	90
6×5	97	98	230	279	225	222	126

TABLE 5. Execution time of running N-Queen.

	Time (min)						
Scale	Improved ROA	ROA	GA	PSO	ABC	BOA	NuSMV
8×8	17.55	18.06	81.53	117	95.49	78.58	53
16×16	32.46	33.92	715	NA	125.47	662	NA
20×20	59.58	64.09	NA	NA	NA	NA	NA

and memory usage of IROA over ROA is marginal, with a difference of only 1.01% and 1.003%, respectively. This could be attributed to the fact that the proposed migration strategy shows significant improvement in scenarios where the state space is large and complex.

3) N-QUEEN CASE STUDY EVALUATION

The NxN refers to the challenge of positioning eight chess queens on an NxN chessboard. The rule is no two queens are allowed to fight each other. As a result, no two queens can be in the same column, row, or square in a solution. Any move in these circumstances will consider as a deadlock.

Our experimental results, presented in Tables 5 and 6, demonstrate that IROA is capable of detecting deadlocks in the N-Queen problem for large N. In contrast, the Pac-Man game, similar to the dining philosopher's problem, presents a more complex state space that posed significant challenges for NuSMV and other meta-heuristic algorithms. As a result, these methods were unable to complete the verification process due to the state space explosion (SSE) problem. By contrast, IROA's incremental approach and verification of visible and reachable zones enabled it to effectively tackle the complexities of the problem domain and provide a robust and efficient solution. The improvement in time and memory usage of IROA over ROA is marginal, with a difference of only 1.04% and 1.023%, respectively.

TABLE 6. Results of N-Queen.

	Memory (MB)						
Scale	Improved ROA	ROA	GA	PSO	ABC	BOA	NuSMV
8×8	69	70	275	291	48	500	
16×16	254	261	NA	503	290	NA	
20×20	381	400	NA	NA	NA	NA	

TABLE 7. Execution time of running Vehicle platoon.

	Time (min)						
Scale	Improved ROA	ROA	GA	PSO	ABC	BOA	NuSMV
6 Cars	0.10	0.10	0.59	0.57	0.29	0.51	0.46
12 Cars	0.36	0.36	1.04	NA	0.6	1.56	127.48
25 Cars	0.52	0.55	2.51	NA	5.51	3.43	NA
49 Cars	1.41	1.51	3.55	NA	NA	7	NA

TABLE 8. Results of vehicle platoon.

	Memory (MB)						
Scale	Improved ROA	ROA	GA	PSO	ABC	BOA	NuSMV
6 Cars	8	8	33	54	24	29	118
12 Cars	14	14	99	NA	210	82	480
25 Cars	32	33	236	NA	506	201	NA
49 Cars	90	93	500	NA	NA	508	NA

I intend to showcase your project to Dr. Sugu as the top-performing assignment.

4) VEHICLE PLATOON CASE STUDY EVALUATION

The vehicle platoon (VP) concept was developed to showcase the potential of vehicle automation technologies in alleviating traffic congestion. The VP system involves multiple vehicles traveling at a consistent speed and maintaining a fixed distance from each other on a highway, with one vehicle designated as the Leader and the rest as Followers. Our study focuses on a VP system comprising 25 vehicles, represented as processes, linked through a channel. Specifically, we aim to determine if the channel can be established between two followers, which requires identifying cases where a process submits a join request to a follower but receives no response, or when a follower sends a join request to another follower or recognizes a join request already sent by a leader. Our approach detects incorrect states and identifies deadlocks when either of the above scenarios occurs.

Table 7 and 8 showcase the results obtained from running all the meta-heuristic algorithms for the Vehicle platoon case study. It is observed that our proposed method, IROA, outperforms the other algorithms in terms of speed and absence of the SSE problem during verification. The speed of our proposed method can be attributed to the generation and checking of two different populations in parallel and the new migration method. Furthermore, GA and BOA demonstrate acceptable results when compared to the other approaches. The reason behind GA and BOA not facing the SSE problem in this case study can be attributed to the definition of the deadlock, which leads to a smaller state space. These results demonstrate the effectiveness of our proposed approach in verifying GTS and overcoming the limitations of the existing meta-heuristic algorithms. The improvement in time and memory usage of IROA over ROA is marginal, with a difference of only 1.03% and 1.015%, respectively.

By applying the efficiency formula presented in equation 4 to all case studies, the overall efficiency of IROA over ROA has been computed as 1.043 for execution time and 1.02 for memory usage.

VI. CONCLUSION

In this paper, we presented an incremental optimization framework based on a novel meta-heuristic algorithm for detecting deadlocks in large-scale software systems while overcoming the challenge of state space explosion. This challenge has persisted over time and continues to pose a major challenge for professionals in this field. Our approach incrementally generated and verified the state space to ensure the absence of deadlocks. We demonstrated the efficacy of our methodology through several complex case studies and found that it outperformed existing methods and tools like NuSMV. Our experiments showed that IROA is more efficient and effective than ROA for deadlock verification tasks, especially in scenarios with a large number of states where the state space explosion problem is likely to occur. The improved migration strategy in IROA helps maintain diversity in the population, allowing the algorithm to explore new regions of the search space and avoid getting stuck in local optima. The total efficiency of IROA over ROA was 1.043 for execution time and 1.02 for memory usage. Although our methodology is capable of completing the verification process and detecting deadlocks within a reasonable time without encountering the state space explosion problem, it is not able to guarantee complete accuracy of deadlock free-dome results. It is due to the stochastic nature of meta-heuristic algorithms that makes them that have the potential for producing incorrect results. Additionally, the proposed method's effectiveness is limited by the nature of the GTS verification problem itself, as this problem is known to be NP-hard in the worst case. However, when our algorithm does find an counter example, we can be confident that there is deadlock in the model. Moreover, our research has limitations in that IROA has primarily been tested on benchmark optimization problems, and its performance on real-world problems, like, is not yet well-established. For example, the correctness of GTS model of communication protocols, control systems, or the systems that provided in [50] and [51]. Future research is needed to assess the effectiveness of IROA on real-world optimization problems, especially those with high-dimensional and nonlinear search spaces. Addressing these two limitations could be part of our future work. Additionally, we plan to identify appropriate CEC benchmark tests to further evaluate and compare our IROA with the recent advancements in optimization algorithms.

REFERENCES

[1] E. Kerkouche, K. Khalfaoui, and A. Chaoui, "A rewriting logic-based semantics and analysis of uml activity diagrams: A graph transformation approach," *Int. J. Comput. Aided Eng. Technol.*, vol. 12, no. 2, pp. 237–262, 2020.

[2] R. Heckel and G. Taentzer, *Graph Transformation for Software Engineers*. Cham, Switzerland: Springer, 2020.

[3] M. Bachras and K. Kontogiannis, "Goal modelling meets service choreography: A graph transformation approach," in *Proc. IEEE 24th Int. Enterprise Distrib. Object Comput. Conf. (EDOC)*, Oct. 2020, pp. 30–39.

[4] E. Dinella, H. Dai, Z. Li, M. Naik, L. Song, and K. Wang, "Hoppity: Learning graph transformations to detect and fix bugs in programs," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020, pp. 1–17.

[5] R. Groner, L. Beaucamp, M. Tichy, and S. Becker, "An exploratory study on performance engineering in model transformations," in *Proc. 23rd ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst.*, Oct. 2020, pp. 308–319.

[6] M. Bozzano and A. Villafiorita, "Improving system reliability via model checking: The FSAP/NuSMV-SA safety analysis platform," in *Proc. Int. Conf. Comput. Saf., Rel., Secur.* Cham, Switzerland: Springer, 2003, pp. 49–62.

[7] S. Z. Koohi, N. A. W. A. Hamid, M. Othman, and G. Ibragimov, "Raccoon optimization algorithm," *IEEE Access*, vol. 7, pp. 5383–5399, 2019.

[8] A. Rensink, "The GROOVE simulator: A tool for state space generation," in *Proc. 2nd Int. Workshop Appl. Graph Transf. With Ind. Relevance (AGTIVE)*, Charlottesville, VA, USA. Springer, 2004, pp. 479–485.

[9] A. Rensink, "The GROOVE simulator: A tool for state space generation," in *Applications of Graph Transformations with Industrial Relevance*. Charlottesville, VA, USA: Springer, Sep. 2004, pp. 479–485.

[10] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *Proc. Int. Conf. Tools Algorithms For Construction Anal. Syst.* Cham, Switzerland: Springer, 1999, pp. 193–207.

[11] J. Dyck, "Verification of graph transformation systems with k-inductive invariants," Ph.D. dissertation, Hasso-Plattner-Inst., Dept. Digit. Eng., Potsdam, Germany, 2020.

[12] E. G. Amparore, S. Donatelli, and F. Galla, "A CTL* model checker for Petri nets," in *Proc. Int. Conf. Appl. Theory Petri Nets Concurrency*, 2020, pp. 403–413.

[13] F. Nejati, A. A. A. Ghani, N. K. Yap, and A. B. Jafaar, "Handling state space explosion in component-based software verification: A review," *IEEE Access*, vol. 9, pp. 77526–77544, 2021.

[14] B. König, D. Nolte, J. Padberg, and A. Rensink, "A tutorial on graph transformation," in *Graph Transformation, Specifications, and Nets*. Springer, 2018, pp. 83–104.

[15] U. G. Montanari, "Separable graphs, planar graphs and web grammars," *Inf. Control*, vol. 16, no. 3, pp. 243–267, May 1970.

[16] A. Rensink, I. Boneva, H. Kastenberg, and T. Staijen, "User manual for the GROOVE tool set," Dept. Comput. Sci., Univ. Twente, Enschede, The Netherlands, 2010. [Online]. Available: https://groove.ewi.utwente.nl/wpcontent/uploads/usermanual1.pdf

[17] N. A. Khan, M. Sulaiman, C. A. T. Romero, and F. S. Alshammari, "Analysis of nanofluid particles in a duct with thermal radiation by using an efficient metaheuristic-driven approach," *Nanomaterials*, vol. 12, no. 4, p. 637, Feb. 2022.

[18] N. A. Khan, M. Sulaiman, P. Kumam, and A. J. Aljohani, "A new soft computing approach for studying the wire coating dynamics with Oldroyd 8-constant fluid," *Phys. Fluids*, vol. 33, no. 3, Mar. 2021, Art. no. 036117.

[19] Y. Zhang, K. Chakrabarty, Z. Peng, A. Rezine, H. Li, P. Eles, and J. Jiang, "Software-based self-testing using bounded model checking for out-of-order superscalar processors," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 3, pp. 714–727, Mar. 2020.

[20] Y. Phyo, C. M. Do, and K. Ogata, "Toward development of a tool supporting a 2-layer divide & conquer approach to leads-to model checking," in *Proc. Int. Conf. Adv. Inf. Technol. (ICAIT)*, Nov. 2019, pp. 250–255.

[21] L. Wu, H. Huang, K. Su, S. Cai, and X. Zhang, "An I/O efficient model checking algorithm for large-scale systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 5, pp. 905–915, May 2015.

[22] R. Patel, K. Patel, and D. Patel, "On-the-fly symmetry reduction of explicitly represented probabilistic models," in *Proc. Int. Conf. Distrib. Comput. Internet Technol.* Cham, Switzerland: Springer, 2015, pp. 203–206.

[23] W. Zhu, H. Wu, and M. Deng, "LTL model checking based on binary classification of machine learning," *IEEE Access*, vol. 7, pp. 135703–135719, 2019.

[24] E. Pira, "A new heuristic for deadlock detection in safety analysis of software systems," Nashriah Muhandesi Barq va Muhandesi Kamputer, Tehran, Iran, 2022.

[25] T. Kumazawa, M. Takimoto, and Y. Kambayashi, "A safety checking algorithm with multi-swarm particle swarm optimization," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2022, pp. 786–789.

[26] K. Gaaloul, C. Menghi, S. Nejati, L. C. Briand, and Y. I. Parache, "Combining genetic programming and model checking to generate environment assumptions," *IEEE Trans. Softw. Eng.*, vol. 48, no. 9, pp. 3664–3685, Sep. 2022.

[27] N. Pourkhodabakhsh, M. M. Mamoudan, and A. Bozorgi-Amiri, "Effective machine learning, meta-heuristic algorithms and multi-criteria decision making to minimizing human resource turnover," *Applied Intelligence*, vol. 53, no. 12, pp. 16309–16331, 2022.

[28] S. Lazreg, M. Cordy, and A. Legay, "Verification of variability-intensive stochastic systems with statistical model checking," in *Proc. Int. Symp. Leveraging Appl. Formal Methods*. Greece: Springer, Oct. 2022, pp. 448–471.

[29] A. Valmari, "A stubborn attack on state explosion," in *Proc. Int. Conf. Comput. Aided Verification*. Cham, Switzerland: Springer, pp. 156–165, 1990.

[30] U. Stern and D. L. Dill, "Improved probabilistic verification by hash compaction," in *Proc. Adv. Res. Work. Conf. Correct Hardw. Design Verification Methods*. Cham, Switzerland: Springer, 1995, pp. 206–224.

[31] L. Bortolussi, L. Cardelli, M. Kwiatkowska, and L. Laurenti, "Central limit model checking," *ACM Trans. Comput. Log.*, vol. 20, no. 4, pp. 1–35, Oct. 2019.

[32] S. Donatelli, "Markov regenerative processes solution and stochastic model checking: An on-the-fly approach," in *Proc. 12th EAI Int. Conf. Perform. Eval. Methodologies Tools*, Mar. 2019, pp. 1–5.

[33] T. Zheng and Y. Liu, "Genetic algorithm for generating counterexample in stochastic model checking," in *Proc. VII Int. Conf. Netw., Commun. Comput.*, Dec. 2018, pp. 92–96.

[34] K. Gaaloul, C. Menghi, S. Nejati, L. C. Briand, and Y. I. Parache, "Combining genetic programming and model checking to generate environment assumptions," 2021, *arXiv:2101.01933*.

[35] Y. Ma, Z. Cao, and Y. Liu, "A probabilistic assume-guarantee reasoning framework based on genetic algorithm," *IEEE Access*, vol. 7, pp. 83839–83851, 2019.

[36] L. M. Duarte, L. Foss, F. R. Wagner, and T. Heimfarth, "Model checking the ant colony optimisation," in *Distributed, Parallel and Biologically Inspired Systems*. Berlin, Germany: Springer, pp. 221–232, 2010.

[37] T. Kumazawa, M. Takimoto, and Y. Kambayashi, "Exploration strategies for balancing efficiency and comprehensibility in model checking with ant colony optimization," *J. Inf. Telecommun.*, vol. 6, no. 3, pp. 341–359, Jul. 2022.

[38] R. Yousefian, S. Aboutorabi, and V. Rafe, "A greedy algorithm versus metaheuristic solutions to deadlock detection in graph transformation systems," *J. Intell. Fuzzy Syst.*, vol. 31, no. 1, pp. 137–149, Jun. 2016.

[39] N. Rezaee and H. Momeni, "A hybrid meta-heuristic approach to cope with state space explosion in model checking technique for deadlock freeness," *J. AI Data Mining*, vol. 8, no. 2, pp. 189–199, 2020.

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

[41] E. Pira, V. Rafe, and A. Nikanjam, "Deadlock detection in complex software systems specified through graph transformation using Bayesian optimization algorithm," *J. Syst. Softw.*, vol. 131, pp. 181–200, Sep. 2017.

[42] V. Rafe, S. Mohammady, and E. Cuevas, "Using Bayesian optimization algorithm for model-based integration testing," *Soft Comput.*, vol. 26, no. 7, pp. 3503–3525, Apr. 2022.

[43] A. Schmidt, "Model checking of visual modeling languages," in *Proc. Conf. PhD Students Comput. Sci.*, 2004, p. 102.

[44] R. Heckel, "Graph transformation in a nutshell," *Electron. Notes Theor. Comput. Sci.*, vol. 148, no. 1, pp. 187–198, Feb. 2006.

[45] D. Swaroop, "String stability of interconnected systems: An application to platooning in automated highway systems," Ph.D. dissertation, California Partners Adv. Transit Highways (PATH), Richmond, CA, USA, 1994.

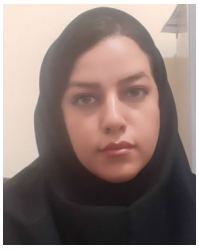
[46] Y.-G. Gao and D.-Y. Song, "A new improved genetic algorithms and its property analysis," in *Proc. 3rd Int. Conf. Genetic Evol. Comput.*, Oct. 2009, pp. 73–76.

[47] A. R. Jordehi, "Enhanced leader PSO (ELPSO): A new PSO variant for solving global optimisation problems," *Appl. Soft Comput.*, vol. 26, pp. 401–417, Jan. 2015.
[48] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm," *J. Global Optim.*, vol. 39, no. 3, pp. 459–471, Oct. 2007.
[49] M. Pelikan, D. E. Goldberg, and S. Tsutsui, "Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms," in *Proc. SICE Annu. Conf.*, vol. 3, 2003, pp. 2738–2743.
[50] N. Ahmad Khan and M. Sulaiman, "Heat transfer and thermal conductivity of magneto micropolar fluid with thermal non-equilibrium condition passing through the vertical porous medium," *Waves Random Complex Media*, vol. 32, pp. 1–25, Aug. 2022.
[51] N. A. Khan, M. Sulaiman, and F. S. Alshammari, "Heat transfer analysis of an inclined longitudinal porous fin of trapezoidal, rectangular and dovetail profiles using cascade neural networks," *Structural Multidisciplinary Optim.*, vol. 65, no. 9, p. 251, Sep. 2022.

FARANAK NEJATI received the M.Sc. degree in computer science from Tabriz University, Tabriz, Iran, and the Ph.D. degree in computer science (software engineering) from the University of Putra Malaysia, Serdang, Malaysia. With a remarkable industry experience of nearly eight years, she has been actively involved in providing software solutions for critical systems, particularly in the domain of remote control systems. She currently holds the position of University Lecturer and continues to contribute to academia. Her research interests include various cutting-edge areas, including artificial intelligence, machine learning, optimization algorithms, mathematical software specification and verification, and component-based software development. Throughout her career, she has demonstrated a strong commitment to interdisciplinary collaboration, actively engaging with researchers from diverse disciplines within the field of computer science.

NOR ASILAH WATI ABDUL HAMID (Senior Member, IEEE) received the Ph.D. degree from The University of Adelaide, in 2008. She has been a Visiting Scholar with the High Performance Computing Laboratory, George Washington University, USA, for two years. She is currently an Associate Professor with the Department of Communication Technology and Network, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Malaysia. She is also an Associate Professor with the Distributed and High Performance Computing (DHPC) Group, working on high-performance distributed and parallel computing technologies and applications. She is also an Associate Researcher and the Coordinator of high-speed machines with the Institute for Mathematical Research (INSPEM), Universiti Putra Malaysia. Her research interests include parallel and distributed computing, cluster computing, distributed information systems, and other applications of high-performance computing.

SINA ZANGBARI KOOHI received the B.Sc. degree in computer science from Mazandaran University, the M.Sc. degree in computer science from Tabriz University, and the Ph.D. degree in parallel and distributed computing from the esteemed University of Putra Malaysia. He is a highly accomplished individual. With a career spanning over 15 years, he has demonstrated exceptional expertise in the field of computer science. His research interests include a diverse range of cutting-edge areas, including distributed systems, parallel computing, graph theory, and artificial intelligence. Notably, his contributions extend beyond academia, as he has gained hands-on experience in the field of robotics, excelling in national robotic tournaments, and earning several prestigious accolades.



ZAHRA RAHMANI ZADEH is a dedicated professional who holds a Bachelor of Science (B.Sc.) degree in software engineering from Arak University of Applied Sciences and Technology (UAST), Arak, Iran. She further pursued her academic aspirations and successfully completed her Master of Science (M.Sc.) degree in software engineering at Shahab Danesh University, Qom, Iran. Following her graduation, she joined the Technical and Vocational High School in Qom, Iran, where she assumed the role of a Lecturer. She also work as a part-time researcher in collaboration with Department of Software Engineering, UAST. In recent years, her endeavours have primarily centred around the development and application of artificial intelligent algorithms. She has actively collaborated with researchers from various disciplines within computer science, showcasing her commitment to interdisciplinary research and collaboration. Her research interests encompass a diverse range of topics within the field of computer science. She has a keen focus on artificial intelligence, machine learning, optimisation algorithms, software specification, and verification.