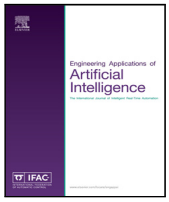
Engineering Applications of Artificial Intelligence 92 (2020) 103684

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai



A novel approach to solve AI planning problems in graph transformations

Einollah Pira

Faculty of Information Technology and Computer Engineering, Azarbaijan Shahid Madani University, Tabriz 5375171379, Iran

ARTICLE INFO

Keywords:
Bayesian Optimization Algorithm
AI planning
Graph transformation system
Bayesian network
Refinement

ABSTRACT

The aim of AI planning is to solve the problems with no exact solution available. These problems usually have a big search space, and planning may not find plans with the least actions and in the shortest time. Recent researches show that using suitable heuristics can help to find desired plans. In planning problems specified formally through graph transformation system (GTS), there are dependencies between applied rules (actions) in the search space. This fact motivates us to solve the planning problem for a small goal (instead of the main goal), extract dependencies from the searched space, and use these dependencies to solve the planning problem for the main goal. In GTS based systems, the nodes of a state (really is a graph) can be grouped due to their type. To create a small (refined) goal, we use a refinement technique to remove the predefined percent of nodes from each group of the main goal. Bayesian Optimization Algorithm (BOA) is then used to solve the planning problem for the refined goal. BOA is an Estimation of Distribution Algorithm (EDA) in which Bayesian networks are used to evolve the solution populations. Actually, a Bayesian network is learned from the current population, and then this network is employed to generate the next population. Since the last Bayesian network learned in BOA has the knowledge about dependencies between applied rules, this network can be used to solve the planning problem for the main goal. Experimental results on four well-known planning domains confirm that the proposed approach finds plans with the least actions and in the lower time compared with the state-of-the-art approaches.

1. Introduction

Planning is a branch of artificial intelligence that concerns about solving the special problems in intelligent systems like agents, autonomous robots, and unmanned vehicles. Planning tries to select a sequence of actions that cause the considered system to transform from the initial state to the goal state. The aim is to find paths (also called plans) with the least actions to reach a specific goal in the shortest time (Russell and Norvig, 2016). Every planning problem includes some components such as actions, states and goals. The way these components are represented is called the planning language. STRIPS is one of the well-known planning languages used in the more general Planning Domain Definition Language (PDDL) (Aeronautiques et al., 1998). In STRIPS problems, a plan is defined by a set of propositional atoms that are meaningful in the problem. In this context, many planners such as Fast-Forward (FF) (Hoffmann and Nebel, 2001), LAMA (Richter and Westphal, 2010), and *k*-Best-first width-search (*k*-BFWS) (Lipovetzky and Geffner, 2017) have been proposed. Since the creation and deletion of objects are not allowed in PDDL, developing a planning system to work directly on graph transformation system (GTS) has recently been taken into consideration. GTS based planning systems usually use domain-independent heuristics in directing the search (Edelkamp et al., 2006; Snippe, 2011; Ziegert, 2014).

In this paper, we propose a novel approach based on a refinement technique and Bayesian Optimization Algorithm (BOA) to solve planning problems in GTS based planning systems. In GTS based systems, the nodes of a state can be grouped due to their type. Therefore, the refinement technique deletes the predefined percent (also called *refPerc*) of nodes from each group of the specific goal state to make a refined goal state. BOA is an Estimation of Distribution Algorithm (EDA) in which Bayesian networks (BNs) are employed to evolve the solution populations (Pelikan et al., 2003). After producing the refined goal state, the proposed approach uses the BOA algorithm to solve the planning problem for this goal. Solving the planning problem for the refined goal state confirms that the last BN learned in BOA has the exact knowledge about dependencies between applied rules, because one of the solutions sampled from this network has led to an optimum solution. The proposed approach employs the last BN to solve the planning problem for the main goal state. To evaluate the effectiveness of the proposed approach, we use the model checker GROOVE as a planning system and implement the approach in it. GROOVE is an open source toolset to design and model checking graph transformation systems (Kastenberg and Rensink, 2006).

The main contributions of the proposed approach are as follows: (1) it extracts the knowledge through solving the refined planning problem and then uses this knowledge to solve the main problem. (2) It employs

E-mail address: pira@azaruniv.ac.ir.

https://doi.org/10.1016/j.engappai.2020.103684
Received 30 December 2019; Received in revised form 25 March 2020; Accepted 28 April 2020
Available online 7 May 2020

evolutionary algorithms and knowledge discovery, for the first time, to solve GTS based planning problems. (3) It beats the state-of-the-art planners in terms of running time and generating short-length plans with the exploration of lower states.

In the rest of the paper, the related work is surveyed in Section 2. In Section 3, we describe the required background such as BOA and GTS formalism. Section 4 describes the proposed approach in details. Experimental results are presented and compared with other approaches in Section 5. Moreover, in this section, we discuss the advantages and limitations of the proposed approach. Finally, we draw the conclusions and ideas for future work in Section 6.

2. Related work

In the state-of-the-art researches, four techniques have been used to solve planning problems. In the first technique, the problem is described by a sufficiently expressive language STRIPS used in the more general Planning Domain Definition Language (PDDL) (Aeronautiques et al., 1998). In STRIPS problems, a plan is defined by a set of propositional atoms that are meaningful in the problem. Based on this technique, many planners such as Fast-Forward (FF) (Hoffmann and Nebel, 2001), LAMA (Richter and Westphal, 2010), and k-Best-first width-search (k-BFWS) (Lipovetzky and Geffner, 2017) have been proposed. The Fast-Forward (FF) planner computes the heuristic using the concept of helpful actions and a GraphPlan-like structure on a relaxed problem. The LAMA planner concentrates on landmarks to improve the performance of FF. k-BFWS is based on novelty measures and width-based search which prunes the states with novelty measure greater than a specific value k (the number of predefined propositional atoms) at each level. The novelty of a state s is i if there exists a set t of i atoms such that (1) s is the first state in which all atoms in t are true. (2) There should not exist any tuple of smaller size having this property. Experimental results over several STRIPS instances show that k-BFWS has a high performance in comparison with FF and LAMA.

In the second technique, planning problems are encoded as instances of boolean satisfiability (SAT) and then benefit from efficient algorithms for solving SAT problems. For example, some planners such as SatPlan and Blackbox (Kautz and Selman, 1998) are based on this technique. Describing planning problems using linear logic is the third technique to solve such problems. In this technique, simultaneous conjunction, linear implication and disjunction are used to respectively represent state, actions and nondeterministic effects. As planners based on this technique, PetriPlan and LinGraph (Kortik and Saranli, 2017; Silva et al., 2000) can be mentioned. PetriPlan formulates a reachability query in an acyclic Petri Net and employs Integer Programming to find a solution. Also, LinGraph exploits the non-monotonicity of linear logic to encode dynamic states effectively. In the fourth technique, a planning system is designed to work directly on graph transformation system (GTS). The works based on this technique usually use domain-independent heuristics in directing the search. For example, the works in Edelkamp et al. (2006), Snippe (2011) propose similarity-based and difference-based heuristic functions that count the nodes and edges that have to be created or deleted to reach a goal state. As another example, the heuristic function proposed in Ziegert (2014) considers the solution length of an abstraction of the problem as an estimate for a given state. Since this article is related to the fourth technique, we concentrate on the approaches based on this technique.

In Edelkamp and Rensink (2007), graph transformation is introduced as a domain for modeling planning problems. GROOVE is also proposed to model graph transformation systems and planning problems. Moreover, it is demonstrated that using a suitable heuristic has many effects on the efficiency of search algorithms of the state space. In Röhs and Wehrheim (2010), the authors proposed an approach which converts a GTS based planning problem into a model checking problem. For this purpose, the property "Is a plan exists?" is given to a model checker. The satisfaction of the property confirms that there exists a plan and the model checker gives a witness as a plan. A witness is a path starting from an initial state and leads to a state in which the reachability property is satisfied. Since a model checker is generally not developed to find a short plan and in the shortest time, the approach cannot outperform other planning techniques. In Edelkamp et al. (2006), several type heuristics such as element counting, formula-based and hamming distance are proposed for using in best-first and A* algorithms in order to solve GTS based planning problems. Moreover, an approach based on A* is proposed in Snippe (2011) to solve such problems. In this approach, two heuristic functions are presented: (1) unweighted element counting. (2) Weighted element counting with predefined weights. The weights depend on the problem and should be specified by the user. Experimental results show that the proposed approach is much better than Breadth-First Search (BFS) and Depth First Search (DFS).

Some approaches like (Pira et al., 2019, 2018, 2016; Rafe et al., 2019), which are presented to verify reachability properties in GTS based systems, can be employed to solve GTS based planning problems. In fact, a goal state of a planning problem can be considered as a reachability property, and the model checker GROOVE modified in these approaches can find a witness as a plan for the planning problem. The approach in Pira et al. (2016), which is called EMCDM, uses data mining techniques for efficient exploration of the state space. To check the complex and large models, EMCDM acquires a knowledge from the information of checking some smaller models. The most important shortage of EMCDM is that designing smaller models in complex systems may be hard (even impossible). To fix the shortage of EMCDM, two approaches are proposed in Pira et al. (2018) which explore a small part of a given model's state space to obtain the required knowledge for checking the given model. The first approach, also called LDM (Learning by Data Mining), employs data mining techniques to find a frequent pattern through the partially explored state space. Whereas, in the second approach, also called LBN (Learning a BN), a BN is learned from the explored state space. This BN holds some dependencies between the applied rules. These approaches then employ the obtained knowledge (i.e. the frequent pattern in LDM and the BN in LBN) to explore the remainder of the model's state space intelligently until a goal state is found. In model checking, a minimum set of rules should be applied to verify a given property. Moreover, the partially explored state space must be so large enough that the extracted frequent pattern or the learnt BN contains this set of rules. Hence, in systems with a large number of transformation rules, LBN and LDM cannot produce such large state space due to state space explosion, and they fail in such systems. The approach in Pira et al. (2019) employed Genetic Algorithm (GA) and BOA to check reachability properties. In the BOA-based approach, the authors considered BNs with fixed structures and proposed three different versions of the approach with different structures. In the first version, which is called BOAcl2 (BOA with a chain of two nodes), the BNs are the type of naïve one. The second version, which is called BOAcln (BOA with a chain of n nodes), uses BNs with the structure of a chain in which each node depends conditionally on the previous node. In the third version, which is called BOActp (BOA with a chain structure and two-parent nodes), the structure of BNs is similar to the BOAcln except that each node in BOActp depends conditionally on two previous nodes. The reported results confirm that BOAcl2 has a better performance in comparison with others. Nevertheless, BOAcl2 has an important shortage as the follows: the learning of BNs is a time-consuming task especially in problem domains with a large number of transformation rules.

Some works such as Particle Swarm Optimization (PSO) (Rafe et al., 2015), Ant Colony Optimization (ACO) (Rafe et al., 2019), Parallel Multiagent Coordination Optimization (PMCO) (Zhang and Hui, 2016), and Cooperative Bat Algorithm (CBA) (Zhang and Hui, 2017) are based on swarm intelligence optimization algorithms. PMCO, which is based on the PSO algorithm, improves the ability to solve large-scale optimization problems and the ability to use more particles in PSO. The

simulation results confirm that PMCO can outperform PSO in terms of the numerical accuracy and the computational time. CBA also presents a graph communication topology and a consensus protocol used in BA to share the velocity information of the bats. Experimental results on several benchmarks show the faster convergence rate of CBA compared with the original BA.

3. Background

3.1. Bayesian optimization algorithm

Bayesian Optimization Algorithm (BOA) is one type of EDA that builds a Bayesian network (BN) through the joint distribution of good partial solutions (Pelikan et al., 2003). EDA belongs to evolutionary algorithms which have found optimum solutions for some hard optimization problems such as Capacitated Vehicle Routing Problems (Tsutsui and Wilson, 2004), Software Testing (Sagarna and Lozano, 2006), Protein Folding (Santana et al., 2008) and Searching the Optimum Path in 3D Spaces (Yuan et al., 2007). In EDA, instead of the traditional genetic operators like crossover and mutation, a probabilistic model is learned from the promising solutions. The learned model is then used to generate new solutions to participate in the next population. BN is a graphical probabilistic model which includes (1) a set of random variables specifying the nodes of the network. (2) A set of directed links between all pairs X and Y of nodes determining conditional dependencies between them. (3) For each node, a table specifying a conditional probability according to the different values of its parents. Formally, a BN represents the Joint Probability Distribution (JPD) of n random variables. Formula (1) shows how to compute every entry in the JPD through the information in the BN:

$$p\left(x_1, x_2, \ldots, x_n\right) = \prod_{i=1}^{n} p(x_i|\text{parents}(x_i)) \tag{1}$$

The details of a BN may be determined by a domain expert or learned from data using machine learning algorithms. There are many approaches such as Bayesian methods, quasi-Bayesian methods, and non-Bayesian methods to learn variables and conditional dependencies between them. Moreover, one method to learn conditional probabilities is to use relative frequencies observed in the data based on the maximum likelihood hypothesis.

BOA produces the initial population randomly. To generate the next population, BOA performs the following operations: (1) it evaluates the current population by the fitness function. (2) It selects a set of most promising solutions and learns a BN through these solutions. (3) It samples the learned BN to generate new candidate solutions and incorporates them into the population using a replacement approach. If the termination criteria such as observing an optimum solution or reaching to a maximum number of iterations are not satisfied, BOA generates again the next population (Pelikan et al., 2003).

3.2. Graph transformation system

GTS is a graph-based formalism which uses graphs and graph transformations to describe states and behaviors of a system respectively (Rozenberg, 1997). An attributed GTS is specified by a tuple (*TG*, *HG*, *R*) where *TG* is a type graph, *HG* is a host graph, and *R* is a set of transformation rules. *TG* specifies all node types and edge types of the system, and also two functions to define source/destination nodes of edges. The initial configuration of the system is specified by *HG* and should be an instance of the type graph *TG*. It means that each component (node or edge) in *HG* should be instantiated from a component (node type or edge type) in *TG*. Each graph transformation rule is specified by a tuple (*LHS*, *RHS*, *NAC*) where *LHS* (also called left-hand side) specifies the conditions that the current host graph should have before applying the rule. *RHS* (also called right-hand side) determines the conditions that should be in the host graph after applying the rule. Moreover, *NAC* (negative application condition) is a condition that its existence in the host graph prevents the application of the rule.

As an example of a system modeled by GTS, we consider the readers–writers problem (Pira et al., 2018). This problem presents a system with several processes (e.g. readers/writers) that want to access (e.g. read/write) the shared resource (e.g. a book) with the following conditions: (1) two or more readers can read the shared resources provided that no writer is writing. (2) More than one writer cannot write to the shared resource simultaneously. Fig. 1 shows a sample model of this problem with 6 readers (denoted by R) and 6 writers (denoted by W) in GROOVE. The type graph of this model (Fig. 1(a)) has three node types n_0 (with type *R*), n_1 (with type *W*), and n_2 (with type *BOOK*) and two edge types n_0n_2 (with type *read*) and n_1n_2 (with type *write*). The allowed labels for n_0/n_1 are *active*, *deactive*, *waitRead/waitWrite*, and etc. Whereas, n_2 can get one of the labels *free*, *reading*, and *writing*. Also, the host graph of the model (Fig. 1(b)) includes the set $\{n_3, n_4, \ldots, n_{11}\}$ of nodes without any edges.

Transformation rules in GROOVE are shown by merging the graphs of *LHS*, *RHS*, and *NAC*, and they distinguished by color coding. To specify some special configurations (also called state properties) in a system, GROOVE uses transformation rules with equal *LHS* and *RHS* graphs. For example, Fig. 1(c) displays a state property of "*all readers are reading and all writers in the waitWrite mode*" for the readers–writers problem with 4 readers and 4 writers. For more information about graph transformations, interested readers can refer to Rozenberg (1997).

4. The proposed approach

In this section, we present an approach based on a refinement technique and Bayesian Optimization Algorithm (BOA) to solve GTS based planning problems. As mentioned in Section 3.2, each system modeled through GTS has a host graph which is based on a type graph. Generally, a host graph is considered as an initial state of the system, and to generate all other states, all transformation rules should be applied on the initial state recursively. Since the graphs *LHS*, *RHS*, and *NAC* of all transformation rules are typed, all possible states such as goal states will also be typed. The application of a rule on a state does not modify the type of its nodes. Hence, the applied rules in transformation of similar-type nodes of the initial state to the ones of each portion of the goal state are equal as well as there are dependencies between themselves. Moreover, the smaller the size (i.e. the number of nodes) of a goal state, the more simpler solving the planning problem. Due to these reasons, the proposed approach, which is called plnBOA (planning with BOA), considers a portion of the goal state so that the portion includes at least one node type of all existing node types in the goal state (such portion is so called the refined goal state). plnBOA then uses the BOA algorithm to solve the planning problem for the refined goal state. Finally, plnBOA employs the last learnt BN, which contains the dependencies between applied rules, to solve the planning problem for the main goal state.

Briefly, plnBOA contains the following steps:

- Step 1: Group the nodes of a given goal state according to their type. Then, remove the *refPerc* percent of nodes from each group to make a refined goal state. These operations are performed by RefGoalState algorithm that its details will be mentioned in Section 4.1.
- Step 2: Employ the BOA algorithm to solve the planning problem for the refined goal state. In Section 4.2, we will discuss this subject.
- Step 3: After solving the problem for the refined goal state, use the last BN learned in BOA to solve the planning problem for the main goal state. This issue will be discussed in Section 4.3.

The plnBOA approach is repeated until a termination criterion such as finding a plan or reaching to a predetermined maximum number of iterations occurs. Fig. 2 shows the architecture of the plnBOA approach.

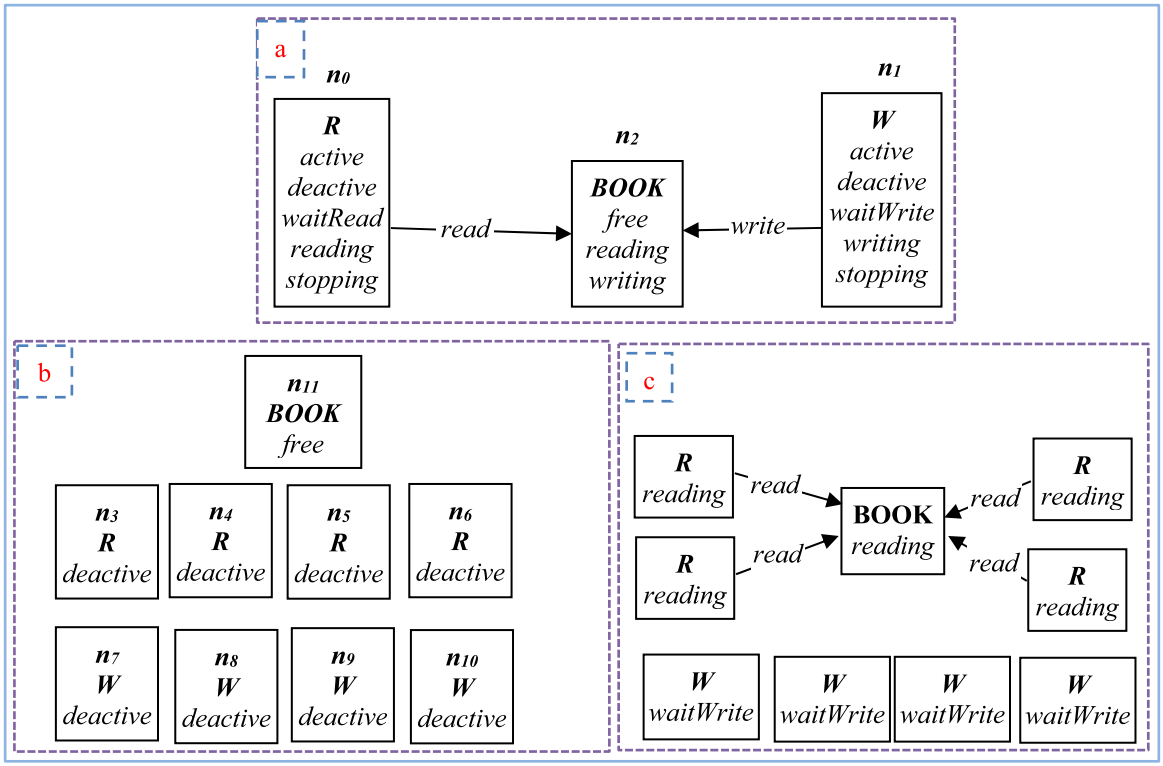


Fig. 1. A GTS-based model of the readers–writers problem designed in the GROOVE toolset.

4.1. The process of the refinement technique

The refinement technique is employed to decrease the size of the given goal state. For this purpose, the RefGoalState algorithm is proposed (Algorithm 1). This algorithm has two important functions: (1) divMGoalToGroups that divides the nodes of the given goal state to different groups according to their type. (2) delFromGroup, which has the main role of the refinement technique, removes the $refPerc$ percent of nodes from each group such that at least one node should remain in the group. Moreover, this function so operates that the connectivity of the goal state is not destroyed as much as possible. Fig. 3 clarifies the operation of RefGoalState algorithm for the goal state of the readers–writers problem (i.e. Fig. 1(c)). As seen in the figure, divMGoalToGroups partitions the goal to three groups with types *R*, *W*, and *BOOK* (the nodes of each group are marked with the same color). The delFromGroup function then removes 50% of nodes of each group, almost 2 nodes from each group but the *BOOK* group in which any node is not removed.

4.2. Solving the planning problem for the refined goal state

After generating a refined goal state, plnBOA exploits the first version of the BOA-based approach presented in Pira et al. (2019), which is called BOAcl2, to solve the planning problem for this goal. BOAcl2 uses a naïve Bayesian network to model the selected chromosomes in each iteration of BOA. Moreover, the structure of BNs is considered as a fixed chain with two nodes. First node encodes the applied rules over the previous states, whereas the applied rules over current states are encoded by the second node. In other words, each sub-path like "$r_i s_k r_j$" in all paths encoded by the chromosomes is mapped to the BN. Also, the probability tables show the probability distribution $p(X_0 = r_i)$ and the conditional probability distribution $p(X_1 = r_j | X_0 = r_i)$ for any rules r_i and r_j in the rules set. More details about BOAcl2 are given in Pira et al. (2019).

4.3. Solving the planning problem for the main goal state

The BNs learned in iterations of BOA have the knowledge about dependencies between applied rules. The quality of these BNs is gradually increased, i.e. the fitness values of sampled chromosomes through the BNs are increased. Reaching BOA to an optimum solution shows that the BN has the exact knowledge about dependencies between applied rules, because one of the solutions sampled from this BN has led to an optimum solution. After solving the planning problem for the refined goal state through BOAcl2, plnBOA uses the last BN to solve the problem for the main goal. To do so, plnBOA repeats the following procedure until a plan with the maximum length *depth* is found or a maximum number of iterations occurs: suppose that the last explored state is marked with *LS*, and so, at first *LS* will be s_0 (the initial state of the model). The first rule to be applied to the state *LS* should be chosen based on the probability table of node X_0, whereas the other rules are selected based on the table of X_1. Also, *LS* should be updated after a rule application. It is noted that the roulette-wheel method is used to select rules. Algorithm 2 shows the pseudocode of this procedure. As shown in the algorithm, after finding a goal state in the search space, a plan will be produced.

5. Experiments

The plnBOA approach is implemented by Java language in the GROOVE toolset. Moreover, to evaluate and compare the effectiveness of plnBOA, we use the state-of-the-art approaches (or planners) BOAcl2 (Pira et al., 2019), LBN (Pira et al., 2018), LDM (Pira et al., 2018), GA (Yousefian et al., 2014), PSO (Rafe et al., 2015), ACO (Rafe et al., 2019), and *k*-BFWS (Lipovetzky and Geffner, 2017). It should be noted that, to have a fair comparison, we have implemented the *k*-BFWS planner in GROOVE. In the following and in Section 5.1, we present the required parameters to run the approaches. Section 5.2 presents the planning domains which are used to evaluate the approaches. In Section 5.3, we present an experiment to compare plnBOA with others in different terms. Section 5.4 proposes another experiment to compare

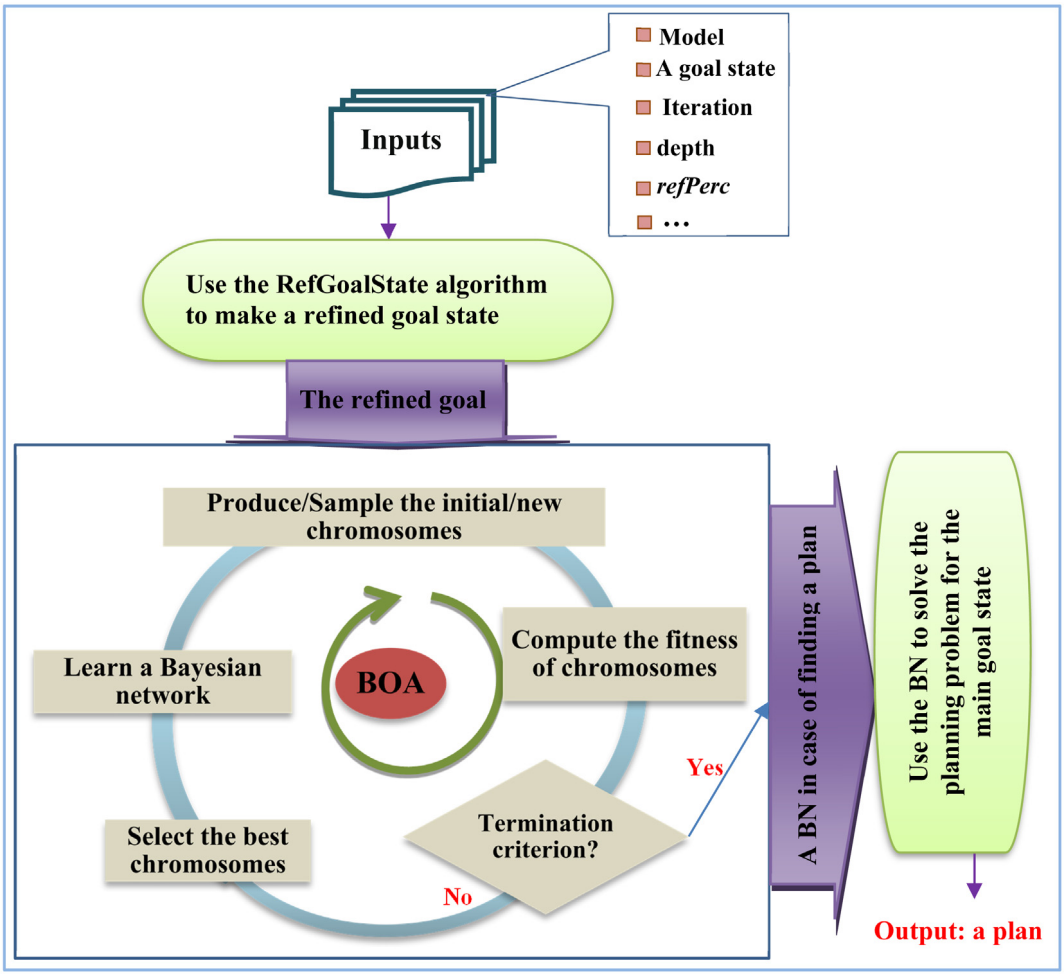


Fig. 2. The architecture of the plnBOA approach.

Algorithm 1 The RefGoalState algorithm

Input: *M*: a model of the given planning problem, *mg*: a main goal state, *refPerc*.
Output: a refined goal state.
1: ArrayList<Group> *allGroups* = new ArrayList< Group > ();
2: *allGroups* = divMGoalToGroups (*mg*, *M*);
3: **for** k=1 to *allGroups*.size() **do**
4: Group *group* = *allGroups*.get (k);
5: *numDel* = *group*.size()**refPerc;*
6: delFromGroup (*M*, *mg*, *group*, *numDel*);
7: **end for**
8: **return** *mg*;

the effect of *depth* on the accuracy of all approaches. Moreover, in Section 5.5, we present another experiment to specify the effect of *refPerc* on the accuracy of plnBOA. Finally, Section 5.6 discusses the advantages, limitations, and convergence of plnBOA. Also, the Mann–Whitney U test is employed to confirm that the results of plnBOA are meaningfully different from the others.

5.1. The required parameters

The planning domains used in the paper are those that used in the proposed approaches BOAcl2 (Pira et al., 2019), ACO (Rafe et al., 2019), LBN and LDM (Pira et al., 2018), PSO (Rafe et al., 2015), and GA (Yousefian et al., 2014). Moreover, all results have been generated on a system with an Intel CORE i5 processor and 3 GB of memory. Hence, suitable values for required parameters of each approach have been taken from the related paper. Table 1 displays these parameters along with their appropriate values.

The plnBOA approach has two important parameters: *depth* and *refPerc*. *depth* specifies the maximum depth of exploration of the search space to find a goal state. To find suitable values for this parameter, we have employed the "*Random Exploration*" strategy in GROOVE. In this strategy, one output transition is selected randomly through the output transitions of the current state. This process is repeated until a goal state (successful run) or a state without any transition (unsuccessful

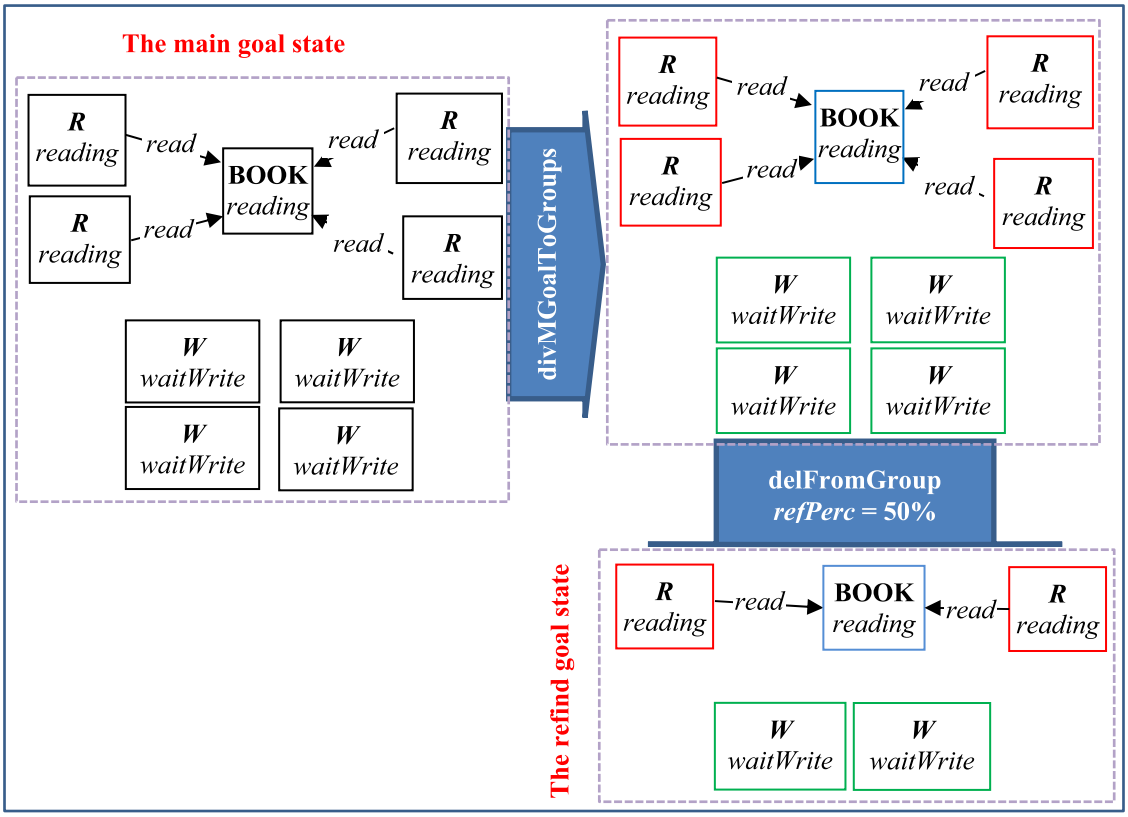


Fig. 3. An example of operation of the RefGoalState algorithm.

Algorithm 2 Solving the Planning Problem for the Main Goal

Input: *bn*: a BN, *M*: a model of the given planning problem, *depth:* a user-specified number, *mg*: a main goal state, *iterations*: a maximum number of iterations.
Output: A plan.

```
iter = 1;
while iter <= iterations do
    Path path =null; path.Add (the initial state of M);
    State LS = the initial state of M; depth = 0;
    Rule curRule = null, prevRule = null;
    while depth <= depth do
        allowedRules = Find all applicable rules over LS;
        if prevRule is null then
            curRule = select a rule r from allowedRules using the roulette-wheel method
                      based on the probabilities of bn.p(X0= r);
        else
            curRule = choose a rule r from allowedRules using the roulette-wheel method
                      based on the probabilities of bn.p(X1= r |X0= prevRule);
        end if
        Apply curRule over LS and update it;
        path.Add (curRule); path.Add (LS);
        if LS equals mg then
            return "The planning problem is solved successfully and a plan is:" + path;
        end if
        prevRule = curRule; depth ++;
    end while
    iter ++;
end while
return "The planning problem is not solved successfully";
```

Table 1
The parameter setting for the approaches.

Iterations			100	
BOAcl2			ACO	
Learning rate	0.4		*Colony size*	50
Replacement rate	0.6		*Ant path length* (λ_{ant})	50
			Stages	15
LBN and LDM				
NumPromis			3	
Minst	Planning domain	Value	Planning domain	Value
	Dining philosophers	0.6	*Readers–writers, shopping, electronic control units*	0.1

PSO		GA	
C1	2	*Crossover rate*	0.6
C2	2	*Mutation rate*	0.3
W	0.8	*Position of crossover*	Middle of the chromosomes

run) is found. The strategy has been executed 30 times for any model of domain problems. By considering the successful runs, the minimum value of the length of explored paths can be an appropriate value for the *depth* parameter. After finding a suitable value for *depth,* we use this value to determine the suitable value for the *refPerc* parameter. The *refPerc* parameter determines the removal percentage of the main goal by the refinement technique, and so its values can modify the size of the refined goal. If the value of *refPerc* is very large, the generated refined goal will be so small that the BOAcl2 may not work successfully. Conversely, the small values for *refPerc* cause that the size of the refined goal is so large that plnBOA may not solve the planning problem for the refined goal. Due to this subject, we have executed plnBOA for five different values of *refPerc*. The value caused the plnBOA to have the best performance (plans with the least length and generated in the shortest time) is selected as an optimum value for *refPerc*. In *k*-BFWS, the propositional atoms for each problem domain is defined due to the specified goal, and the number of these atoms is considered as *k*. Table 2 shows the parameters of plnBOA and *k*-BFWS along with their appropriate values.

5.2. Planning domains

This section gives a brief introduction to the problem domains like as dining philosophers (Schmidt, 2004), readers–writers (Pira et al., 2018), shopping (Hausmann, 2005), and electronic control units (Ziegert, 2014).

Dining philosophers problem

This problem models the situation in which some philosophers have sat around a table with a fork between both of them. The philosophers can be in one of the *thinking*, *hungry, hasLeft*, and *eating* modes, which firstly are in the *thinking mode.* Each *thinking*-mode philosopher can go to the *hungry* mode, and if its left fork is also free, it can go to the *hasLeft* mode. If the right fork of a *hasLeft*-mode philosopher is free, the philosopher gets the fork and goes to the *eating* mode. After a while, a philosopher with the *eating* mode gives up the left and right forks respectively and goes to the *thinking* mode. This process iterates until a deadlock state occurs. A deadlock state happens when all philosophers have picked up their left fork and wait for their right fork. In this problem, a state property "*there exists a state in which all philosophers are in the hasLeft mode*" is considered as a goal state.

Readers–writers problem

This problem has previously been described in Section 3.2. In this problem, we consider a state property "*there exists a state in which all readers and writers have finished their processing*" as a goal state.

Shopping problem

By this problem, the process of shopping by customers in a store is described. In different models of this problem, the process of shopping finishes if all customers end their shopping and stop. In this problem, a state property "*there exists a state in which one customer has purchased half of all existing goods*" is considered as a goal state.

Electronic control units

This problem, which is relating to the reconfiguration of Electronic Control Units (ECUs) in automotive systems, describes the runtime reconfiguration of software components running on a Runtime Environment (RTE) middleware. This middleware connects software components to a basic software that controls hardware. The aim of this problem is to search some hardware failures such as shutting down an ECU or rebooting a RTE after an upgrading software. In this problem, a state property "*the half of existing ECUs should be shut down and all software components should be instantiated* " is considered as a goal state.

5.3. Experiment 1: the performance comparison of plnBOA with others in different terms

This section compares plnBOA with the other approaches in terms of running time, the number of explored states, and the length of generated plans. For this purpose, we run all approaches 30 times to different sizes of aforementioned problem domains. Before reporting the running results, a general preview is presented as the following. According to Pira et al. (2019), BOAcl2 outperforms LBN, LDM, GA, PSO, and ACO in all problem domains. Since planBOA calls BOAcl2 to solve the refined planning problem and then uses the knowledge learned by BOAcl2 to solve the main planning problem, plnBOA can solve the large and complex planning problems which even BOAcl2 and others cannot solve them. *k*-BFWS checks the predefined propositional atoms for each (next) state in the next level of search space to compute its novelty and prunes the extra states. In the problem domains such as dining philosophers, readers–writers, and electronic control units, there is a lower number of next states compared to the shopping problem. The reason can be related to the low number of rules of these problem domains or the structure of the host graphs. Nevertheless, *k*-BFWS has a high performance in comparison with GA, PSO, and ACO. Whereas, it cannot find any plans in the shopping problem. Similar to *k*-BFWS, LBN and LDM also have a bad performance in problem domains like shopping because these domains have a large number of transformation rules and extracting the required knowledge is time-consuming. Usually, most running time is consumed for exploring and evaluating the states. In all approaches except *k*-BFWS, only the last state of current solutions (i.e. particles in PSO, ant solutions in ACO, or

Table 2
The parameter setting for the plnBOA and *k*-BFWS approaches.

Planning domain	Size	*RefPerc*	*depth*	*k* (the number of propositional atoms)
Dining philosophers	20 philosophers	90%	100	20
	40 philosophers	92%	120	40
	60 philosophers	94%	140	60
	80 philosophers	96%	180	80
Readers–writers	4 readers and 4 writers	50%	50	8
	5 readers and 5 writers	60%	70	10
	6 readers and 6 writers	70%	90	12
Shopping	6 customers and 16 goods	60%	50	17
	8 customers and 18 goods	70%	50	19
	10 customers and 20 goods	80%	50	21
Electronic control units	4 ECUs	50%	30	8
	6 ECUs	65%	50	12
	8 ECUs	75%	90	16

chromosomes in GA and BOA) is evaluated. Hence, in these approaches, the average running time is approximately proportional to the average number of explored states. Whereas, in *k*-BFWS, since all explored states should be evaluated, the proportion of the average running time and the average number of explored states can be different from this proportion in other approaches.

The charts in Figs. 4–7 show the average running time of the runs. The term "Not found" in some charts expresses that the corresponding approach either has used up all available memory or has reached to the maximum number of iterations without finding a plan. Of course, if an approach can find a plan in a very high time (in comparison with others), this term is used again. As seen in the charts, the average running time of plnBOA is less than the others in most problem domains. BOAcl2 also has the less average running time in comparison with others. Whereas, LBN, LDM, and *k*-BFWS have the high average running time in the shopping problem (even in small models of this problem).

The number of explored states is another important criterion to evaluate and compare the effectiveness of approaches. Whatever the number of explored states by an approach is less, the consumed memory is less too. As a result, this approach can find a plan in large models without using up all available memory. In charts of Figs. 8–11, we display the average number of explored states for the aforementioned runs of all approaches. These charts confirm that plnBOA can find plans with the lower number of explored states in most problem domains. Although, GA, PSO and ACO explore the maximum number of states, they cannot find any plans in very large models of some problem domains.

The length of generated plans is another important criterion to evaluate and compare the effectiveness of approaches. It is possible that the generated plans represent some undesirable behaviors of domains and so they can be used to analyze the domains. As a result, generating shorter plans is a major goal in solving the planning problems. The charts in Figs. 12–15 display the average length of generated plans for the aforementioned runs of all approaches. According to these charts, plnBOA produces the shorter plans in most problem domains.

5.4. Experiment 2: comparison of the effect of depth on the accuracy of all approaches

In here, we define the accuracy of an approach as the ratio of the number of successful runs to the total runs. The *depth* parameter specifies that an approach can explore until how deep of the state space. In fact, *depth* controls the accuracy and the length of generated plans. If the value of this parameter is very large, the approach finds a goal state successfully and so the accuracy will increase. Moreover, the length of generated plans will be long, and this violates the primary goal of planning that its goal is to find plans with the least actions. Conversely, the small values for *depth* cause that the approach may not find any plans, and this decreases its accuracy. In this section, we compare the effect of *depth* on the accuracy of all approaches. To do this, we consider the readers–writers problem with 6 readers and 6 writers. The chart in Fig. 16 shows the accuracy of all approaches for 20 independent runs. According to the chart, when the value of *depth* is very small, the accuracy of all approaches (even plnBOA) is low. By increasing the value of *depth*, the accuracy also improves. After a threshold (90 for the readers–writers problem), the accuracy of all approaches will be 1, i.e. they always work successfully.

5.5. Experiment 3: the effect of refperc on the accuracy of plnBOA

The *refPerc* parameter specifies the removal percentage of the main goal by the refinement technique, and so its values can change the refined goal. If the value of *refPerc* is very large, i.e. a large number of nodes of the main goal state are removed, the refined goal will be so small that the planning problem with this goal may not be solved. As a result, the accuracy will be decreased. Conversely, the small values for *refPerc* cause that the refined goal is relatively large and BOAcl2 may not solve the planning problem for the refined goal. Hence, similar to large values, small values also decrease the accuracy. To specify the effect of *refPerc* on the accuracy of plnBOA, we consider the readers–writers problem with 4 readers and 4 writers, the shopping problem with 6 customers and 16 goods and the electronic control units problem with 4 ECUs. The chart in Fig. 17 displays the effect of *refPerc* on the accuracy of plnBOA for 20 independent runs. According to the chart, if the value of *refPerc* is very large, the accuracy is very low. Also, whatever the value of *refPerc* is small (of course, until a threshold), the accuracy is high.

5.6. Discussion

The advantages of the plnBOA approach in comparison with others can be summarized as follows: (1) the reported results in Section 5.3 show that plnBOA has a high speed. Its main reason is that plnBOA not only solve the refined planning problem; it also extracts the knowledge from this process and then uses the knowledge to solve the main planning problem. Whereas, other approaches try to solve the main planning problem and this causes they solve the problem with low speed. To confirm that the average running time of plnBOA is significantly different from the other approaches, we use the Mann–Whitney U test. This test is a non-parametric statistical hypothesis employed when two independent samples are compared (Arcuri and Briand, 2011). This test can be performed using the SPSS toolbox. In this toolbox, if the output decision criterion (sig.) is less than 0.05, we can conclude that there exists a significant difference between two sets of data. In here, this test is performed on the results of plnBOA against BOAcl2, ACO, LBN, LDM, PSO, GA, and *k*-BFWS approaches. As seen in Table 3, the decision criterion (sig.) in all cases is less than 0.05. It means that the average running time of plnBOA is significantly different from the others. (2) According to the charts of Figs. 8–11,

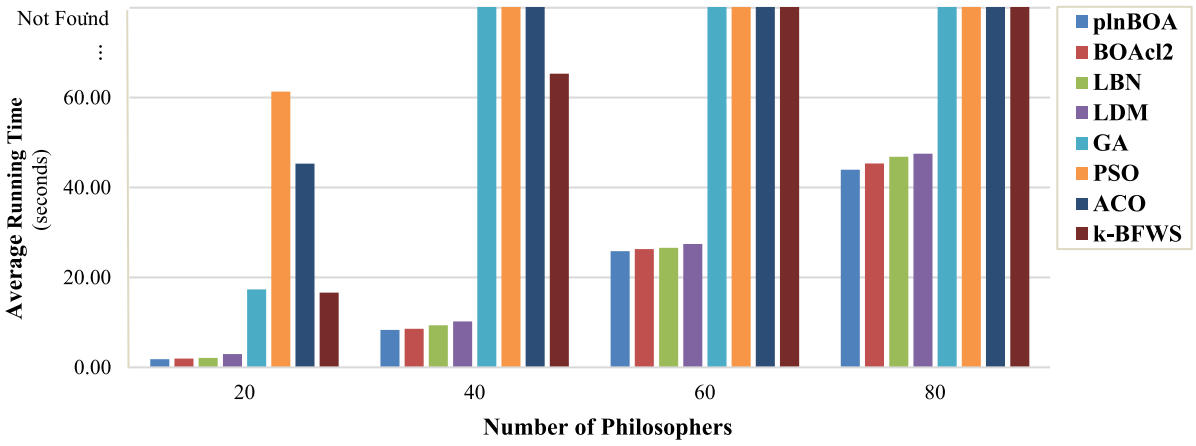


Fig. 4. The comparison of the average running time of all approaches for the dining philosophers problem.

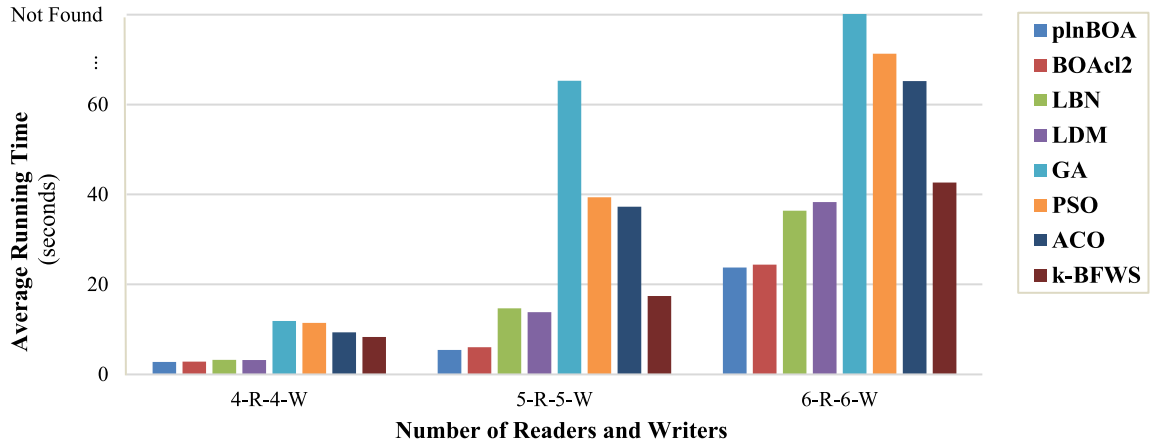


Fig. 5. The comparison of the average running time of all approaches for the readers–writers problem.

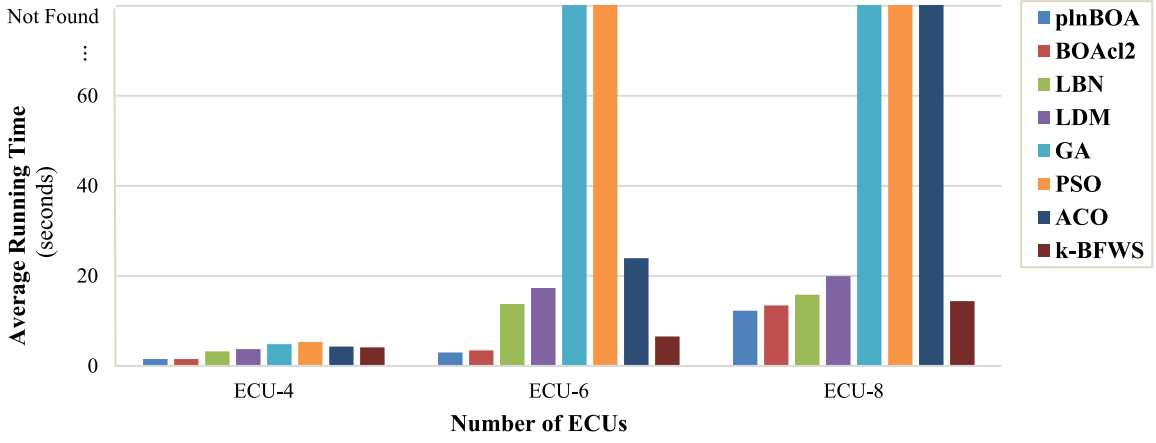


Fig. 6. The comparison of the average running time of all approaches for the electronic control units problem.

plnBOA explores a lower number of states in comparison with others. (3) The length of generated plans by plnBOA is shorter in most problem domains. The charts in Figs. 12–15 confirm this claim. (4) The accuracy is a prominent goal for the proposed approaches. To compare the accuracy of plnBOA with others, we consider the dining philosopher's problem with 10 philosophers. Moreover, the value of *depth* is set to 20, the minimum possible value in this model. As seen in the chart of Fig. 18, plnBOA, BOAcl2, LBN, LDM, and *k*-BFWS are more accurate than the others.

The plnBOA approach also has some limitations. The spent time for generating the refined goal state can be a bottleneck for this approach, particularly when the given goal state is very complex. Another limitation of plnBOA, relating to BOAcl2, is that the learning of BNs is a time-consuming work especially in problem domains with a large number of transformation rules.

To examine the convergence of plnBOA, we should analyze the convergence of two important sub-algorithms: BOAcl2 for solving the refined planning problem and Algorithm 2 for solving the main planning problem. According to He and Yu (2001), the most important

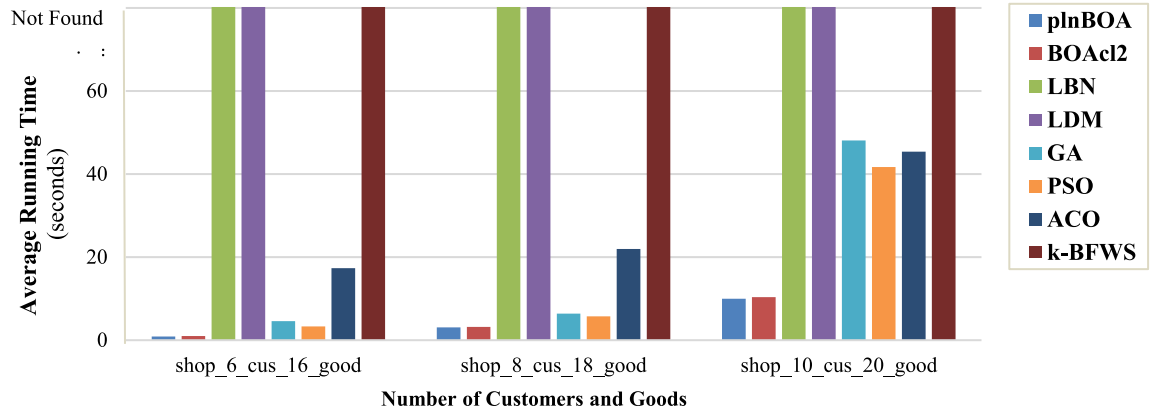


Fig. 7. The comparison of the average running time of all approaches for the shopping problem.

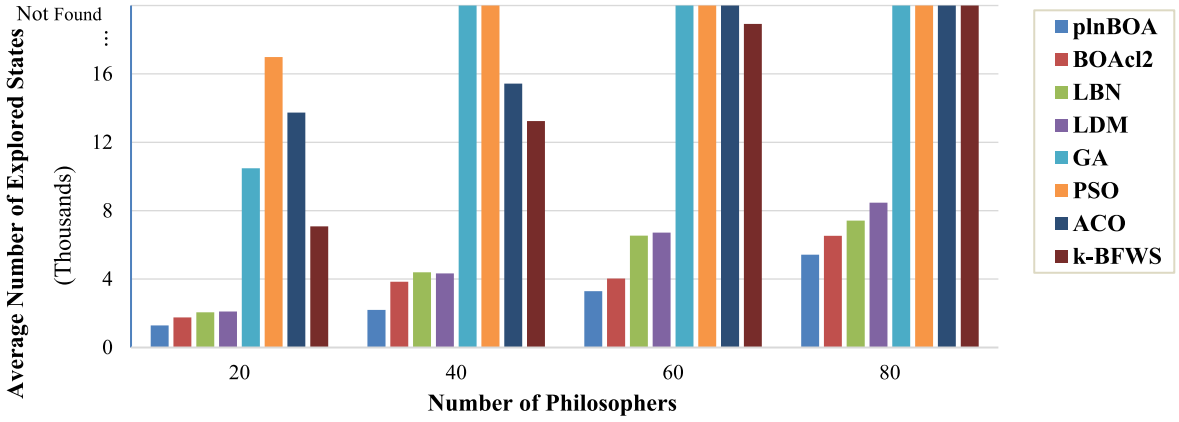


Fig. 8. The comparison of the average number of explored states by all approaches for the dining philosophers problem.

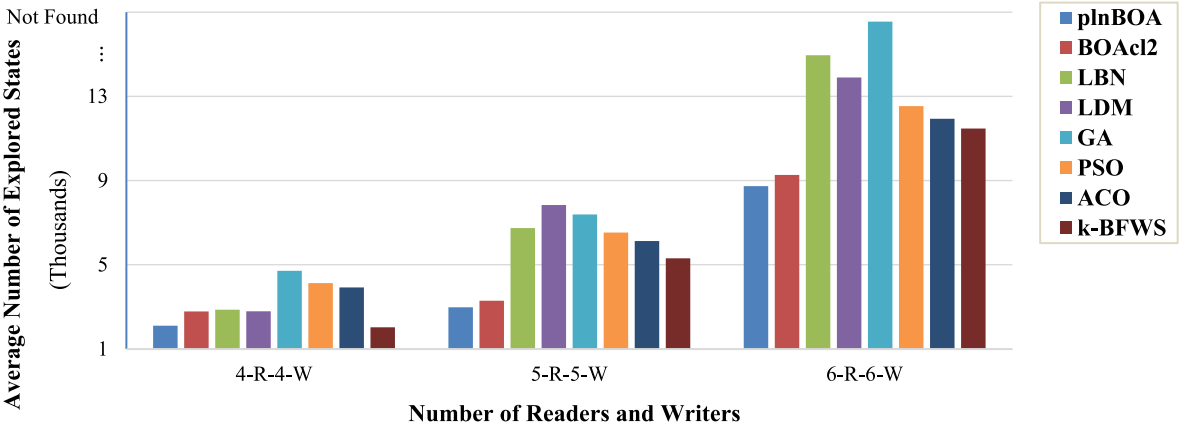


Fig. 9. The comparison of the average number of explored states by all approaches for the readers–writers problem.

Table 3
The results of the Mann–Whitney U test.

	plnBOA – BOAcl2	plnBOA- ACO	plnBOA - LBN	plnBOA - LDM	plnBOA –PSO	plnBOA –GA	plnBOA –*k*-BFWS
Z	−2.004	−2.938	−2.022	−2.112	−3.213	−3.143	−2.531
Asymp. Sig. (2-tailed)	0.039	0.027	0.034	0.036	0.021	0.019	0.029

condition for the convergence of evolutionary algorithms such as BOA is that the fitness values of generations must gradually tend to the optimum value. In Pira et al. (2019), the authors have verified the correctness of this condition. When BOAcl2 reaches to an optimum solution, the last BN has the exact knowledge about dependencies between applied rules. In addition, since the main goal state is a large-scale instance of the refined goal state, Algorithm 2 can solve the main planning problem using the information of the BN. Hence, because of the convergence of BOAcl2 and Algorithm 2, the convergence of plnBOA can be concluded.

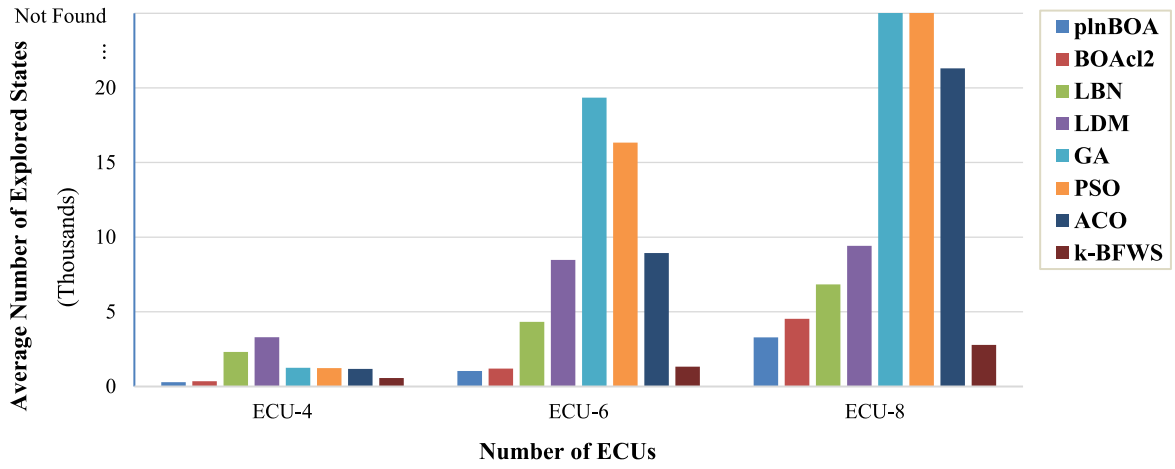


Fig. 10. The comparison of the average number of explored states by all approaches for the electronic control units problem.

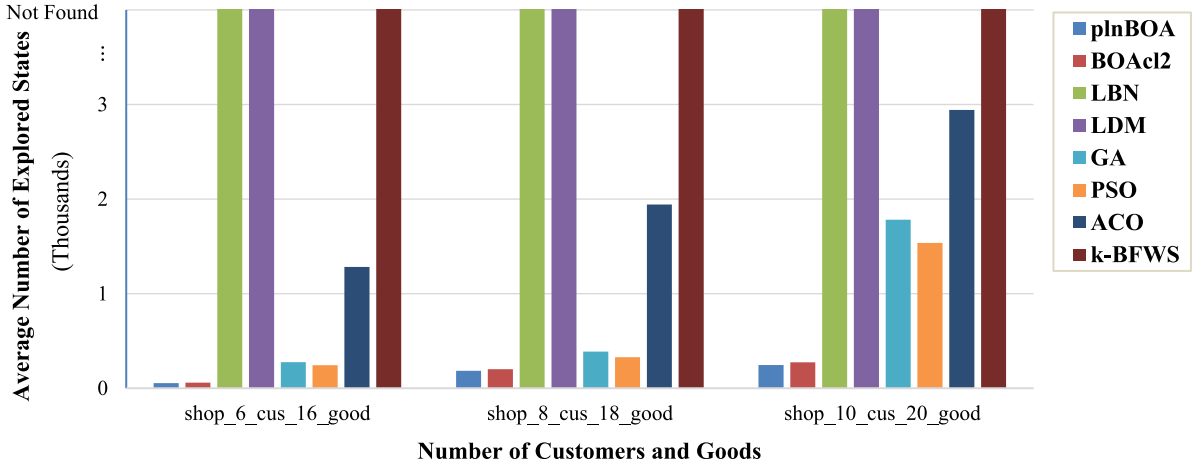


Fig. 11. The comparison of the average number of explored states by all approaches for the shopping problem.

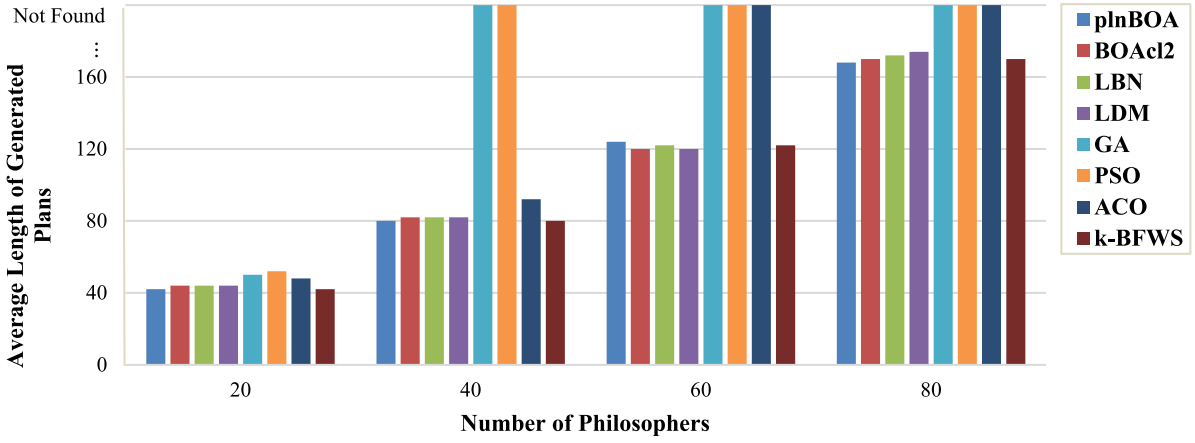


Fig. 12. The comparison of the average length of generated plans by all approaches for the dining philosophers problem.

6. Conclusion and future works

In planning problems specified formally through graph transformation system (GTS), there are dependencies between applied rules (actions) in the search space. Based on this fact, in this paper, we presented a novel approach to solve planning problems in such systems. The proposed approach, also called plnBOA, refines the given goal state in order to create a small goal, then it uses the Bayesian Optimization Algorithm (BOA) to solve the planning problem for this small goal. If BOA can solve such planning problem, the last BN learned in BOA will have the exact knowledge about dependencies between applied rules. So, plnBOA uses the last BN to solve the planning problem for the main goal state. The proposed approach is implemented in GROOVE as a planning system. To evaluate and compare the effectiveness of plnBOA,

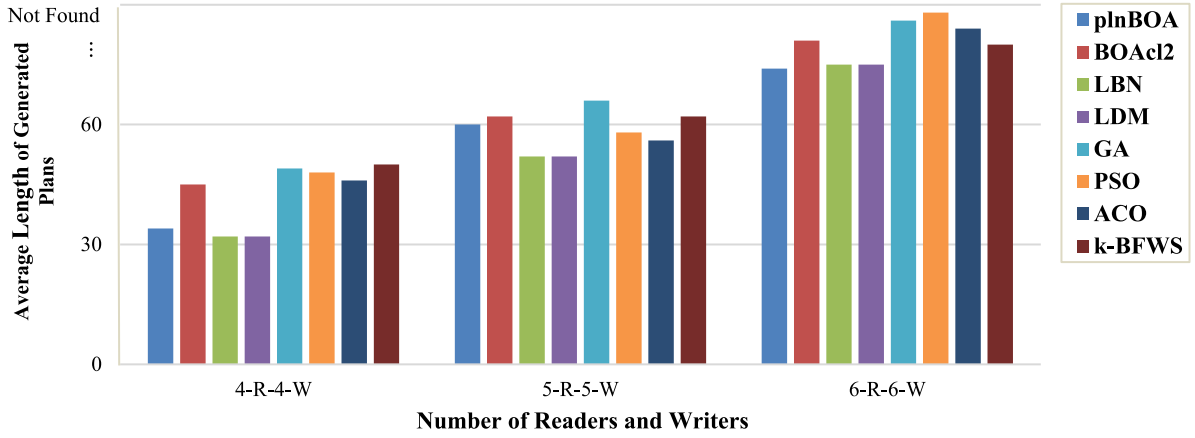


Fig. 13. The comparison of the average length of generated plans by all approaches for the readers–writers problem.

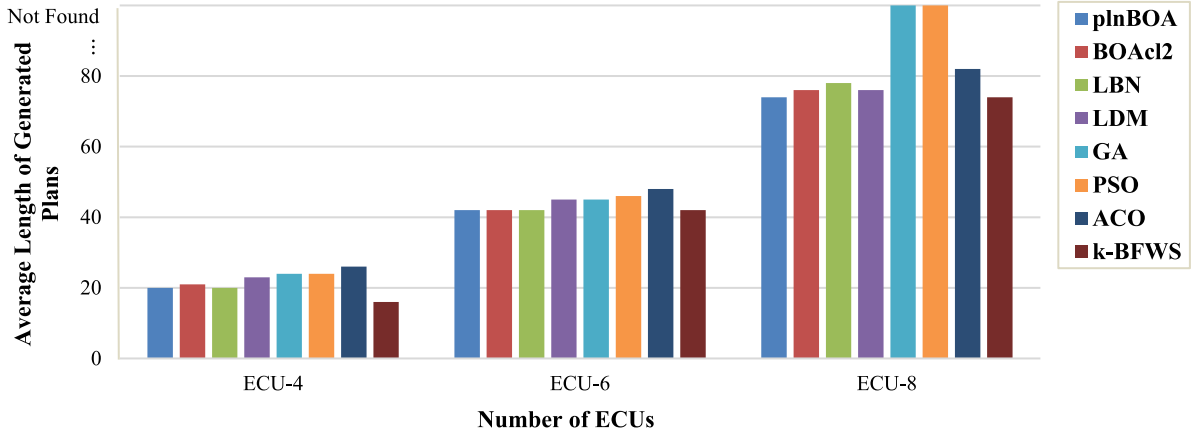


Fig. 14. The comparison of the average length of generated plans by all approaches for the electronic control units problem.

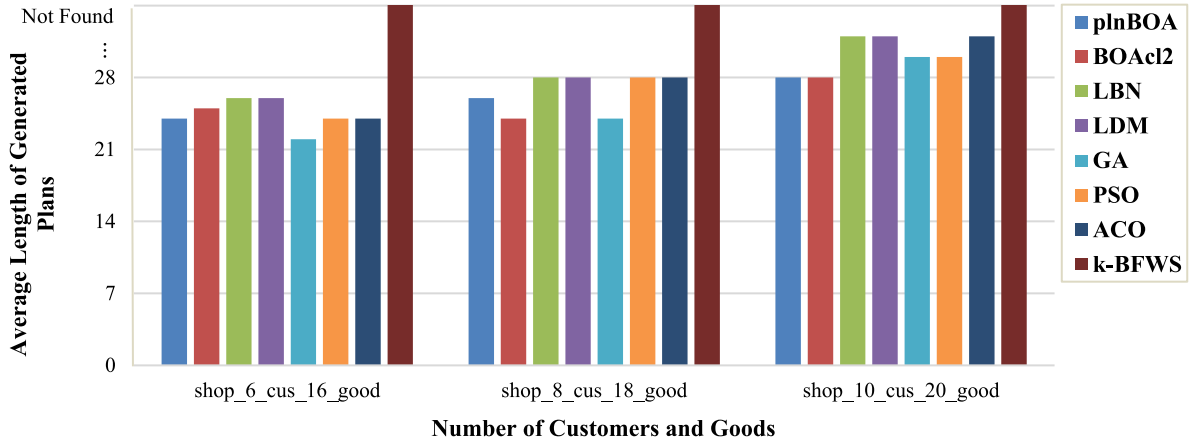


Fig. 15. The comparison of the average length of generated plans by all approaches for the shopping problem.

we use the state-of-the-art approaches (or planners) BOAcl2, LBN, LDM, GA, PSO, ACO, and *k*-BFWS. It should be noted that, for a fair comparison, we have implemented the *k*-BFWS planner in GROOVE. The implementation results of the approaches on four well-known problem domains confirm that plnBOA outperforms the others.

The plnBOA approach has many advantages such as faster execution speed, shorter length of generated plans, fewer number of explored states, and higher accuracy in comparison with the others. plnBOA also has some limitations such as (1) it spends much time to generate the refined goal state, especially when the given goal state is very complex. (2) The learning of BNs (of course through BOAcl2) is time-consuming especially in problem domains with a large number of rules. Since two important sub-algorithms of plnBOA i.e. BOAcl2 and Algorithm 2 have the convergence to an optimum solution, the convergence of plnBOA can be concluded. In this paper, we used BOA to solve planning problems. Using other meta-heuristic and evolutionary algorithms can be considered as a future work.

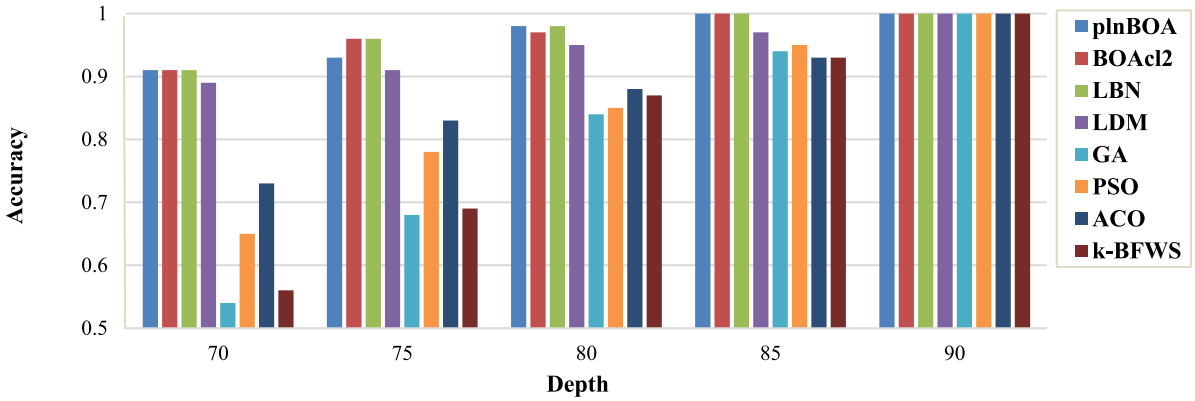


Fig. 16. Comparison of the effect of *depth* on the accuracy of all approaches for the readers–writers problem with 6 readers and 6 writers.

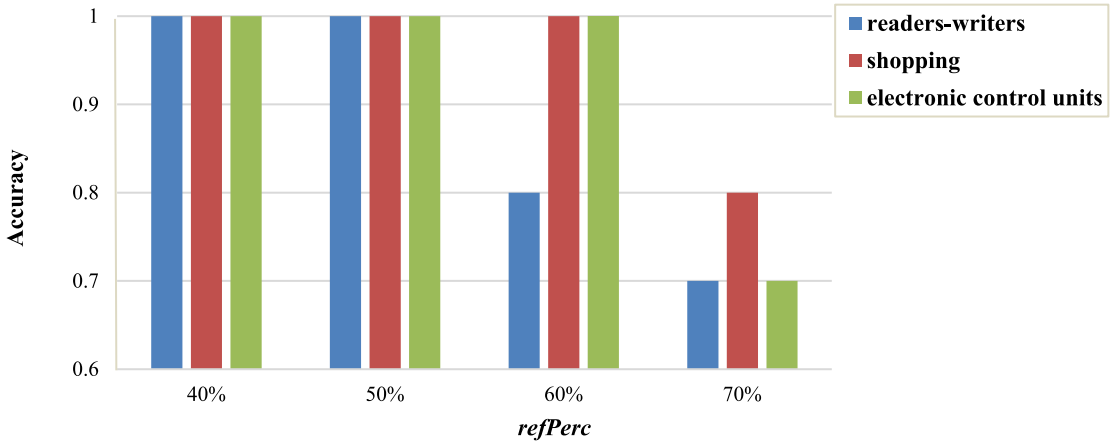


Fig. 17. Comparison of the effect of *refPerc* on the accuracy of plnBOA.

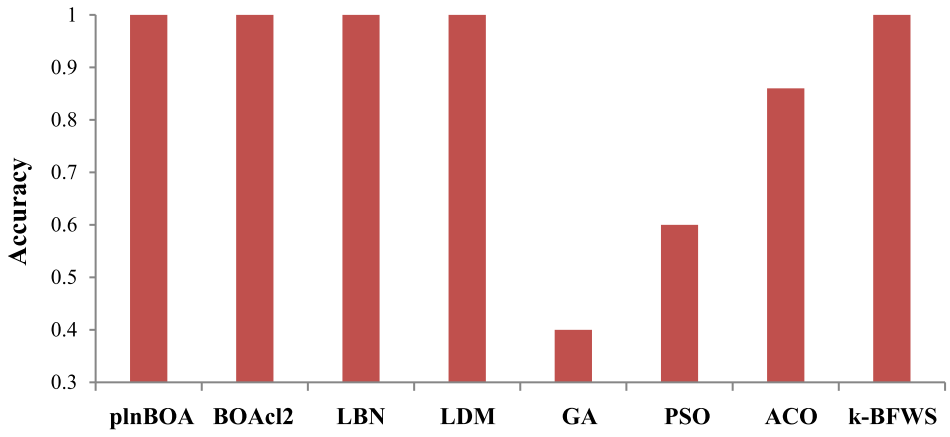


Fig. 18. Comparison of the accuracy of the approaches to solve the planning problem in the dining philosopher's problem with 10 philosophers, *depth limit* = 20.

CRediT authorship contribution statement

Einollah Pira: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing - original draft, Writing - review & editing, Visualization, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Aeronautiques, C., Howe, A., Knoblock, C., McDermott, I.D., Ram, A., Veloso, M., Weld, D., Sri, D.W., Barrett, A., Christianson, D., 1998. PDDL | The Planning Domain Definition Language.

Arcuri, A., Briand, L., 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: 2011 33rd International Conference on Software Engineering (ICSE). IEEE, pp. 1–10.

Edelkamp, S., Jabbar, S., Lafuente, A.L., 2006. Heuristic search for the analysis of graph transition systems. In: International Conference on Graph Transformation. Springer, pp. 414–429.

Edelkamp, S., Rensink, A., 2007. Graph transformation and ai planning. In: Knowl. Eng. Compet. ICKEPS R. I. USA.

Hausmann, J.H., 2005. Dynamic META modeling: a semantics description technique for visual modeling languages.
He, J., Yu, X., 2001. Conditions for the convergence of evolutionary algorithms. J. Syst. Archit. 47, 601–612.
Hoffmann, J., Nebel, B., 2001. The FF planning system: Fast plan generation through heuristic search. J. Artificial Intelligence Res. 14, 253–302.
Kastenberg, H., Rensink, A., 2006. Model checking dynamic states in GROOVE. In: International SPIN Workshop on Model Checking of Software. Springer, pp. 299–305.
Kautz, H., Selman, B., 1998. BLACKBOX: A new approach to the application of theorem proving to problem solving. In: AIPS98 Workshop on Planning as Combinatorial Search. pp. 58–60.
Kortik, S., Saranli, U., 2017. Lingraph: a graph-based automated planner for concurrent task planning based on linear logic. Appl. Intell. 47, 914–934.
Lipovetzky, N., Geffner, H., 2017. A polynomial planning algorithm that beats LAMA and FF. In: Twenty-Seventh International Conference on Automated Planning and Scheduling.
Pelikan, M., Goldberg, D.E., Tsutsui, S., 2003. Hierarchical Bayesian optimization algorithm: toward a new generation of evolutionary algorithms. In: SICE 2003 Annual Conference (IEEE Cat. No. 03TH8734). IEEE, pp. 2738–2743.
Pira, E., Rafe, V., Nikanjam, A., 2016. EMCDM: Efficient model checking by data mining for verification of complex software systems specified through architectural styles. Appl. Soft Comput. 49, 1185–1201.
Pira, E., Rafe, V., Nikanjam, A., 2018. Searching for violation of safety and liveness properties using knowledge discovery in complex systems specified through graph transformations. Inf. Softw. Technol. 97, 110–134.
Pira, E., Rafe, V., Nikanjam, A., 2019. Using evolutionary algorithms for reachability analysis of complex software systems specified through graph transformation. Reliab. Eng. Syst. Saf. 106577.
Rafe, V., Darghayedi, M., Pira, E., 2019. MS-ACO: a multi-stage ant colony optimization to refute complex software systems specified through graph transformation. Soft Comput. 23, 4531–4556.
Rafe, V., Moradi, M., Yousefian, R., Nikanjam, A., 2015. A meta-heuristic solution for automated refutation of complex software systems specified through graph transformations. Appl. Soft Comput. 33, 136–149.
Richter, S., Westphal, M., 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. J. Artificial Intelligence Res. 39, 127–177.
Röhs, M., Wehrheim, H., 2010. Sichere konfigurationsplanung selbst-adaptierender systeme durch model checking. Entwurf Mechatronischer Syst. 272, 253–265.
Rozenberg, G., 1997. Handbook of Graph Grammars and Comp.. World scientific.
Russell, S.J., Norvig, P., 2016. Artificial Intelligence: A Modern Approach. Pearson Education Limited, Malaysia.
Sagarna, R., Lozano, J.A., 2006. Scatter search in software testing, comparison and collaboration with estimation of distribution algorithms. European J. Oper. Res. 169, 392–412.
Santana, R., Larrañaga, P., Lozano, J.A., 2008. Protein folding in simplified models with estimation of distribution algorithms. IEEE Trans. Evol. Comput. 12, 418–438.
Schmidt, Á., 2004. Model Checking of Visual Modeling Languages. Bp. Univ. Technol. Hung.
Silva, F., Castilho, M.A., Künzle, L.A., 2000. Petriplan: a new algorithm for plan generation (preliminary report). In: Advances in Artificial Intelligence. Springer, pp. 86–95.
Snippe, E., 2011. Using heuristic search to solve planning problems in GROOVE. In: 14th Twente Student Conference on IT. University of Twente, Available at Fmt. Cs. Utwente. Nl/Education/Bachelor/73.
Tsutsui, S., Wilson, G., 2004. Solving capacitated vehicle routing problems using edge histogram based sampling algorithms. In: Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753). IEEE, pp. 1150–1157.
Yousefian, R., Rafe, V., Rahmani, M., 2014. A heuristic solution for model checking graph transformation systems. Appl. Soft Comput. 24, 169–180.
Yuan, B., Orlowska, M., Sadiq, S., 2007. Finding the optimal path in 3D spaces using EDAs–the wireless sensor networks scenario. In: International Conference on Adaptive and Natural Computing Algorithms. Springer, pp. 536–545.
Zhang, H., Hui, Q., 2016. Parallel multiagent coordination optimization algorithm: implementation, evaluation, and applications. IEEE Trans. Autom. Sci. Eng. 14, 984–995.
Zhang, H., Hui, Q., 2017. Cooperative bat searching algorithm: A combined perspective from multiagent coordination and swarm intelligence. In: 2017 13th IEEE Conference on Automation Science and Engineering (CASE). IEEE, pp. 1362–1367.
Ziegert, S., 2014. Graph transformation planning via abstraction. ArXiv Prepr. arXiv: 1407.7933.