Mathematical and Computational Applications

MDPI

Article

A Hybrid Approach for Reachability Analysis of Complex Software Systems Using Fuzzy Adaptive Particle Swarm Optimization Algorithm and Rule Composition

Nahid Salimi [1], Seyfollah Soleimani [1,*], Vahid Rafe [2] and Davood Khodadad [3,*]

[1] Department of Computer Engineering, Faculty of Engineering, Arak University, Arak 38156-8-8349, Iran; n.salimi.n@gmail.com
[2] Department of Computer Science, City St George's, University of London, London EC1V 0HB, UK; vahid.rafe@city.ac.uk
[3] Department of Applied Physics and Electronics, Umeå University, 901 87 Umeå, Sweden
* Correspondence: s-soleimani@araku.ac.ir (S.S.); davood.khodadad@umu.se (D.K.)

Abstract: Model checking has become a widely used and precise technique for verifying software systems. However, a major challenge in model checking is state space explosion, which occurs due to the exponential memory usage required by the model checker. To address this issue, meta-heuristic and evolutionary algorithms offer a promising solution by searching for a state where a property is either satisfied or violated. Recently, various evolutionary algorithms, such as Genetic Algorithms and Particle Swarm Optimization, have been applied to detect deadlock states. While these approaches have been useful, they primarily focus on deadlock detection. This paper proposes a fuzzy algorithm to analyse reachability properties in systems specified through Graph Transformation Systems with large state spaces. To achieve this, the existing Particle Swarm Optimisation algorithm, which is typically used for deadlock detection, has been extended to analyse reachability properties. To further enhance accuracy, a Fuzzy Adaptive Particle Swarm Optimization algorithm is introduced to determine which states and paths should be explored at each step-in order to find the corresponding reachable state. Additionally, the proposed hybrid algorithm was applied to models generated through rule composition to assess the impact of rule composition on execution time and the number of explored states. These approaches were implemented within an open-source toolset called GROOVE, which is used for designing and model checking Graph Transformation Systems. Experimental results demonstrate that proposed hybrid algorithm reduced verification time by up to 49.86% compared to Particle Swarm Optimization and 65.17% compared to Genetic Algorithms in reachability analysis of complex models. Furthermore, it explored 32.7% fewer states on average than the hybrid method based on Particle Swarm Optimization and Gravitational Search Algorithms, and 57.4% fewer states compared to Genetic Algorithms, indicating improved search efficiency. The application of rule composition further reduced execution time by 35.7% and the number of explored states by 41.2% in large-scale models. These results confirm that proposed hybrid algorithm significantly enhances reachability analysis in the systems modelled via Graph Transformation, improving both computational efficiency and scalability.

Keywords: fuzzy adaptive particle swarm optimization; graph transformation system; model checking; reachability property; rule composition

Academic Editor: Leonardo Trujillo

Received: 25 April 2025
Revised: 31 May 2025
Accepted: 3 June 2025
Published: 10 June 2025

Citation: Salimi, N.; Soleimani, S.; Rafe, V.; Khodadad, D. A Hybrid Approach for Reachability Analysis of Complex Software Systems Using Fuzzy Adaptive Particle Swarm Optimization Algorithm and Rule Composition. *Math. Comput. Appl.* **2025**, *30*, 65. https://doi.org/10.3390/mca30030065

1. Introduction

In recent decades, software systems have played a crucial role in software development and security considerations. As software development techniques have evolved, they have enabled the creation of increasingly complex software systems. Model-Driven Engineering (MDE) leverages models to describe complex systems at multiple levels of abstraction [1]. Modelling systems allows for the application of model checking, a formal analysis technique used to verify software and detect system errors during the design phase—an approach that is both easier and more cost-effective than identifying issues after implementation.

Using a modelling language is essential for representing systems and verifying them through model checking. Graph Transformation Systems (GTS) is a visual, graph-based formal language used to model software systems with dynamic structures [2]. GTS employs graphs to represent and analyse the structure of complex systems [3]. However, when applied to large systems, GTS models can become enormous, leading to the well-known state space explosion problem, which poses a significant challenge to model checking [4]. Various classical methods, such as symbolic verification [5], partial order reduction [6], and symmetry model checking [7], have been introduced to reduce time and memory consumption during the model checking process.

Several heuristic approaches have also been proposed to address the state space explosion problem. These include Depth-First Search [8], Best-First Search [9], an adaptation of the A* algorithm [10], Coverage-First Search [11], and an A* search algorithm for verifying liveness properties in explicit-state model checking [6]. However, even with these methods, exhaustive searches within the state space can still lead to memory limitations and reduced computational speed.

In recent years, metaheuristic and evolutionary algorithms have gained attention for their efficiency in addressing these challenges compared to classical and heuristic approaches. These methods explore only a subset of the system's state space rather than the entire space, making them more practical for mitigating state space explosion. Several metaheuristic approaches have been proposed to detect deadlocks and refute security properties. For instance, the Ant Colony Optimization (ACO) algorithm has been applied in this context [12,13]. Additionally, a novel approach utilizing the Genetic Algorithm (GA) has been introduced to verify the correctness of communication protocols [14]. This genetic validation technique has been tested on both a manually created protocol and the Transmission Control Protocol (TCP).

In [15], two distinct learning algorithms were proposed to verify safety, reachability, and liveness properties of systems whose state spaces can be expressed through regular expressions. Another study presented an ACO-based approach to mitigate the state space explosion problem when detecting deadlocks in complex networks modelled using the Calculus of Communicating Systems (CCS) [12]. Furthermore, two other studies [16,17] introduced an ACO-based model to refute safety and liveness properties in concurrent systems. This approach applies the GA algorithm to generate several random paths of a specific length starting from the initial state and identifies the first path leading to a deadlock state. To assess its efficiency, this solution was implemented using the GROOVE toolset.

Another proposed approach for addressing this problem utilizes the Particle Swarm Optimization (PSO) algorithm to detect deadlocks in Graph Transformation Systems [18]. Additionally, researchers have developed a hybrid algorithm that combines PSO with the Gravitational Search Algorithm (GSA) to overcome the issue of local optima (PSO-GSA). In this paper, GSA was integrated to enhance PSO's performance, and the results demonstrated that combining PSO with GSA, along with other metaheuristic algorithms, could significantly improve model checking analysis for problems defined through GTS.

In [19], an efficient approach utilizing data mining techniques, called EMCDM, is proposed to analyse complex software system models. These systems are designed according to a specific architectural style and formally modelled using GTS. The EMCDM approach is applied to verify the reachability property as a means of refuting the safety property.

In another study [20], a method based on the Bayesian Optimization Algorithm (BOA) is introduced to detect deadlocks in systems specified through graph transformations. The results indicate significant improvements in terms of both speed and accuracy. One of the key properties verified in the model checking process is reachability, which can serve as an alternative to refuting a similar safety property. The authors of [21] applied machine learning techniques, such as the ensemble classification technique, for reachability verification and model checking of systems modelled through GTS. This approach starts with generating a small model and search the explored state space to find the paths forwarded to goal states. Then, the ensemble classification technique was applied to extract necessary knowledge paths and intelligently explore the state space of the bigger model.

An incremental optimization framework for model checking GTS was proposed in [22], focusing on deadlock detection using a novel metaheuristic algorithm called the Raccoon Optimization Algorithm. Additionally, a new approach for solving AI planning problems in systems specified via GTS, based on the Bayesian Optimization Algorithm, was introduced in [23]. The authors of [24] proposed a deep reinforcement learning-based algorithm utilizing a Double Deep Q-Network (DDQN) to search for reachability properties in systems specified through graph transformation.

Reachability has become an essential aspect of verification, as other properties, such as safety, can be analysed through reachability verification of states that refute safety properties. Since safety and reachability are dual concepts, verifying the reachability property "not p" serves as an alternative to verifying the safety property "p" [24].

Unlike prior metaheuristic approaches that focus narrowly on deadlock detection with static parameters and generic fitness metrics, this study introduces a fuzzy-adaptive PSO framework with a GTS-aware fitness function—filling the gap in accurate and scalable reachability verification for complex transformation-based models.

Rule composition is also a fundamental concept in graph transformations. For example, refs. [25,26] present two different types of composition: sequential and parallel (spatial). Sequential composition is achieved through concatenation. Given two transformations, p: G1 $\rightarrow$ G2 and p′: G2 $\rightarrow$ G3, the composed rule p″: G1 $\rightarrow$ G3 is defined as their sequential composition [26]. Parallel composition is realised through disjoint union or amalgamation: In the simplest case, for given transformation p: L $\leftarrow$ I $\rightarrow$ R and p′: L′ $\leftarrow$ I′ $\rightarrow$ R′, the transformation p″: p + p′ = (L + L′ $\leftarrow$ I + I′ $\rightarrow$ R + R′). Here, "+" denotes the binary coproduct [27].

The increasing complexity of modern software systems has intensified the challenge of verifying behavioural properties such as reachability, especially in systems modelled using GTS. Reachability analysis—the task of determining whether a system can evolve from an initial state to a defined goal state—is crucial for ensuring system correctness. However, in the context of GTS, this process is hindered by the state space explosion problem, where the number of possible system configurations grows exponentially with the number of transformation rules and components.

Traditional verification methods, including symbolic model checking and SAT-based approaches, are often insufficient in such contexts due to their requirement for exhaustive exploration, leading to scalability limitations. Likewise, although metaheuristic algorithms such as GA, ACO, and basic PSO have been employed to reduce computational load, these techniques frequently focus on deadlock detection and do not generalize well to broader reachability verification tasks. More importantly, they struggle with parameter tuning and

often lack mechanisms for adapting to the dynamic nature of large state spaces, resulting in suboptimal performance or premature convergence.

In this work, we address the specific need for an adaptive, efficient, and scalable approach to reachability analysis in GTS. Our proposed Fuzzy Adaptive PSO (FAPSO) method meets this need by integrating a fuzzy inference system to dynamically adjust PSO parameters based on system diversity and search progression. This adaptability ensures that the algorithm maintains an effective balance between exploration and exploitation, even in large and irregular search spaces. Additionally, we introduce a domain-specific fitness function that evaluates path validity based on graph structural similarity and penalizes violations of Negative Application Conditions (NACs), ensuring that the search remains both targeted and semantically sound.

The suitability of FAPSO for this domain stems from its ability to navigate complex, high-dimensional search spaces without requiring exhaustive model exploration. Unlike fixed-parameter or hybrid methods that assume static conditions, FAPSO continuously adjusts its behaviour based on the evolving characteristics of the search space. This makes it highly effective for scenarios where the state space is not only large but also dynamically structured—a common trait in real-world GTS applications.

Therefore, FAPSO is not merely an alternative to existing algorithms; it is a necessary and well-aligned solution for the challenges posed by reachability analysis in GTS. Its adaptive behaviour, guided by system-specific feedback, ensures efficiency and accuracy in settings where traditional methods falter.

This paper is organized as follows: the proposed approaches based on PSO and Fuzzy PSO algorithms have been presented 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 obtaining the experimental results based on several well-known case studies. Moreover, a discussion of the observations is presented. The superiority of the proposed approaches has been discussed in Section 5. Finally, Section 6 concludes the paper and highlights the future works.

2. Background

2.1. Model Checking

Model checking is a fully automated technique used to verify the correctness properties of various systems. This method assesses whether a specific correctness property applies to a given system by exploring the potential transitions between different states. Some of the properties that can be verified during model checking include safety, reachability, liveness, and fairness. A safety property ensures that "no undesired situation should occur" or that a "desired event" must always happen within the system. This property is considered satisfied when all finite and infinite paths in the model meet this condition. If a finite path leads to a goal state that violates the safety property, the property is deemed violated. Regarding the reachability property, model checking determines whether a specific configuration will be reached within the system. If a finite path reaches a goal state that satisfies this property, the system is verified to meet the reachability condition. The two properties—safety and reachability—are dual in nature. In fact, verifying the reachability property as the negation of the safety property can serve as a counterexample that violates the safety property [28].

2.2. Graph Transformation System

Using model checking to verify a system requires that the system be described using a formal language. To model systems with dynamic structures, graph transformation can be employed as a graph-based visual formal language [29]. One of the fundamental features

of a graph transformation system is its formal and precise mathematical foundation [3]. An attributed GTS is represented as a triple: AGT = (TG, HG, R), where TG is the Type graph, HG is the host graph, and R is the ruleset. The Type graph (TG) defines the system's overview and meta-model, while the host graph (HG) represents the initial configuration of the system as an instance of the Type graph. Different configurations of the system can be generated by applying transformation rules to the host graph. A graph transformation rule set, R, on a Type graph (TG) can be defined by a triple (LHS, RHS, NAC). The left-hand side (LHS) and right-hand side (RHS) specify the rule's preconditions and postconditions, respectively. The NAC (Negative Application Condition) is a specific configuration used to ensure that no subgraph exists in the rule.

2.3. Particle Swarm Optimization Algorithm

The PSO algorithm is one of the most widely used optimization techniques for solving multi-dimensional problems. Initially, the algorithm randomly generates an initial population of candidate solutions, represented as particles. The position of the i^{th} particle is described by the vector *xi*. In each iteration, the algorithm calculates the fitness of each particle, which measures how optimal the particle is as a potential solution. This process continues until a termination criterion is met. During the iterations, the "personal best" (p-best) and "global best" (g-best) values are updated for all particles. The velocity (*vi*) and position (*xi*) of the particles are then updated, as shown in Equations (1) and (2):

$$\nu_i(t+1) = W * \nu_i(t) + C1 * (p\text{-}best_i - x_i(t)) * R1 + C2 * (g\text{-}best - x_i(t)) * R2 \quad (1)$$

$$x_i(t+1) = x_i(t) + \nu_i(t+1) \quad (2)$$

C1 and C2 represent the "cognitive coefficient" and "social coefficient", respectively, which specify the influence of personal experience and collective experience on particle behaviour. These coefficients are real-valued and typically fall within the range $0 \leq C1 + C2 \leq 4$. The momentum weight, denoted by W, determines the extent to which a particle's velocity at the current step influences its velocity at the next step. Additionally, R1 and R2 are two random factors, each containing a diagonal matrix of random real numbers within the range (0, 1) [30]. The iterative process of calculating fitness and updating the global best (g-best), personal best (p-best), position, and velocity of all particles continues until a termination condition is met. The pseudo-code related to the algorithm can be seen in Algorithm 1.

Algorithm 1. Pseudo Code for PSO Algorithm

1. Initialization
a. **Initialize** the particle's position $x_i(t)$ for ($t = 0$, *i:1...N*)
b. **Initialize** the particle's best position to its initial position $p_i(t) = x_i(t)$ for ($t = 0$, *i:1...N*)
c. **Calculate** the fitness of each particle $f(x_i(t))$ for ($t = 0$, *i:1...N*)
d. **if** $f(x_i(0)) \leq f(x_j(0))$ **then** Initialize the global best as $g\text{-}best = x_j(t)$ for each *i,j:1...N*, $i \neq j$)
2. while a stopping criterion is not met **repeat** the following steps:
Update the velocity v_i for each particle: $v_i(t+1) = W^* v_i(t) + C_1(p\text{-}best_i - x_i(t))\, R_1 + C_2(g\text{-}best - x_i(t))\, R_2$
Update the position x_i for each particle: $x_i(t+1) = x_i(t) + v_i(t+1)$
Evaluate the fitness $f(x_i(t+1))$ for each particle.
if $f(x_i(t+1)) \geq f(p\text{-}best_i)$ **then** **Update** personal best: $p\text{-}best_i = x_i(t+1)$
if $f(x_i(t+1)) \geq f(g\text{-}best)$ **then** **Update** personal best: $g\text{-}best = x_i(t+1)$
At the end of iterative process, the best solution is represented by *g-best*.

2.4. Fuzzy Inference System

Fuzzy logic is a mathematically based framework for representing human knowledge and experience. A Fuzzy Logic Controller (FLC) consists of a Knowledge Base that encodes expert knowledge through a series of IF-THEN rules. An IF-THEN rule is a conditional statement structured as follows: If a specific set of conditions is met, then a corresponding set of consequences can be inferred.

A fuzzy system comprises three main components: fuzzification, the Fuzzy Inference System (FIS), and defuzzification:

- In the fuzzification process, the linguistic variables (for both inputs and outputs) are defined.
- The FIS defines the rules that describe how the system works and maps inputs to outputs.
- In the defuzzification process, the outputs are calculated.

To design a fuzzy model, the following elements must be defined:

1. The input and output variables.
2. The fuzzy membership functions.
3. The fuzzy rules.
4. The parameters used in defining the membership functions and rules.

3. The Proposed Approaches

In this paper, two approaches are proposed to manage the state space explosion in systems formally defined by GTS for verifying the reachability property: one based on PSO and the other on a fuzzy adaptive PSO. Although metaheuristic algorithms, such as PSO, are widely used to address optimization problems, the proposed approach leverages these algorithms to search for the reachability property within a potentially vast state space, which may even be infinite.

This study aims to overcome key limitations in existing metaheuristic model checking methods by introducing a fuzzy-adaptive swarm optimization framework tailored for GTS-based systems. It addresses both the algorithmic rigidity of traditional PSO and the structural blind spots of generic fitness functions.

The integration of a fuzzy system into the PSO algorithm is justified by the need for context-sensitive control of swarm behaviour in highly dynamic state spaces. Rule composition supports this by simplifying the transformation rules, reducing the branching factor, and accelerating convergence.

3.1. Problem Formulation

Most prior metaheuristic-based methods—such as GA, ACO, and basic PSO—primarily address deadlock detection or safety refutation. These approaches do not generalize well to the broader reachability analysis tasks needed for verifying dynamic systems specified by GTS.

Previous algorithms often suffer from premature convergence due to fixed parameter settings (e.g., C1 and C2 in PSO), especially in complex, high-dimensional state spaces typical of GTS.

Conventional methods apply general heuristics (like path length minimization), which are agnostic to the semantics and structure of graph-based models.

To overcome these gaps, we propose a novel Fuzzy Adaptive Particle Swarm Optimization (FAPSO) algorithm that introduces:

- A fuzzy inference system for dynamic adjustment of PSO parameters based on iteration progress and particle diversity, improving exploration and convergence.

- A domain-specific fitness function that evaluates graph similarity and penalizes violations of Negative Application Conditions (NACs), allowing more precise navigation of GTS-based state spaces.
- Integration with rule composition to reduce model complexity and the number of explored states.

These innovations advance the state of the art by enabling more accurate, efficient, and scalable reachability analysis, particularly in systems modelled using graph transformations, where exhaustive approaches are computationally infeasible.

In the context of verifying software systems modelled using Graph Transformation Systems (GTS), reachability analysis involves determining whether a system can evolve from an initial configuration to a target state that satisfies a specific property. This task becomes computationally intractable as system complexity increases, due to the exponential growth of the state space, a phenomenon known as state space explosion. The core problem addressed in this study is:

Given a system SSS modelled by a GTS triple (TG, HG, R), and a reachability property graph $G_p = (V_p, E_p)$ along with associated Negative Application Conditions (NACs), find a sequence of transformation rules $r_1, r_2, \ldots, r_n \in R$ that, when applied to the initial host graph HG, results in a state G_h such that:

1. Gh maximally matches the structural and semantic features of Gp;
2. No NAC conditions are violated in Gh;
3. The number of explored intermediate states and computational time are minimized.

This problem formulation encompasses both correctness (via semantic matching and NAC compliance) and efficiency (via minimization of state exploration and computation time). The solution space consists of all possible paths through the state transition graph generated by applying rules in RRR to the host graph HG. However, exhaustive traversal of this space is impractical for large systems.

To overcome these challenges, the search for valid transformation sequences is reframed as an optimization problem, where each candidate solution (i.e., a sequence of rule applications) is evaluated based on its fitness in reaching a goal state resembling G_p and avoiding NAC violations. Traditional Particle Swarm Optimization (PSO) methods, while useful, suffer from premature convergence and static parameter settings that limit adaptability to the dynamic structure of GTS-based state spaces.

Therefore, this paper proposes an enhanced approach, Fuzzy Adaptive PSO (FAPSO), to address the reachability verification problem more effectively by dynamically tuning search parameters based on search progress and diversity, thereby maintaining a balanced and targeted exploration of the solution space.

Let a software system be specified using a Graph Transformation System (GTS), defined as the triple G = (TG, HG, R), where:

- TG is the Type graph, specifying the meta-model of the system;
- HG is the host graph, representing the initial configuration of the system;
- R = {r1, r2, . . ., rk} is the rule set, where each rule ri = (LHSi, RHSi, NACi) defines a transformation on graphs.

The reachability verification problem is defined as follows:

Given a target graph $G_p = (V_p, E_p)$, representing the state that satisfies a desired reachability property, determine whether there exists a sequence of transformation rules $\pi = [r_{i1}, r_{i2}, \ldots, r_{in}] \in R_n$ such that:

1. Applying $\pi\backslash pi\pi$ to HG yields a graph Gh = π(HG);
2. Gh maximally matches Gp in structure and labels;
3. Gh does not violate any NACs defined for Gp;

4. The path length $|\pi|$ and the number of explored states is minimized.

Mathematically, this can be viewed as an optimization problem:

Maximize Similarity (G_p, G_h) − Penalty$_{NAC}$ (G_h, NAC_p)
Subject to $G_h = \pi(HG)$, $\pi \in R^n$ G_h satisfies all structural constraints of TG

where:

- Similarity (Gp, Gh) is a domain-specific function measuring graph isomorphism or structural resemblance (e.g., node/edge label matches);
- PenaltyNAC penalizes configurations that violate the Negative Application Conditions associated with Gp.

Concrete Example: Dining Philosophers Problem

Let us consider a simplified instance of the dining philosophers problem with five philosophers and five forks. Each philosopher may transition through the following states:

- Thinking, Hungry, HasLeft, Eating, HasRight.

Let the transformation rules be:

- r1: GoHungry;
- r2: GetLeft;
- r3: GetRight;
- r4: ReleaseLeft;
- r5: ReleaseRight.

The reachability property we wish to verify is a deadlock state, where all philosophers are in the HasLeft state, each waiting for their right fork. This state is represented by a target graph G_p with labelled nodes indicating each philosopher in HasLeft and corresponding fork usage constraints.

Thus, the goal is to find a rule sequence $\pi \in R_n$ that leads the system from the initial Thinking state to a configuration where all philosophers satisfy the HasLeft state condition, while ensuring that:

- No forks are shared (NAC);
- All graph transformations are valid under the GTS semantics.

In this specific case:

- The similarity function Similarity (Gp, Gh) counts matched philosopher states labelled HasLeft and correctly allocated forks.
- The penalty function PenaltyNAC ensures no philosopher has picked both forks, which would violate the deadlock property.

3.2. Particle Encoding

A reachability property p can be verified by finding a state where p occurs, through exploring the system's reachable state space. In the proposed algorithms, the output is the path that starts from an initial state and ends at the state that satisfies the reachability property. As mentioned, particles represent candidate solutions, and their positions are defined by a sequence of numbers that represent a path. Each number corresponds to a transition at each stage, with a minimum value of 0 and a maximum value equal to the highest number of possible outgoing transitions in the state space. For example, the path '1 0 2 1' in Figure 1 represents the position of a particle. This position can be encoded as the path "r_1 r_0 r_2 r_1" where r_i denotes the applied rule.

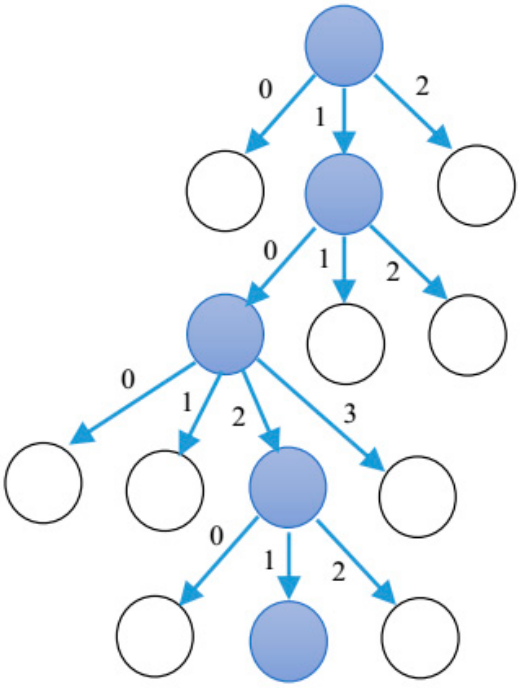

Figure 1. A solution encoded by the path <1,0,2,1>.

3.3. Fitness Function

Fitness is a measure of how suitable a particle is as a potential solution to the goal. In this approach, each particle represents a path of specific length, starting from the initial state and ending at another state in the state space. The goal is to find a state that matches the defined reachability property. It can be assumed that the more similar the final state of the path is to the reachability property, the more likely the path is to be a promising candidate. Therefore, the similarity between the path's final state and the specified reachability property is used to define the fitness function.

In this paper, the system is modelled using GTS, and the associated states and properties are represented by graphs using the GROOVE toolset. As mentioned earlier, a GTS is represented as a triple (TG, HG, R), where TG, HG, and R correspond to the Type graph, host graph, and graph transformation rule set, respectively. Additionally, R can be specified by the triple (LHS, RHS, NAC), where LHS and RHS represent the left- and right-hand sides, describing the pre- and post-conditions of the rules, respectively. NAC stands for Negative Application Condition, which defines a configuration that must not exist for the rule to be applied.

In the GROOVE toolset, LHS, RHS, and NAC are represented as individual graphs, with colours used to distinguish the original LHS, RHS, and NAC graphs. If the blue dashed edges and nodes appear in the LHS, the rule can be applied to the host graph, and they can be removed after the rule is applied. The bold green solid edges and nodes belong to the RHS, which should be created after rule application [31]. In the graph, NACs are represented by red, bold, dashed edges and nodes, indicating configurations that must not be present for the rule to apply.

Each node and edge in the graph can have labels, which can be defined by self-loop edges named after the node's label. The fitness function takes two inputs: a particle and the reachability property under study. The fitness value is calculated as follows:

1. Identify pairs of nodes where the first node belongs to the given property graph (excluding NAC nodes) and the second node belongs to the last state of the path specified by the particle, ensuring that they have the same labels.

2. Count the total number of pairs found in the first step.
3. Count the total number of each NAC node and edge for the given properties occurring in the graph that represents the path's last state, encoded by the given particle.
4. The fitness value is the difference between the total calculated in step 2 and the total from step 3.

Let $G_p = (V_p, E_p)$ be the goal property graph (the reachability target), and $G_h = (V_h, E_h)$ be the final state graph reached by a particle.

Let $NAC_p = (V_{nac}, E_{nac})$ denote the NAC subgraphs associated with the property.

1. Node and Edge Matching Score:

Let $match_n(v) = 1$ if node v in V_p has a matching node in V_h (based on label), otherwise 0.

Let $match_e(e) = 1$ if edge e in E_p has a matching edge in E_h with the same source, target, and label.

Then, the matching score is defined as:

$M(G_p, G_h) = \sum_{v \in V_p} match_n(v) + \sum_{e \in E_p} match_e(e)$

2. NAC Penalty:

Let $viol_n(v) = 1$ if node v in V_{nac} is found in V_h, otherwise 0.

Let $viol_e(e) = 1$ if edge e in E_{nac} is found in E_h, otherwise 0.

The penalty function is defined as:

$P(G_h, NAC_p) = \sum_{v \in V_{nac}} viol_n(v) + \sum_{e \in E_{nac}} viol_e(e)$

3. Final Fitness Function:

The fitness value F for a particle is computed as:

$F(G_h, G_p, NAC_p) = M(G_p, G_h) - P(G_h, NAC_p)$

This function rewards paths that end in goal-like states and penalizes paths that contain NAC violations.

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

Algorithm 2. Fitness Function for PSO and FAPSO Approaches

1. Input and 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
$E\text{-}Count_{ij}$ = 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;
$PE\text{-}Count_{ij}$ = size of ENP
$DE\text{-}Count_{ij} = E\text{-}Count_{ij} - PE\text{-}Count_{ij}$
end for
end for

Algorithm 2. *Cont.*

4. EQ-Count = 0
   while all $Visited_{ij}$ is not *true* **do**
      Find the smallest $DE\text{-}Count_{ij}$ **that** $Visited_{ij}$ = *false*
   $Visited_{ij}$ = *true*
   if $pVisited_{ij}$ is not *true* **&&** $hVisited_{ij}$ is not *true* **then**
      EQ-Count += $E\text{-}Count_{ij}$;
      $pVisited_{ij}$ = *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 edges and nodes of NAC_i occurring in G_h
   end for
   return EQ-Count − NEQ-Count

The proposed fitness function in the FAPSO-based approach differs significantly from conventional fitness functions used in PSO-based reachability analysis. Traditional fitness functions in PSO for reachability checking often rely on heuristic distance measures, such as the number of transitions needed to reach a target state or the minimization of unexplored states. These conventional methods, while effective, do not dynamically adapt to the structure of the state space, leading to inefficiencies when dealing with complex models. In contrast, the proposed FAPSO fitness function incorporates a similarity-based evaluation between the last state in the explored path and the target reachability property, leveraging GTS and fuzzy logic for adaptive control.

The proposed FAPSO fitness function diverges significantly from conventional fitness mechanisms used in evolutionary algorithms such as GA, PSO, or ACO for model checking. Traditional approaches often use generic heuristics like path length minimization or transition count, which are agnostic to the structural and semantic characteristics of the model under analysis. In contrast, the FAPSO fitness function is intricately designed for GTS and leverages the structural similarity between the terminal node of a search path and the graph representing the reachability property.

This graph-based similarity measure ensures that paths ending in states closely resembling the target property receive higher fitness scores. Moreover, the penalty for encountering elements defined in NACs significantly enhances the precision of the search by avoiding invalid or misleading states. This nuanced handling of graph semantics is unattainable using traditional scalar fitness measures and makes FAPSO particularly effective in complex software models where state semantics play a critical role.

3.4. PSO-Based Approach

The first proposed approach applies the PSO algorithm to search the state space and find a path that starts from the initial state and ends at a state that satisfies the given reachability property. As previously mentioned, each particle in this algorithm is encoded as a path of transitions. The algorithm begins with an initial random population of particles. Then, the fitness value of each particle is calculated, and the global best (g_{best}) and personal best (p_{best}) are updated based on the fitness values of the particles. Next, the termination condition is checked. If the updated *gbest* is a perfect solution or if the current iteration number exceeds the predefined maximum number of generations, the algorithm terminates.

Otherwise, Equations (1) and (2) are used to update each particle's velocity and position, respectively. The algorithm continues to run, repeatedly calculating fitness, until one of the termination conditions is met.

3.5. FAPSO-Based Approach

Another approach proposed in this paper applies the Fuzzy Adaptive PSO algorithm, referred to as FAPSO, to verify the reachability property of systems defined using GTS. In traditional PSO, the parameters C1 and C2 are constant values that do not change throughout the generations. However, better results can be achieved by dynamically adjusting C1 and C2 during the algorithm's execution. In FAPSO, fuzzy inference systems are used to adapt C1 and C2 in each generation. Like the standard PSO algorithm, the proposed FAPSO begins with an initial population of randomly generated particles. The fitness value of each particle is then calculated, and the global best (*gbest*) and personal best (*pbest*) are updated based on the fitness values of the particles. If the termination condition is not met, two input parameters for the fuzzy system—Diversity and Iteration—are calculated. The diversity measure reflects the dispersion of particles, the greater the separation between particles, the higher the diversity. As defined in Equation (3), the diversity measure can be the average Euclidean distance between each particle's position and the best position in the corresponding generation [32].

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

The second input to the fuzzy system is the percentage of iterations, calculated using Equation (4). At the beginning of the algorithm, the "Iteration" value is considered "Low," and it gradually increases as the number of algorithm iterations approaches the maximum iteration limit [32].

$$\text{Iteration} = \frac{\text{Current Iteration}}{\text{Maximum of Iteration}} \tag{4}$$

The two measures mentioned above are used as inputs to the fuzzy system, which adjusts C1 and C2. These values, C1 and C2, are the outputs of the fuzzy system. It is important to note that the fuzzy system's inputs are constrained within the range [0, 1]. While the Iteration variable can be directly defined within the acceptable range of values, the Diversity measure requires normalization to convert it into a value between 0 and 1. The normalization process applied to Diversity is shown in Equations (5) and (6). Equation (5) illustrates that when the maximum and minimum Euclidean distances are equal, the normalized Diversity is 0, indicating that the particles' positions have not changed. If the maximum and minimum Euclidean distances differ, the normalized Diversity is calculated using Equation (6) [32].

$$\text{Normal Diversity} = \begin{cases} 0 & \text{MinDiversity} = \text{MaxDiversity} \\ \text{Norm} & \text{MinDiversity} \neq \text{MaxDiversity} \end{cases} \tag{5}$$

$$\text{Norm} = \frac{Diversity - MinDiversity}{MaxDiversity - MinDiversity} \tag{6}$$

The choice of fuzzy membership functions considerably impacts the FAPSO algorithm's performance. Different membership functions influence how the algorithm dynamically adjusts the PSO parameters (C1 and C2). The fuzzy system in FAPSO is designed to enhance exploration and exploitation by modifying these coefficients based on diversity and iteration count. If inappropriate membership functions are chosen, it can lead to

premature convergence or ineffective exploration, reducing the efficiency of reachability analysis in large state spaces.

In this approach, Mamdanı′s fuzzy system inference method with two input variables of Iteration and Diversity and two output variables of C1 and C2 is proposed. Figure 2 presents the proposed fuzzy system.

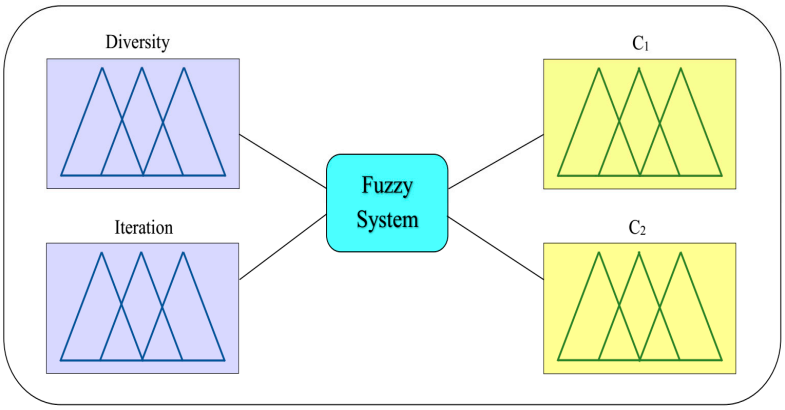

Figure 2. Architecture of the Fuzzy System in the Proposed Method.

For each input of the fuzzy system, three triangular membership functions are designed [32]. Figures 3 and 4 illustrate the membership functions used for the input variables of Iteration and Diversity, respectively. Since it is recommended to select C1 and C2 within the range of [0.5, 2.5] [33], the output variables are adjusted to fall within this range. As shown in Figures 5 and 6, the output variables of C1 and C2 are represented by five triangular membership functions. Tables 1 and 2 present the fuzzy system's rule sets. It is important to note that two key points must be considered when defining the fuzzy system's rules.

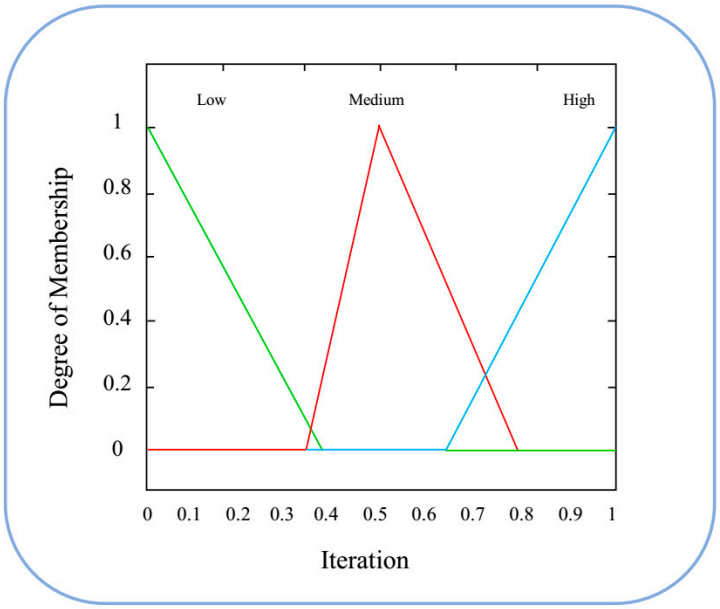

Figure 3. The membership functions for the input variable 1: Iteration.

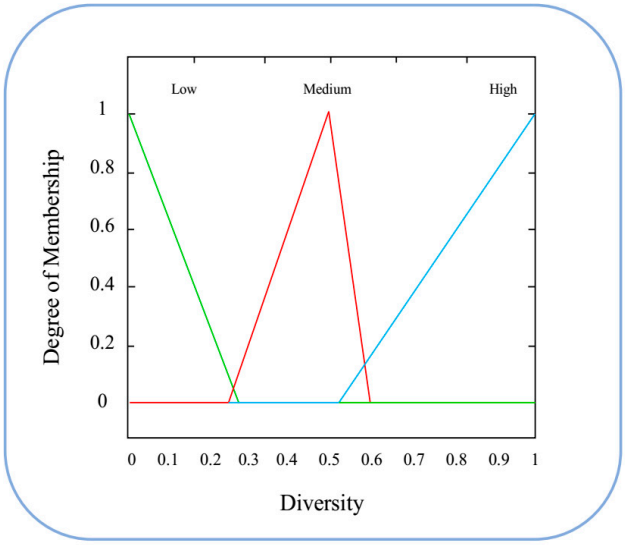

Figure 4. The membership functions for the input variable 2: Diversity.

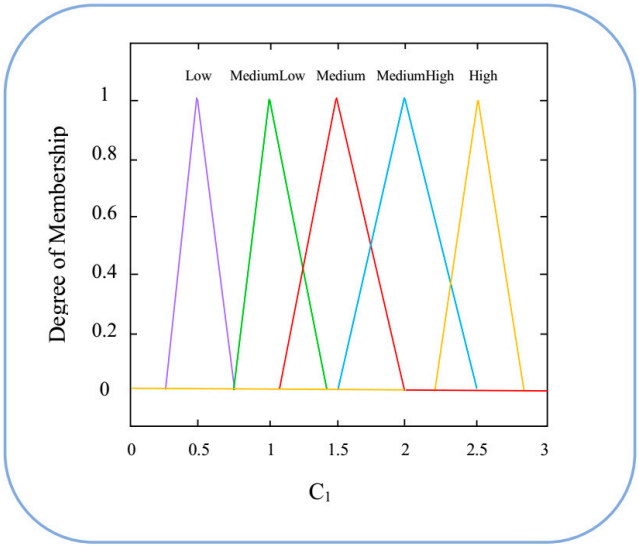

Figure 5. The membership functions for the output variable 1: C_1.

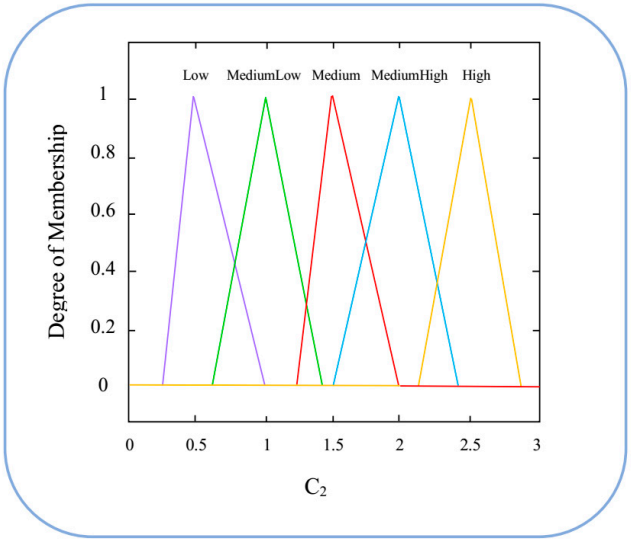

Figure 6. The membership functions for the output variable 2: C_2.

Table 1. Rule set of fuzzy system to calculate C_1.

Diversity \ Iteration	Low	Medium	High
Low	High	MediumHigh	Medium
Medium	MediumHigh	Medium	MediumLow
High	MediumHigh	MediumLow	Low

Table 2. Rule set of fuzzy system to calculate C_2.

Diversity \ Iteration	Low	Medium	High
Low	Low	MediumLow	High
Medium	Medium	Medium	MediumHigh
High	MediumLow	MediumHigh	High

The first point is that exploration should be prioritized in the early iterations of the PSO algorithm to eventually exploit the desired solutions. The second point is that exploration should occur when the diversity is low, whereas exploitation should take place when the diversity is high, meaning when the particles are spread out [32,33].

Once the fuzzy system's input variables, namely Diversity and Iteration, are calculated using Equations (3) and (4), their membership percentages are determined in the fuzzification phase through the membership functions shown in Figures 5 and 6. Each membership function is categorized into three levels: high, medium, and low. In the next step, the rules defined by the inference system, as shown in Tables 1 and 2, are applied to calculate C1 and C2. The values obtained from the inference system represent the percentage of membership for C1 and C2 across five categories: high, medium-high, medium, medium-low, and low. The defuzzification method used is the centroid method, which calculates the final values of C1 and C2 through the output membership functions presented in Figures 7 and 8, respectively.

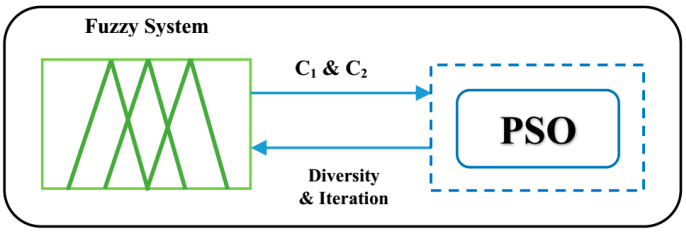

Figure 7. The main idea of the FAPSO approach.

As shown in Figure 7, the proposed approach dynamically adjusts the PSO algorithm's parameters in each iteration by utilizing a fuzzy system.

Once the values of C1 and C2 are obtained from the fuzzy system, the PSO algorithm updates each particle's velocity using Equations (1) and (2). The fitness calculation phase then continues until one of the termination conditions is met.

The design of the fuzzy inference system in the FAPSO approach is structured to dynamically adjust the PSO parameters C1C_1C1 and C2C_2C2, thereby enhancing the search process for reachability analysis in GTS. The system incorporates two input variables—Diversity and Iteration—to represent the dispersion of particles and the progress of the algorithm, respectively. These inputs are mapped to three triangular membership functions (Low, Medium, High) to balance exploration and exploitation effectively. The output variables, C1C_1C1 and C2C_2C2, are assigned five membership functions (Low,

Medium-Low, Medium, Medium-High, High), ensuring finer granularity in adjusting the cognitive and social coefficients. The fuzzy rule set is constructed to prioritize exploration in early iterations and shift towards exploitation as the algorithm converges, preventing premature stagnation. Sensitivity analyses confirm the robustness of this adaptive approach, demonstrating that variations in the membership functions or rule sets have a minimal impact on the overall convergence trend while significantly reducing computational overhead. The experiments show that the adaptive fuzzy system maintains optimal balance across different problem sizes, leading to improved accuracy and faster convergence compared to fixed-parameter PSO approaches.

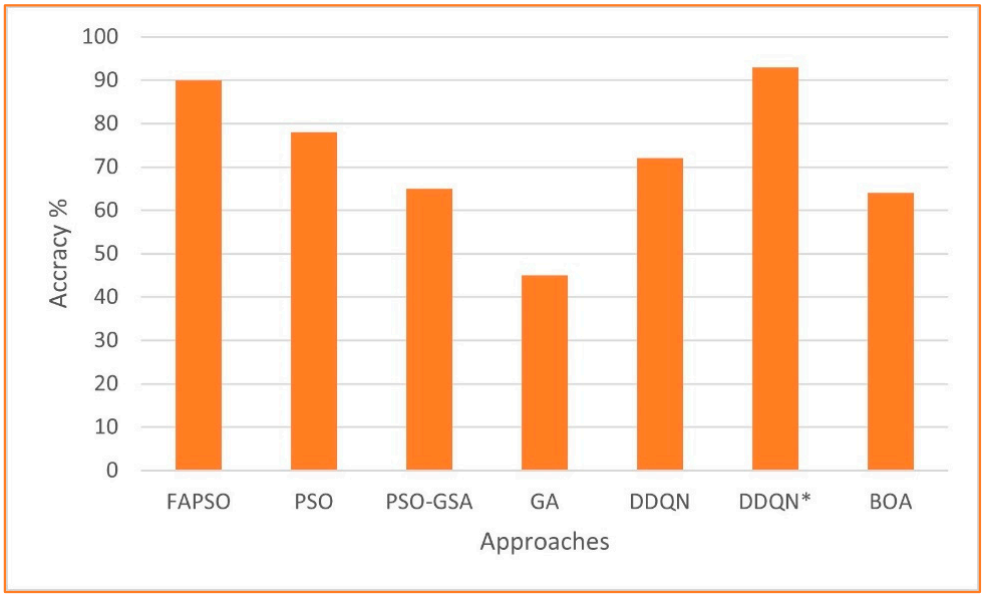

Figure 8. Comparing the accuracy of the proposed approaches to the existing methods.

4. Experimental Results

4.1. Basic Models

The approaches were implemented in the GROOVE toolset using the Java programming language to evaluate and compare their performance. Some existing classes in GROOVE were modified, and new classes were created to better implement the approaches. GROOVE was selected for its robust capabilities in modelling and verifying systems based on GTS, which correspond directly to the formalism applied in this study. Its support for fully customizable transformation rules, compatibility with Java (the language used for implementation), and ease of integration with external algorithms made it a suitable environment. As clarified in Section III, GROOVE serves primarily as the modelling and simulation platform, while the FAPSO algorithm and the associated fitness function were implemented independently. The approach remains adaptable to other model-checking tools that support GTS or similar semantics, ensuring that it is not bound to a specific platform and retains broad applicability.

For evaluation purposes, several models were considered, including the dining philosophers [34], Pac-Man [29], shopping [35], process life cycle, N-Queen, and 8-puzzle [36] models. These models are available for download on the web. The initial parameters used in the PSO algorithm are listed in Table 3. The values of W and Iteration in the FAPSO algorithm are the same as those used in the PSO method. A PC with an Intel CORE i5 processor and 3 GB of memory was used for the experiments. It should be noted that parameters such as depth limit and population size are not fixed universally but are instead defined in accordance with the characteristics of each specific problem and the

complexity of the corresponding model. As the model size increases, the associated state space becomes significantly more expansive and complex, which in turn increases the difficulty of locating the desired goal state within a reasonable time frame. In such cases, setting higher values for the depth limit allows particles to explore longer transformation paths, while a larger population size increases the diversity of candidate solutions, enhancing the algorithm's ability to avoid premature convergence. These parameter adjustments are therefore essential to maintain search effectiveness and improve convergence in large-scale or highly interconnected models. Failing to scale these parameters appropriately may result in suboptimal performance, including missed goal states or excessive computation times. As such, careful tuning of these values is a critical aspect of applying the FAPSO approach to varying problem domains.

Table 3. Initial parameters of the PSO-based approach.

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

It is important to note that the Genetic Algorithm proposed in [2] and the PSO-GSA approach in [18] were modified to compare the efficiencies of the proposed methods. These two approaches were originally designed to evaluate the safety property of systems specified through GTS by detecting deadlock states. To adapt them for verifying reachability properties, we replaced their original fitness functions with the one presented in Section 4.2.

To determine an appropriate population size and depth limit, the proposed approaches were tested across a wide range of values. The results showed that a larger population size enhances exploration but also demands more computational resources, while a smaller population size may lead to suboptimal solutions due to limited search diversity. Additionally, a higher depth limit allows for the exploration of a larger state space, increasing the chances of finding a reachable state, but it also increases memory usage and computation time.

The experimental results from the paper indicated that larger population sizes and depth limits improve accuracy, but they come with trade-offs in execution time and memory consumption. Therefore, we selected the optimal values for population size and depth limit based on the tested results.

The experimental results, averaged over 20 independent runs, are presented in two tables for each case study.

4.1.1. Dinning Philosopher's Problem

Dinning philosopher's problem was first introduced by E.W. Dijkstra. In this scenario, several philosophers are seated around a table, with a fork placed between each pair of adjacent philosophers. After thinking, the philosophers become hungry. Each philosopher picks up the left and right forks to use them for eating. A philosopher can only begin eating if they have both the left and right forks. After eating, the philosopher places the forks back on the table and resumes thinking. This process continues until a deadlock occurs, where *all philosophers are waiting for their right fork after already having picked up their left fork* [34]. This deadlock state is the reachability property that is checked in various versions of this problem. The results obtained from applying the proposed approaches to verify the reachability property in this problem are presented in Table 4.

Table 4. Comparing the average time to verify the reachability property in the dining philosophers problem for all existing approaches.

Number of Philosophers	Depth Limit	Population	FAPSO (Second)	GA (Second)	PSO (Second)	PSO-GSA (Second)
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

4.1.2. Pac-Man Game Problem

In the Pac-Man game, there are three types of objects: Pac-Man, marbles, and ghosts [29]. According to the game rules, both Pac-Man and the ghosts can move to an adjacent box during each stage. If Pac-Man moves to a new box and there is a marble in it, he eats the marble. However, if a ghost moves into the same box as Pac-Man, the ghost kills Pac-Man. The game ends when all marbles are eaten, or Pac-Man is killed by a ghost. In this scenario, the following reachability property must be checked: *Pac-Man wins by eating all the marbles*. The results obtained from applying different approaches to the Pac-Man game problem are presented in Table 5.

Table 5. Comparing the average time to verify the reachability property in the Pac-Man Game problem for all existing approaches.

Dimension of Pac-Man Game	Depth Limit	Population	FAPSO (Second)	GA (Second)	PSO (Second)	PSO-GSA (Second)
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.

4.1.3. Process Life Cycle Problem

The process life cycle describes the stages associated with the life cycle of a process in an operating system. The cycle begins with the creation of a new process, which is then loaded into memory, provided there is enough available space. Afterward, the process waits for I/O devices or the CPU. Once the process has completed execution, all allocated resources are released, and the process terminates. The reachability property to be verified in the models of this problem is: *All processes have been completed*. The results obtained from various approaches for the process life cycle problem are presented in Table 6. As shown in the table, the FAPSO approach, unlike the other proposed methods, is able to find the given reachability property even as the dimensions of the problem increase.

Table 6. Comparing the average time to verify the reachability property in the process life cycle problem for all existing approaches.

Process Life Cycle	Depth Limit	Population	FAPSO (Second)	GA (Second)	PSO (Second)	PSO-GSA (Second)
20 process 8 memory	180	20	7.08	6.58	37.09	17.66

Table 6. *Cont.*

Process Life Cycle	Depth Limit	Population	FAPSO (Second)	GA (Second)	PSO (Second)	PSO-GSA (Second)
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		

4.1.4. Shopping Problem

The shopping problem pertains to the customer purchase process in a store, originally presented in [35]. The reachability property considered in this case is as follows*: all customers have successfully completed their shopping*. Table 7 presents the results obtained from different approaches for the shopping problem.

Table 7. Comparing the average time to verify the reachability property in the shopping problem for all existing approaches.

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

4.1.5. N-Queen Problem

The N × N chessboard and N queens form the elements of this problem. The goal is to place the queens on the chessboard in such a way that no queen can attack another. In chess, each queen can move horizontally, vertically, or diagonally as far as she wants, and two queens can threaten each other if they share the same row, column, or diagonal. Therefore, the acceptable arrangement is one where no two queens share the same row, column, or diagonal. The reachability property considered in the various models of this problem is*: all queens are placed in positions where none can threaten another.* Table 8 presents the results obtained by different approaches for the N-Queens problem.

Table 8. Comparing the average time to verify the reachability property in the N-Queen problem for all existing approaches.

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

4.1.6. 8-Puzzle Problem

In this problem, there is a nine-box board where eight boxes are filled with numbered tiles (from 1 to 8) and one box remains empty [36]. A tile can move into the empty box if it

is adjacent to it. The goal of the game is to start with an arbitrary configuration of tiles and arrange the numbers in ascending order. Table 9 presents the results obtained by different approaches for this problem.

Table 9. Comparing the average time to verify the reachability property in the 8-puzzle problem for all existing approaches.

Initial Arrangement	Depth Limit	Population	FAPSO (Second)	GA (Second)	PSO (Second)	PSO-GSA (Second)
	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

4.1.7. Positioning FAPSO Against Machine Learning-Based Approaches

Recent advancements in deep learning, particularly reinforcement learning using methods like Double Deep Q-Networks (DDQN), have been proposed for state-space exploration in software verification [24]. Table 10 indicates the comparison of the average time for the reachability verification via FAPSO and two novel approaches based on Deep Reinforcement Learning. DDQN and DDQN* approaches use the double deep q-network method. The term DDQN* refers to the method implemented via reinforcement learning which uses a specific reward function for each problem. DDQN approach is referred to the reinforcement learning based method with general rewards. These methods demonstrate high potential in learning generalized strategies for state exploration. However, their training requirements, computational complexity, and lack of interpretability pose significant limitations.

Table 10. Comparing the average time of FAPSO with the reinforcement learning-based algorithms.

Approaches	8-Puzzle (Second)				N-Queen (Second)			Dining Philosophers (Second)		
					8	12	16	40	70	90
FAPSO	2.77	6.57	34.93	102.5	3.07	12.54	24.17	55.23	97.18	131.11
DDQN *	1.98	6.04	33.9	836.45	21.54	34.33	87.85	2.09	4.07	8.27
DDQN	2.32	5.65	195.6	1759.55	40.78	242.57	522.58	12.04	75.46	265.23

While reinforcement learning methods like DDQN offer strong generalization, they demand substantial training and yield less interpretable outcomes. In contrast, FAPSO delivers competitive or superior performance with significantly lower computational overhead and offers explainable paths tailored to GTS structures. It does not require pre-training, adapts dynamically through fuzzy logic, and produces deterministic, interpretable search paths. As shown in Table 10, FAPSO consistently outperforms DDQN and DDQN* in large-scale models in terms of execution time, especially as model complexity increases. While DDQN variants suffer from scalability issues (e.g., exceeding memory limits in the 8-puzzle and process life cycle models), FAPSO maintains robust performance. This demonstrates that while ML-based methods are promising, FAPSO provides a more practical and scalable solution for GTS-based reachability analysis.

4.2. Models Applied for Rule Composition

As mentioned in the introduction, the proposed approach for rule composition in [26] not only improves the efficiency of model checking algorithms by reducing their runtime, but also minimizes the number of explored states by eliminating unnecessary intermediate states. This reduction in the number of explored states significantly decreases the memory consumption required for executing the algorithm.

In this section, the basic models for the dining philosophers and shopping problems, previously discussed, have been refined using the principles of rule composition [26]. As a result, these models now contain fewer rules. However, it is essential that rule composition is applied carefully, ensuring that the core specifications of the model are not compromised. Furthermore, attention must be given to avoid inadvertently removing critical intermediate states during the process.

4.2.1. Dinning Philosopher's Problem

The dining philosophers problem consists of five transformation rules, as follows:

1. **GoHungry**: The philosopher transitions from the "Thinking" state to the "Hungry" state.
2. **GetLeft**: The philosopher picks up the left fork and transitions to the "HasLeft" state.
3. **GetRight**: The philosopher picks up the right fork and transitions to the "Eating" state.
4. **ReleaseLeft**: The philosopher releases the left fork and transitions to the "HasRight" state.
5. **ReleaseRight**: The philosopher releases the right fork and returns to the "Thinking" state.

During the rule composition stage, the first and second rules are merged, resulting in the following updated rule set:

1. **GoHungry-GetLeft**: The philosopher picks up the left fork and transitions from the "Thinking" state to the "HasLeft" state.
2. **GetRight**: The philosopher picks up the right fork and transitions to the "Eating" state.
3. **ReleaseLeft**: The philosopher releases the left fork.
4. **ReleaseRight**: The philosopher releases the right fork and returns to the "Thinking" state.

In the modified model, the "Hungry" state is removed, and the philosopher transitions directly from the "Thinking" state to the "HasLeft" state. This combination does not violate any specifications of the dining philosophers problem and reduces the number of rules to four.

The results of applying the proposed approaches to the modified dining philosophers model are presented below.

Table 11 compares the average time required to verify the reachability property in both the basic and modified dining philosophers models using the proposed approaches. Table 12 shows the average number of explored states required to verify the reachability property in both the basic and modified models.

Table 11. Comparing the average time to verify the reachability property in the *basic and* modified dining philosophers model for proposed approaches.

Number of Philosophers	Depth Limit	Population	Basic Model		Modified Model	
			FAPSO (Second)	PSO (Second)	FAPSO (Second)	PSO (Second)
10	50	15	2.81	8.12	1.77	7.05
20	100	20	29.2	85	11.93	18.62
30	200	60	49.86	137	15.18	37.87
40	200	60	Out of Memory		66.4	304.1
60	200	60	Out of Memory		98.6	389.6

Table 12. Comparing the average number of explored states to verify the reachability property in the modified dining philosophers model for proposed approaches.

Number of Philosophers	Depth Limit	Population	Basic Model		Modified Model	
			FAPSO	PSO	FAPSO	PSO
10	50	15	1124	3212	64	95
20	100	20	5080	7730	273	531
30	200	60	5766	8450	465	850
40	200	60	Out of Memory		972	2422
60	220	60	Out of Memory		6177	11,046

As the results indicate, applying rule composition by merging just a pair of rules and reducing the total number of rules by one has a significant impact on reducing both the running time of the algorithms and the number of explored states.

4.2.2. Shopping Problem

The shopping problem consists of nine rules, including:

1. **TakeCart:** The customer initiates the shopping process by picking up a cart.
2. **CreateBill:** An unpaid bill is generated for the customer.
3. **SelectGood:** The customer picks up an item from the rack and places it in their cart.
4. **DeselectGood:** The customer returns an item from the cart to the rack.
5. **BillGood:** The customer′s items in the cart are added to the bill to begin the payment process.
6. **PayBill:** The customer′s bill is paid once all items have been added to the bill.
7. **SettleBill:** The shopping process concludes, and the customer′s shopping flag is deactivated.
8. **Finish:** The customer releases the cart, and their bill is deleted.

In the resulting model after applying rule composition, two pairs of rules are merged, reducing the total number of rules to six. The modified rule set is as follows:

1. **TakeCart-CreateBill:** The customer initiates the shopping process by picking up a cart, and an unpaid bill is generated for the customer.
2. **SelectGood:** The customer picks up an item from the rack and places it in their cart.
3. **DeselectGood:** The customer returns an item from the cart to the rack.
4. **BillGood:** To begin the payment process, the customer′s items in the cart are added to their bill.
5. **PayBill-SettleBill:** The customer′s bill is paid once all items have been added, and the shopping process concludes by deactivating the customer′s shopping flag.
6. **Finish:** The customer releases the cart, and their bill is deleted.

Applying these rule compositions does not violate any of the specifications of the shopping problem, nor does it eliminate any essential intermediate states. Table 13 compares the average time required to verify the reachability property in both the basic and modified models of the shopping problem using the proposed approaches. Meanwhile, Table 14 presents the average number of explored states needed to verify the reachability property for both the basic and modified models of the problem, as evaluated through the proposed methods.

Table 13. Comparing the average times to verify the reachability property in the modified shopping problem for proposed approaches.

Shopping Dimension	Depth Limit	Population	Basic Model		Modified Model	
			FAPSO (Second)	PSO (Second)	FAPSO (Second)	PSO (Second)
10 customer 30 good	160	20	2.45	3.89	0.98	2.5
15 customer 30 good	170	30	10.53	34.5	2.16	8.41
20 customer 30 good	180	40	33.58	Out of Memory	2.82	12.58
25 customer 30 good	180	50	Out of Memory		3.24	13.16
30 customer 30 good	180	50	Out of Memory		5.61	14.3

Table 14. Comparing the average number of explored states to verify the reachability property in the modified shopping problem for proposed approaches.

Shopping Dimension	Depth Limit	Population	Basic Model		Modified Model	
			FAPSO	PSO	FAPSO	PSO
10 customer 30 good	150	20	243	287	133	142
15 customer 30 good	160	30	482	603	131	151
20 customer 30 good	170	40	765	Out of Memory	155	161
25 customer 30 good	180	50	Out of Memory		152	158
30 customer 30 good	180	50	Out of Memory		167	164

As the results show, applying rule composition—by merging a pair of rules and reducing the total number of rules by one—significantly reduces the algorithm's running time and the number of explored states. This improvement is achieved by simplifying the modeL′s structure, removing redundant or unnecessary intermediate states, and optimizing the overall execution process. Reducing the number of rules not only streamlines the model but also minimizes computational overhead, resulting in faster processing and lower memory consumption. Additionally, this approach preserves the model′s essential specifications while enhancing efficiency, making it especially beneficial for larger and more complex problem dimensions, where performance gains are even more noticeable.

To ensure that the essential semantics of the original model are preserved during rule composition, the following techniques have been applied:

- Validation through Formal Methods: The modified rules have been formally verified to ensure that they do not change the expected system behaviour.
- Checking Intermediate States: Rule composition did not remove critical intermediate states that affect the reachability of essential states.

- Structural Consistency: The graph transformation rules maintain structural constraints to ensure the logical flow remains intact.

As the results indicate, some models lead to "out-of-memory" errors, especially when using competing algorithms such as GA and PSO-GSA. In these cases, the reported computation time reflects the actual runtime until either a solution was found, or the execution reached the maximum timeout limit of one hour for complex models.

The experiments were designed to cover both classical benchmark models and complex transformation structures. Time to verification, number of explored states, and convergence trends were used as metrics to evaluate the effectiveness, efficiency, and robustness of the proposed approach.

Although the experiments were conducted on a resource-constrained hardware setup (Intel i5 with 3 GB RAM) to evaluate algorithmic robustness under practical limitations, we recognize that modern applications typically run on more powerful platforms. The consistent superiority of the FAPSO approach, even under such constraints, suggests its scalability and potential applicability in larger-scale settings.

Furthermore, while benchmark datasets were employed in this study due to their established role in model checking research, we acknowledge that real-world GTS-based models (such as software workflow systems, distributed sensor networks, or enterprise architecture models) present more dynamic and irregular transformation patterns. Future work will incorporate such real-world datasets to validate the applicability of FAPSO in industrial and operational environments, addressing the evolving needs of software verification in contemporary systems.

5. Discussion

The advantages and limitations of the proposed approaches are outlined as follows.

As explained in the literature, previous approaches focus on detecting deadlock states to refute the safety properties of GTS-specified systems. In contrast, the FAPSO approach has been applied to verify the correct reachability property in problems such as the dining philosophers, process life cycle, and 8-puzzle, and the results have been compared with those from evolutionary approaches that aim to refute safety properties through deadlock detection. Table 15 indicates the average time to refute safety by reachability property for FAPSO, BOA, GA, PSO, PSO-GSA, Beam Search (BS), and a Greedy Best Fit Algorithm (BFA).

An important strength of the FAPSO approach is its accuracy in verifying reachability properties. Accuracy is measured as the ratio of successful runs—those in which the algorithm correctly identifies a valid path—to the total number of executed runs. A higher number of successful runs directly correlates with improved accuracy, indicating the algorithm's effectiveness and reliability across multiple trials. Figure 8 presents a comparative analysis of the accuracy achieved by the proposed methods across all evaluated problems, demonstrating FAPSO's superior performance in consistently verifying the reachability of the specified properties.

As the previously presented results indicate, the execution speed of these approaches, particularly FAPSO, is faster than that of the others. Additionally, the proposed approaches generate shorter counterexamples/witnesses compared to the other methods. The chart in Figure 9 compares the length of the witnesses produced by different approaches to verify reachability in the dining philosophers problem, specifically in the case of 10 philosophers.

Table 15. 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 (Second)	BOA (Second)	GA (Second)	PSO (Second)	PSO-GSA (Second)	BFA (Second)	BS (Second)
dining philosophers (10 philosophers)	25	15	4.41	0.71 ± 0.15	10.12	13.45	38.92	0.94	3.85
dining philosophers (20 philosophers)	100	20	29.2	1.04 ± 0.15	23	158	170	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	Not found	Not found
process life cycle (50-process-8-memory)	450	80	40.95	1.81 ± 0.39	Not found	Not found	Not found	Not found	Not found
8-puzzle (Second argument)	100	50	6.57	1.15 ± 0.33	35.81	94.72	16.7	0.16	1.33
8-puzzle (Third argument)	100	60	34.93	Not found	165	165.51	147.7	3.5	2.33

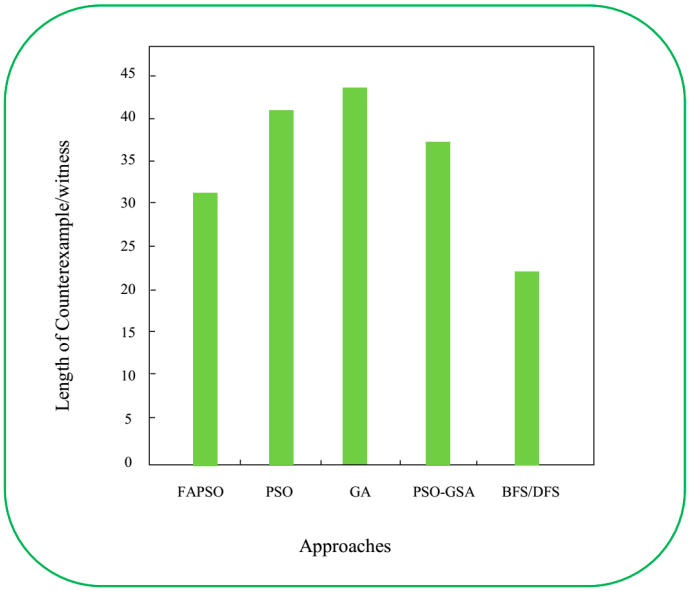

Figure 9. Comparing the length of witness of the implemented methods to reachability verification in the dining philosophers problem with 10 philosophers.

It is important to note that generating shorter counterexamples/witnesses plays a crucial role in the model checking process. Therefore, the length of the generated counterexamples/witnesses in the proposed approaches was controlled using the depth limit parameter.

The proposed approaches, particularly FAPSO, outperform other methods in terms of the number of explored states. The chart in Figure 10 compares the number of states explored by different approaches to verify the reachability property in the dining philosophers problem, specifically for the case of 10 philosophers.

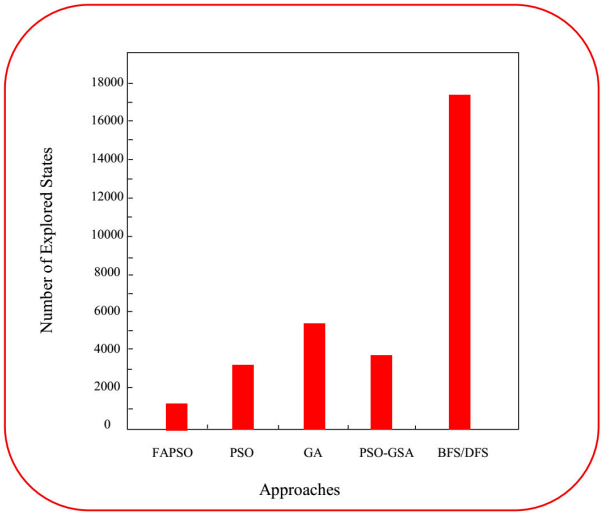

Figure 10. Comparing the explored states number of the proposed approaches to reachability verification in the dining philosophers problem with 10 philosophers.

Figure 11 presents the convergence graph of the proposed approach applied to a dining philosophers model with 10 philosophers. This graph illustrates the progress of the FAPSO algorithm over iterations, showing whether the fitness function improves consistently.

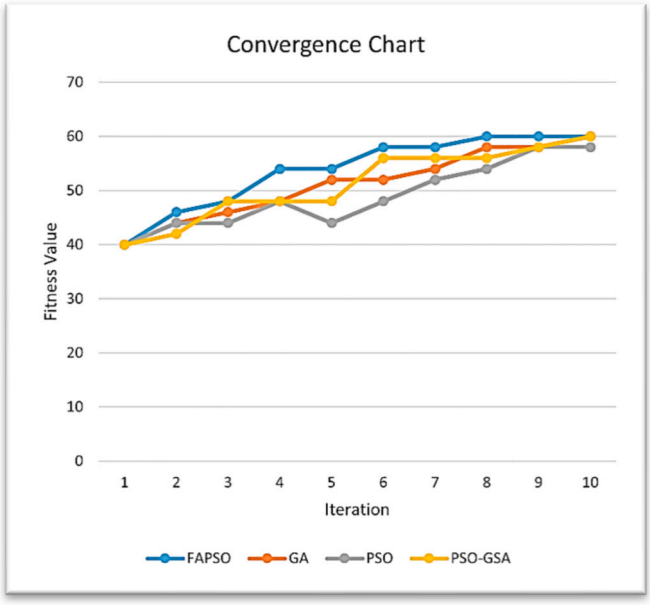

Figure 11. Convergence graph of the proposed approaches to reachability verification for the dining philosophers problem with 10 philosophers.

When comparing the convergence of FAPSO with other methods (PSO, GA, and PSO-GSA), it is evident that the proposed algorithm reaches an optimal solution more quickly. Unlike the PSO approach, it does not stagnate at a suboptimal solution (local optima).

To verify whether there is a significant difference between the results of the proposed method and those of previous techniques, the results were evaluated using the Wilcoxon signed-rank test. This test is a non-parametric statistical hypothesis used to compare related samples. It can be performed in the SPSS Statics v27 software, and if the obtained output (sig) is less than 0.05, it can be concluded that there is a significant difference between the two groups of data. Table 16 presents the Wilcoxon test results for the FAPSO approach.

Table 16. Wilcoxon test results for the FAPSO approach.

Approaches	FAPSO-GA	FAPSO-PSO	FAPSO-BFA	FAPSO-BS	FAPSO-A *	FAPSO-DDQN
Z Asymp. Sig. (2-tailed)	0.001	0.001	0.03	0.007	0.03	0.042

The experimental results demonstrate that the proposed FAPSO approach maintains high efficiency and accuracy in reachability analysis, particularly as model complexity increases. Its adaptive parameter tuning mechanism—driven by a fuzzy inference system—enables dynamic control over exploration and exploitation, helping the algorithm converge faster and more effectively than traditional PSO and other metaheuristic methods. This adaptability is especially valuable in complex state spaces where static parameter settings often lead to premature convergence or inefficient search trajectories.

Despite these strengths, several limitations must be acknowledged. As the size of the model and the complexity of rule compositions grow, FAPSO requires a higher number of fitness function evaluations (FFEs), which increases per-iteration computational overhead. While the adaptive mechanism improves solution quality and reduces the total number of generations needed, the additional cost of diversity assessment and fuzzy rule evaluation introduces a scalability trade-off, particularly for very large models. Furthermore, FAPSO depends on well-defined fuzzy membership functions and rule sets, which may need to be tuned manually for different domains. The algorithm's effectiveness is also influenced by parameters such as population size and depth limit, which must be carefully configured to balance performance and resource usage.

These challenges are not unique to FAPSO. Prior work involving GA, ACO, and hybrid approaches such as PSO-GSA has highlighted similar issues, including sensitivity to parameter tuning and difficulty scaling to larger models. However, unlike these earlier methods, which rely on fixed or semi-static search strategies, FAPSO introduces a feedback-driven adaptation mechanism that adjusts its behaviour based on real-time diversity and iteration progress. This capability reduces the likelihood of stagnation and enhances the search process, especially in environments with evolving structural complexity. Additionally, while many existing algorithms focus primarily on deadlock detection, this study expands the scope to full reachability verification using a graph-aware fitness function tailored to GTS, offering a more semantically aligned solution.

The computational complexity of FAPSO can be analysed in terms of FFEs, with the total cost per iteration proportional to the swarm size and number of generations. Although the fuzzy control system introduces additional processing at each step, this cost is offset by faster convergence and fewer required iterations. As shown in the experimental results, FAPSO consistently outperforms baseline PSO, GA, and PSO-GSA in both execution time and the number of explored states, particularly for complex models. This indicates a net computational gain despite the added per-iteration workload.

To improve scalability further, future work should explore parallelizing particle evaluations and incorporating heuristic-guided pruning techniques to reduce unnecessary exploration. Integrating predictive models or hybrid strategies that combine fuzzy metaheuristics with machine learning could enhance search efficiency and adaptability. Additionally, while this study focused on reachability, extending FAPSO to verify other properties—such as fairness, liveness, or probabilistic behaviour—would require adjustments to the fitness function and search dynamics, offering a promising direction for continued research.

In summary, this study offers several key contributions to the field of model checking and optimization-based software verification. From a theoretical standpoint, it introduces

an adaptive fuzzy mechanism within the PSO framework that dynamically adjusts search behaviour based on system diversity and iteration progress, addressing common limitations of static parameter settings in traditional metaheuristics. The practical implications lie in demonstrating how this approach improves convergence speed, search efficiency, and solution quality for reachability analysis in GTS, making it suitable for use in real-world model verification tasks with large and complex state spaces. Managerially, the findings suggest that organizations adopting model-driven engineering practices can benefit from intelligent, scalable verification techniques like FAPSO to enhance quality assurance and reduce verification time, especially in systems with evolving or modular rule sets. At the same time, the study acknowledges its limitations, including the added computational overhead introduced by the fuzzy logic system, the need for careful parameter tuning, and potential challenges in adapting the method to other modelling formalisms beyond GTS. Addressing these limitations through future extensions, such as parallelized architectures, auto-tuning strategies, and hybrid learning-based guidance, can further enhance the applicability and impact of the proposed approach.

Although the experiments in this study were conducted on a resource-constrained system (Intel i5 with 3 GB RAM) to reflect baseline performance and maintain comparability with earlier approaches, we acknowledge that modern verification environments often benefit from significantly more powerful computing resources. Evaluating the FAPSO algorithm on advanced hardware such as Intel i7 or i9 processors would likely further highlight its computational advantages. This will be considered in future extensions of this work.

Additionally, the choice of a one-hour computation time limit was made to reflect real-world verification practices, where such timeouts are common in toolchains to avoid indefinite execution in complex models. This limit was uniformly applied across all algorithms to ensure fairness in evaluation. Notably, FAPSO consistently converged well within this limit across all tested scenarios, reinforcing its efficiency.

6. Conclusions

This paper presented a Fuzzy Adaptive Particle Swarm Optimization (FAPSO) approach for reachability analysis in complex software systems modelled using Graph Transformation Systems (GTS). By incorporating fuzzy logic into the standard PSO framework, the proposed method dynamically adjusts the cognitive and social coefficients (C1, C2) based on search diversity and iteration progress. This adaptive mechanism enhances the balance between exploration and exploitation, leading to faster convergence and higher accuracy.

Extensive experiments across multiple benchmark models demonstrated that FAPSO outperforms traditional PSO, Genetic Algorithms (GA), and hybrid PSO-GSA in terms of execution time, number of explored states, and solution quality. Furthermore, integrating rule composition techniques reduced unnecessary intermediate states, improving model scalability and efficiency without compromising semantic correctness.

FAPSO's adaptive nature makes it well-suited for analysing large-scale models, offering interpretable and deterministic results without requiring pre-training, unlike many machines learning-based methods. The algorithm is particularly beneficial in contexts where verification must be both efficient and transparent.

Future work will focus on:

- Enhancing the fuzzy system through refined membership functions and rules.
- Extending the fitness function to verify other properties such as safety, liveness, and fairness.
- Applying FAPSO to broader optimization benchmarks and real-world case studies.

- Automating rule composition to reduce manual effort and improve scalability.
- Exploring hybrid integrations with machine learning techniques for further performance improvements.
- Finally, while the current work relied on classic benchmark models, an important direction for future research is the integration of real-world GTS datasets. These may include configurations extracted from software repositories, business process models, and cyber-physical system simulations. Such expansions will allow for deeper insight into the algorithm's adaptability and performance under diverse, large-scale, and non-synthetic conditions.

Overall, the proposed FAPSO algorithm offers a robust, scalable, and interpretable solution for reachability verification in GTS-based models, addressing key limitations of traditional and learning-based approaches.

Author Contributions: Conceptualization, N.S., S.S. and V.R.; methodology, V.R.; software, N.S.; validation, N.S., S.S. and V.R.; formal analysis, N.S.; investigation, N.S.; resources, N.S.; data curation, N.S.; writing—original draft preparation, N.S.; writing—review and editing, S.S. and D.K.; visualization, D.K.; supervision, S.S. and V.R.; project administration, S.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Data Availability Statement: Data used for analysis has been provided in the manuscript, and any additional information desired will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Denil, J.; Jukss, M.; Verbrugge, C.; Vangheluwe, H. Search-Based Model Optimization Using Model Transformations. In *System Analysis and Modeling: Models and Reusability*; Springer International Publishing: Berlin/Heidelberg, Germany, 2014; pp. 80–95.
2. Yousefian, R.; Rafe, V.; Rahmani, M. A heuristic solution for model checking graph transformation systems. *Appl. Soft Comput.* **2014**, *24*, 169–180. [CrossRef]
3. Baresi, L.; Heckel, R. Tutorial introduction to graph transformation: A software engineering perspective. In *Graph Transformation: First International Conference, ICGT 2002, Barcelona, Spain, 7–12 October 2002*; Springer: Berlin/Heidelberg, Germany, 2002.
4. Lafuente, A.L. Symmetry reduction and heuristic search for error detection in model checking. In Proceedings of the Workshop on Model Checking and Artificial Intelligence, Acapulco, Mexico, 10 August 2003.
5. Clarke, E.; McMillan, K.L.; Campos, S.V.A.; Hartonas-Garmhausen, V.I. Symbolic Model Checking. Available online: https://www.cs.cmu.edu/~emc/papers/Conference%20Papers/Symbolic%20Model%20Checking.pdf (accessed on 15 April 2025).
6. Edelkamp, S.; Leue, S.; Lafuente, A.L. Directed Explicit-state model checking in the validation of communication protocols. *Int. J. Softw. Tools Technol. Transf. (STTT)* **2004**, *5*, 247–267. [CrossRef]
7. Gyuris, V.; Sistla, A.P. On-the-fly model checking under fairness that exploits symmetry. *Form. Methods Syst. Des.* **1999**, *15*, 217–238. [CrossRef]
8. Lin, F.J.; Chu, P.M.; Liu, M.T. Protocol verification using reachability analysis: The state space explosion problem and relief strategies. In Proceedings of the ACM Workshop on Frontiers in Computer Communications Technology, Stowe, VT, USA, 11–13 August 1987; ACM: New York, NY, USA; pp. 126–135.
9. Yang, C.H.; Dill, D.L. Validation with guided search of the state space. In Proceedings of the DAC'98: Proceedings of the 35th Annual Design Automation Conference, San Francisco, CA, USA, 15–19 June 1998; ACM: New York, NY, USA, 1998; pp. 599–604.
10. Edelkamp, S.; Reffel, F. OBDDs in Heuristic Search. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 1998; pp. 81–92.
11. Friedman, G.; Hartman, A.; Nagin, K.; Shiran, T. Projected state machine coverage for software testing. In Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis, Rome, Italy, 22–24 July 2002; ACM: New York, NY, USA, 2002; pp. 134–143.
12. Francesca, G.; Santone, A.; Vaglini, G.; Villani, M.L. Ant Colony Optimization for Deadlock Detection in Concurrent Systems. In Proceedings of the 35th IEEE Annual Computer Software and Applications Conference, Munich, Germany, 18–22 July 2011; pp. 108–117.

13. Duarte, L.M.; Foss, L.; Wagner, R.; Heimfarth, T. Model Checking the Ant Colony Optimisation. In *Distributed, Parallel and Biologically Inspired Systems, IFIP Advances in Information and Communication Technology*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 221–232.
14. Alba, E.; Troya, J.M. Genetic Algorithms for Protocol Validation. In Proceedings of the International Conference on Parallel Problem Solving from Nature (PPSN IV), Berlin, Germany, 22–26 September 1996; Springer: Berlin/Heidelberg, Germany, 1996; pp. 869–879.
15. Vardhan, A. Learning to Verify Systems. Ph.D. Thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 2006.
16. Alba, E.; Chicano, F. Finding safety errors with ACO. In Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, London, UK, 7–11 July 2007; pp. 1066–1073.
17. Alba, E.; Chicano, F. Searching for Liveness Property Violations in Concurrent Systems with ACO. In Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, Atlanta, GA, USA, 12–16 July 2008; ACM: New York, NY, USA, 2008.
18. Moradi, M.; Rafe, V.; Yousefian, R.; Nikanjam, A. A Meta-Heuristic Solution for Automated Refutation of Complex Software Systems Specified through Graph Transformations. *Appl. Soft Comput.* **2015**, *33*, 136–149.
19. Pira, E.; Rafe, V.; Nikanjam, A. EMCDM: Efficient Model Checking by Data Mining for Verification of Complex Software Systems Specified through Architectural Styles. *Appl. Soft Comput.* **2016**, *44*, 1185–1201. [CrossRef]
20. Pira, E.; Rafe, V.; Nikanjam, A. Deadlock detection in complex software systems specified through graph transformation using Bayesian optimization algorithm. *J. Syst. Softw.* **2017**, *131*, 181–200. [CrossRef]
21. Partabian, J.; Bagherifard, K.; Rafe, V.; Parvin, H.; Nejatian, S. Checking Reachability Property in Complex Concurrent Software Systems with a Knowledge Discovery Approach. *J. Soft Comput. Inf. Technol.* **2023**, *12*, 41–51.
22. Nejati, F.; Hamid, N.A.W.A.; Koohi, S.Z.; Zadeh, Z.R. An Incremental Optimization Algorithm for Efficient Verification of Graph Transformation Systems. *IEEE Access* **2023**, *11*, 75748–75760. [CrossRef]
23. Pira, E. Using Deep Learning Techniques for Solving AI Planning Problems Specified through Graph Transformations. *Soft Comput.* **2022**, *26*, 12217–12234. [CrossRef]
24. Mehrabi, M.J.; Rafe, V. Using Deep reinforcement learning to search reachability properties in systems specified through graph transformation. *Soft Comput.* **2022**, *26*, 9635–9663. [CrossRef]
25. Lambers, L. Certifying Rule-Based Models Using Graph Transformation. Ph.D. Thesis, Technische Universität Berlin, Berlin, Germany, 2019; ISBN 978-3-8381-1650-1.
26. Große, M.; Presicce, F.P.; Simeoni, M. Refinements of Graph Transformation Systems via Rule Expressions. In *Theory and Application of Graph Transformation: 6th International Workshop, TAGT'98 Paderborn, Germany, 16–20 November 1998*; Springer: Berlin/Heidelberg, Germany, 2000; pp. 368–382.
27. Taentzer, G. Parallel high-level replacement systems. *Theor. Comput. Sci. TCS* **1997**, *186*, 43–81. [CrossRef]
28. Rensink, A.; Schmidt, Á.; Varró, D. Model Checking Graph Transformations: A Comparison of Two Approaches. In *International Conference on Graph Transformation*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 226–241.
29. Heckel, R. Graph Transformation in a Nutshell. *Electron. Notes Theor. Comput. Sci. (ENTCS)* **2006**, *148*, 187–198. [CrossRef]
30. Kennedy, J. Particle Swarm Optimization. In *Encyclopedia of Machine Learning*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 760–766.
31. Kastenberg, H.; Rensink, A. Model Checking Dynamic States in GROOVE. In *Model Checking Software: International SPIN Workshop, Vienna, Austria, 30 March–1 April 2006*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 299–305.
32. Melin, P.; Olivas, F.; Castillo, O.; Valdez, F.; Soria, J.; Valdez, M. Optimal design of fuzzy classification systems using PSO with dynamic parameter adaptation through fuzzy logic. *Expert Syst. Appl.* **2013**, *40*, 3196–3206. [CrossRef]
33. Olivas, F.; Castillo, O. Particle Swarm Optimization with Dynamic Parameter Adaptation Using Fuzzy Logic for Benchmark Mathematical Functions. In *Recent Advances on Hybrid Intelligent Systems*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 247–258.
34. Schmidt, A. Model Checking of Visual Modeling Languages. Master's Thesis, Budapest University of Technology, Budapest, Hungary, 2004.
35. Hausmann, J.H. Dynamic Meta Modeling: A Semantics Description, Technique for Visual Modeling Techniques. Ph.D. Thesis, University of Paderborn, Paderborn, Germany, 2005.
36. Gaschnig, J. Performance Measurement and Analysis of Certain Search Algorithms. Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1979; technical report.