Front. Comput. Sci., 2023, 17(4): 174326
https://doi.org/10.1007/s11704-022-2023-7

RESEARCH ARTICLE

An improved master-apprentice evolutionary algorithm for minimum independent dominating set problem

Shiwei PAN[1,2], Yiming MA[1,2], Yiyuan WANG[1,2], Zhiguo ZHOU (✉)[1,2], Jinchao JI (✉)[1,2], Minghao YIN (✉)[1,2], Shuli HU[1,2]

1 School of Computer Science and Information Technology, Northeast Normal University, Changchun 130117, China
2 Key Laboratory of Applied Statistics of MOE, Northeast Normal University, Changchun 130117, China

Abstract The minimum independent dominance set (MIDS) problem is an important version of the dominating set with some other applications. In this work, we present an improved master-apprentice evolutionary algorithm for solving the MIDS problem based on a path-breaking strategy called MAE-PB. The proposed MAE-PB algorithm combines a construction function for the initial solution generation and candidate solution restarting. It is a multiple neighborhood-based local search algorithm that improves the quality of the solution using a path-breaking strategy for solution recombination based on master and apprentice solutions and a perturbation strategy for disturbing the solution when the algorithm cannot improve the solution quality within a certain number of steps. We show the competitiveness of the MAE-PB algorithm by presenting the computational results on classical benchmarks from the literature and a suite of massive graphs from real-world applications. The results show that the MAE-PB algorithm achieves high performance. In particular, for the classical benchmarks, the MAE-PB algorithm obtains the best-known results for seven instances, whereas for several massive graphs, it improves the best-known results for 62 instances. We investigate the proposed key ingredients to determine their impact on the performance of the proposed algorithm.

Keywords evolutionary algorithm, combinatorial optimization, minimum independent dominating set, local search, master apprentice, path breaking

1 Introduction

Given an undirected graph $G = (V, E)$, a dominating set (DS) is a subset D of V such that each vertex not in D is adjacent to at least one vertex of D and an independent set (IS) is a subset I of V, where any two vertices in I are not adjacent. An independent dominating set (IDS) refers to a subset of V, which is both an IS and a DS. The purpose of the minimum independent dominating set (MIDS) problem is to find an independent dominating set with the minimum size in a given graph.

The models of IDs and DSs have been widely used in many real-world fields. In the following, we briefly introduce several applications related to these problems. In terms of DS problems, they have been applied in various fields, such as wireless communication [1], metro networks [2], gateway placement [3], and biological networks [4]. The DS model has been applied to extract proteins that control protein-protein interaction networks and to reveal the correlation between structural analysis and biological functions [5]. The IS problem has many important applications, including code theory, economics, and information retrieval [6,7]. Several methods of graph theory can be used to express the coding problem, one of which is to find the maximum IS [8].

Combining the respective properties of the independent and dominating sets, the MIDS problem has been widely used in different real-world domains. For example, wireless sensor and actor networks (WSANs) usually need to provide services in each part of the deployment area especially coverage services which are important goals in many WSANs applications. High-quality coverage should minimize the overlap between the action ranges of actors and include all sensors deployed in the monitoring area. To achieve good coverage, researchers usually establish a clustered WSANs architecture where each cluster head takes certain actions based on the data received from the sensors in the cluster [9]. To achieve good distribution of actors in WSANs (for full coverage,) researchers usually model this problem into an independent dominating set and place the actors next to the location of the nodes in the network [10]. Because the price of the actors is often very expensive, our goal is to find the minimum number of actors in the network to achieve full coverage, that is, the MIDS problem. In addition to the above introduction of applications of the MIDS problem, many studies have been conducted on wireless network clustering algorithms [11,12], which shows that the MIDS model can be used for the initial clustering scheme of wireless networks [13–15].

In the following, we will introduce the related works of MIDS and propose our main contributions for solving MIDS.

Received January 13, 2022; accepted June 27, 2022

E-mail: zhouzg281@nenu.edu.cn; jijc100@nenu.edu.cn; ymh@nenu.edu.cn

1.1 Related works

It is well-known that the MIDS problem has been proven to be an NP-hard problem [16]. This means that there is no constant $\varepsilon > 0$, for which the MIDS problem can be approximated within a factor of $|V|^{1-\varepsilon}$ polynomial time unless P = NP, where $|V|$ is the number of vertices. Owing to the wide applications of the MIDS problem, many researchers have devoted themselves to designing MIDS algorithms that can mainly be divided into two types: exact algorithms and heuristic algorithms. In the past decades, there have been several exact algorithms for solving the MIDS problem. Gaspers and Liedloff designed a branch-and-reduce algorithm to solve the MIDS problem, which can obtain the result of $O(1.3575^{|V|})$ running time [17]. To solve the MIDS problem in sparse graphs, Liu and Song proposed exact algorithms with a time complexity of $O(1.3803^{|V|})$ and $O(1.5369^{|V|})$ [18]. Bourgeois et al. introduced a fast exact algorithm for solving the MIDS problem with a running time of $O(1.3351^{|V|})$ and a polynomial space [19]. Because of their NP-hard characteristics, although exact algorithms can guarantee the optimality of their solutions, they may not be able to solve large-scale instances.

To handle such large-scale instances, researchers have considered using heuristic algorithms to solve the MIDS problem. Although heuristic algorithms are not guaranteed to obtain the optimal solution, they can obtain a good solution within an acceptable time [20–25]. Normally, the effectiveness of heuristic algorithms depends on the properties of algorithms and the basic structure of problems to adapt to the corresponding specific implementations, which can search for promising search spaces and avoid falling into local optima. Recently, many heuristic algorithms for solving the MIDS problem have been proposed. For example, a greedy random adaptive search process based on a new heuristic path cost and tabu mechanism called GRASP+PC has been proposed to solve the MIDS problem [26]. The proposed GRASP+PC algorithm uses a new vertex attribute to define the scoring function, and during the search process, the algorithm exchanges a pair of vertices to further improve the solution quality according to the new scoring function. A tabu search-based memetic algorithm called MEMETIC was designed for the MIDS problem based on two ideas: the forgetting-based vertex weighting strategy and the repairing-based crossover strategy [27]. Specifically, the former idea exploited the possible spaces by making use of the current information of local search, while the latter idea not only inherited the results of parent solutions but also made up the infeasible solution. Haraguchi developed a metaheuristic framework that iteratively repeated the local search and the plateau search, where the local search used k-swap as the neighborhood operation and the plateau search examined solutions of the same size as the current solution that were obtainable by exchanging a solution vertex and a non-solution vertex [28]. Haraguchi proposed two algorithms, ILPS2 and ILPS3, according to different k values. Very recently, for solving the MIDS problem, Wang et al. used two-phase removal strategies, including the double-checked removal strategy and random diversity removing strategy, resulting in a two-phase removing algorithm called drMIDS [29]. The results show that drMIDS performs better than other MIDS heuristic algorithms on most classical benchmarks.

1.2 Our contributions

In this work, inspired by the idea of the master-apprentice evolutionary (MAE) algorithm proposed in [30], we design an improved algorithm for solving the MIDS problem. The traditional population-based evolutionary algorithm will always maintain a large number of populations, which leads to high resource consumption. Therefore, to avoid wasting computing resources, the MAE algorithm has been proposed. It utilizes an evolutionary mechanism based on two individuals, making the exploration space of solutions in this algorithm more diversified because it updates two individuals simultaneously.

Combining a master-apprentice evolutionary algorithm with the path-breaking strategy, a new algorithm called MAE-PB is proposed for solving the MIDS problem. The main contributions of this work can be summarized as follows:

- First, the proposed MAE-PB algorithm is the first adaptation of the general master-apprentice evolutionary algorithm tailored to the MIDS problem. The algorithm integrates a set of original features, including a construction function used to initialize and restart the master and apprentice solutions, and a multiple neighborhood-based local search function used to improve the master and apprentice solutions.
- Second, of particular interest is the ability of the proposed MAE-PB to explore different search spaces by using a perturbation method during the local search process and using path-breaking based on the definition of solution similarity during the solution recombination process. By allowing the search to oscillate as many areas as the algorithm can, the proposed MAE-PB promotes exploration of large search spaces based on master and apprentice solutions and helps to identify high-quality solutions.
- Third, we show the competitiveness of the MAE-PB algorithm by presenting computational results on classical benchmarks from the literature and several massive graphs from real-world applications. The experimental results demonstrate the high competitiveness of MAE-PB compared to the five state-of-the-art algorithms. In particular, MAE-PB updates 69 best-known results.

The reminder of the paper is organized as follows. Section 2 presents some basic definitions and a review of the master-apprentice evolutionary algorithm. In Section 3, we describe the proposed algorithm and its ingredients. In Section 4, we present computational studies and comparisons between the proposed algorithm and state-of-the-art algorithms. Finally, we draw conclusions and provide perspectives for future studies.

2 Background

2.1 Basic definitions and notations

For an undirected graph $G = (V, E)$, a vertex set is

$V = \{v_1, v_2, \ldots, v_n\}$ and an edge set $E = \{e_1, e_2, \ldots, e_m\}$. For each edge $e = (u, v)$, the vertices u and v are called the endpoints of edge e. For vertex v, the neighbors of v is denoted as $N(v) = \{u \in V | (v, u) \in E\}$. Further, we define the close neighborhood of vertex v as $N[v] = N(v) \cup \{v\}$. We use $dist(u, v)$ to denote the distance between u and v that is the number of edges from the shortest path of u to v. For a vertex v, $N_i(v) = \{u | dist(u, v) = i\}$ is defined as its ith level neighborhood, and $N_i[v] = N_i(v) \cup \{v\}$. We define $N^k(v) = \bigcup_{i=1}^{k} N_i(v)$ and $N^k[v] = N^k(v) \cup \{v\}$. Obviously, $N(v) = N_1(v)$ and $N[v] = N_1[v]$. For a vertex set $S \subseteq V$, $N[S] = \bigcup_{v \in S} N[v]$.

Given a graph $G = (V, E)$, a dominating set (DS) is a subset of $D \subseteq V$ such that each vertex in G belongs to D or is adjacent to a vertex in D. An independent set (IS) is a subset $I \subseteq V$ such that no two vertices are adjacent, i.e., $\forall v, u \in I$, $(v, u) \notin E$. The minimum independent dominating set (MIDS) problem requires a subset $S \subseteq V$ of the minimum cardinality such that S is both a dominating set and an independent set. For a vertex $v \in V$, the vertex v is dominated by a candidate solution S if $v \in N[S]$, and otherwise is non-dominated.

2.2 Review for master-apprentice evolutionary algorithm

The idea of the MAE algorithm originated from the social activities that apprentices learn skills from their masters. During one round, two apprentices evolve for a given number of generations. When the generation cycle ends, they become masters and one of them will replace the apprentice to continue the evolution, in order to preserve the good information from the previous generation. Ding et al. first proposed the MAE algorithm using only two individuals to solve the flexible job shop scheduling problem [30]. The inspiration of the MAE algorithm comes from HEAD [31], which is used to solve the k-coloring problem. The MAE algorithm maintains diversity by replacing the idea of one of the two individuals with random feasible solutions when the two individuals are close. Recently, many algorithms based on the MAS framework have been proposed. For example, Peng et al. designed a path-relinking algorithm framework based on an MAE framework. In addition, the algorithm used a solution-based tabu search and distance control relinking operator to solve the satellite broadcast scheduling problem [32]. For the production scheduling problem of assembly manufacturing systems with uncertain processing time and random machine failures, an improved MAE algorithm was proposed [33]. In the proposed algorithm, the extended sub-component adjacency matrix was used to deal with the sequence constraints of the operations. Owing to the similarity between the flow shop scheduling problem and the job shop scheduling problem, Sun et al. used the MAE algorithm to deal with the large-scale flow shop scheduling problem with uncertain time [34]. To solve the minimum weight vertex cover problem, a mixed tabu search evolutionary algorithm MAE-HTS was proposed, where the proposed algorithm based on two individuals was proposed to enhance the diversity of solutions [35].

2.3 Review for score strategy of MIDS

In this section, we briefly introduce the scoring strategy for the MIDS problem. During the search process, how to select candidate vertices is very important during the search process. The scoring function is recently proposed by Wang et al. [26]. Each vertex $v \in V$ has a property: path cost, denoted as $pc[v]$. It works as follows:

1) At the beginning, $pc[v] = 1$ for $\forall v \in V$;
2) At the end of each iteration of local search, $pc[v] = pc[v] + 1$ for each non-dominated vertex v.

Based on the above property of path cost, we introduce the path cost based scoring function denoted as sc to decide how to select candidate vertices for addition or deletion in each step of local search. The scoring function sc is defined as follows:

$$sc(v_i) = \begin{cases} \sum_{u \in N[v_i] \wedge inde[u]=0} pc(u), & \forall v_i \notin S, inde[v_i] = 0, \\ 0, & \forall v_i \notin S, inde[v_i] \neq 0, \\ -\sum_{u \in N[v_i] \wedge inde[u]=1} pc(u), & \forall v_i \in S. \end{cases}$$

In the above formula: $inde[u]$ is used to denote the number of the close neighborhood of a vertex u dominated by the candidate solution S. We can see the benefits of changing vertex state intuitively through the positive and negative values of the function $sc(v_i)$. Assuming that $v_i \notin S$, $sc(v_i)$ is non-negative, and we can see that $u \in N[v_i]$ with $inde[u] = 0$ is a set of non-dominated vertex sets that can be dominated by adding v_i to S. Similarly, if $v_i \in S$, $sc(v_i)$ is negative since $u \in N[v_i]$ with $inde[u] = 1$ is a set of dominated vertices that can be non-dominated by removing v_i from S.

3 A novel master-apprentice evolutionary algorithm for MIDS

In this section, we present a novel master-apprentice evolutionary algorithm called MAE-PB based on the general master-apprentice evolutionary framework [30]. The primary innovative ingredients of the proposed MAE-PB algorithm include the modified framework to be suitable for solving the MIDS problem, a path-breaking strategy based on the similarity of solutions to control the balance between search intensification and diversification, and a fast local search to further improve the quality of the solution.

3.1 General scheme

The proposed MAE-PB algorithm (see the flowchart in Fig. 1) consists of five main components: master-apprentice initialization, path-breaking distribution, local search, master-apprentice updating, and apprentice re-initialization. The pseudocode of the MAE-PB is shown in Algorithm 1.

Initially, the algorithm initials two individuals S_1 and S_2 by calling the *Construct* function (line 1), which will be introduced in Section 3.2. Specifically, the algorithm first constructs a feasible solution S_1, which is an IDS. Then, the algorithm attempts to generate an initial solution S_2 by finding a feasible solution in which $|S_2|$ is smaller than $|S_1|$; otherwise, an infeasible solution whose size is $|S_1| - 1$. Then, the algorithm begins with the global optimal solution S^* and the optimal solution in the previous round S_p^* by using a better feasible solution between S_1 and S_2 (lines 2 and 3). If S_2 is a feasible solution, that is, both S_2 and S_1 are independent

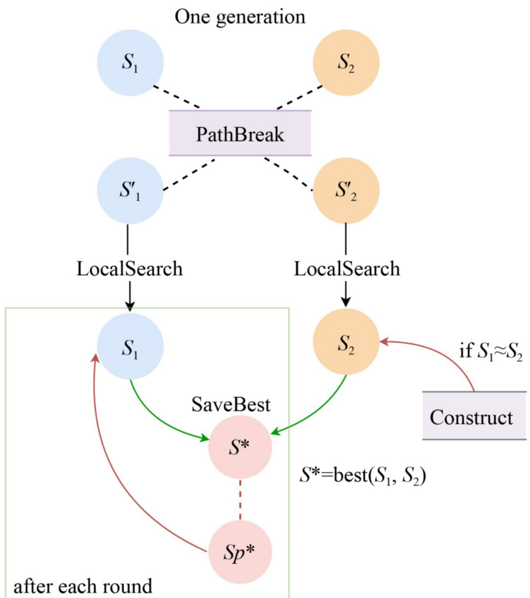

Fig. 1 The flowchart of MAE-PB

Algorithm 1 The MAE-PB algorithm

Input: A graph $G = (V, E)$, the *cutoff* time and parameters $\beta, \gamma, s, \pi, \theta, \alpha$
Output: An independent dominating set of G
// construct an initial solution; Sect. 3.2
1 $S_1 := Construct(|V|)$, $S_2 := Construct(|S_1|)$;
2 **if** S_2 is a feasible solution **then** $S^*_p := S^* := S_2$;
3 **else** $S^*_p := S^* := S_1$;
4 $total\_step := 1$;
5 **while** elapsed time < *cutoff* **do**
    // generate an offspring solution; Sect. 3.3
6     $S'_1 := PathBreak(S_1, S_2, \beta)$, $S'_2 := PathBreak(S_2, S_1, \beta)$;
    // improve a solution by local search; Sect. 3.4
7     $S_1 := LocalSearch(S'_1, \gamma, s, \pi)$, $S_2 := LocalSearch(S'_2, \gamma, s, \pi)$;
8   **if** S_1 is a feasible solution **then**
9   **if** S_2 is a feasible solution and $|S_2| < |S_1|$ **then** $S^* := S_2$;
10   **else** $S^* := S_1$;
11   **else if** S_2 is a feasible solution **then** $S^* := S_2$;
12   **if** $total\_step \% \theta == 0$ **then**
13         $S_1 := S^*_p$, $S^*_p := S^*$;
    // perturb a solution;
14     $similarity = \frac{|S1 \cap S2|}{max\{|S1|, |S2|\}}$;
15   **if** $similarity > \alpha$ **then** $S_2 := Construct(|V|)$;
16     $total\_step := total\_step + 1$;
17 **return** S^*;

dominating sets and S_2 is better than S_1, then S^* and Sp^* should be updated by S_2. Otherwise, S^* and Sp^* are updated by S_1. During the following search process, *total_step* is used to record the number of total steps (line 4).

After initialization, the algorithm executes a loop until the time limit is reached (lines 5–16), and then the best-obtained solution S^* is returned (line 17). During the loop, the algorithm combines the respective properties of S_1 and S_2 to produce two offspring solutions S'_1 and S'_2 by performing the *PathBreak* function, which will be introduced in Section 3.3 (line 6). For the newly generated solutions S'_1 and S'_2, the algorithm improves them through the local search process *LocalSearch* (which will be mentioned in Section 3.4) (line 7). After each step of the local search process, we use S^* to save the global optimal solution (lines 8–10). After one round (i.e., every θ step), S_1 is reset to the best solution in the previous round (i.e., S^*_p) and S^*_p is updated by the best solution in the current round (i.e., S^*) (lines 12 and 13). In the next step, we define a similarity function *similarity* to denote the ratio of the same vertices in S^*_p and S^*. When the similarity of S_1 and S_2 is very high, the S_2 solution is reconstructed by calling the *Construct* function (line 15). At the end of each step, *total_step* is increased by one (line 16).

3.2 The construction function for MIDS

The proposed MAE-PB algorithm uses the *Construct* function to complete two tasks, including initializing two individuals S_1 and S_2 (line 1 in Algorithm 1) and reconstructing an individual S_1 when the ratio of similarity is very high (line 15 in Algorithm 1). The pseudocode of the *Construct* function is presented in Algorithm 2.

Algorithm 2 The construct function

Input: the best size of a solution obtained *max_size*
Output: An initial solution S
1 $S := \emptyset$;
2 **while** $|S| < max\_size - 1$ **do**
3   select a vertex v with the biggest sc value from $V \setminus S$, breaking ties randomly;
4     $S := S \cup \{v\}$;
5   **if** S is a feasible solution **then**
6   **return** S;
7 **return** S;

First, candidate solution S is set to an empty set (line 1). *Construct* tries to greedily construct a feasible solution S by iteratively adding a vertex with the largest sc value. If *Construct* finds a feasible solution S, then S will be returned. Otherwise, the algorithm returns an infeasible solution S whose size equals *max_size*-1.

3.3 The PathBreak strategy for MIDS

In this section, we use a new path-breaking strategy called *PathBreak* to generate a new sub-solution by reconnecting the paths of the two individuals. The original path-breaking strategy was proposed by Xu et al. [36] was used as an effective local search algorithm to solve the MaxSAT problem by improving the idea of path relinking. The trajectory structure between the elite solution and the inverse solution is broken by flipping the variable, and the search allows only high-quality solutions to be focused. The path-break strategy randomizes the construction of the trajectory sequence. If the search falls in the local optimal solution, a strong mutation of the random flip variable is performed. If the search needs to be further dispersed, a weak mutation is performed. If the mutation does not allow the improvement of the local optimal solution, the search is restarted. The difference between our path-breaking strategy and the original one is that our algorithm improves two different candidate solutions instead of the current solution and its inverse solution. Second, we flip the variable by probability, that is, the set of adding or deleting vertices is not only determined by the trajectory of a solution to its inverse solution but also by the number of same vertices in both candidate solutions. The detailed process of *PathBreak* is described in Algorithm 3 [1)].

Algorithm 3 The *PathBreak* function

Input: a starting solution S_s, an ending solution S_e and parameter β
Output: a candidate solution S
1 $S_{sr} := S_s \setminus S_e$, $S_{er} := S_e \setminus S_s$ and $S_{same} := S_s \cap S_e$;
2 $S := S_{cr} := \emptyset$;
3 **if** $|S_{same}| > \frac{|S_s|}{2}$ **then** $S := S_{sr}$ and $S_{cr} := S_{same}$;
4 **else** $S := S_{same}$ and $S_{cr} := S_{sr}$;
5 **while** $S_{cr} \neq \emptyset$ **do**
6   select a random vertex v from S_{cr};
7     $S_{cr} := S_{cr} \setminus \{v\}$;
8   **if** $rand()\%100 \leqslant \beta$ **then** $S := S \cup \{v\}$;
9 **while** $S_{er} \neq \emptyset \&\& |S| < |S^*|$ **do**
10   select a random vertex v from S_{er};
11     $S_{er} := S_{er} \setminus \{v\}$;
12   **if** $rand()\%100 \leqslant \beta$ **then** $S := S \cup \{v\}$;
13 $Conflict := \{(v,u)|(v,u) \in E, v \in S, u \in S\}$;
14 **while** $Conflict \neq \emptyset$ **do**
15   select a random edge e from $Conflict$;
16   select a random vertex w from the two endpoints of e;
17     $Conflict := Conflict \setminus \{(w,u)|u \in S, (w,u) \in E\}$;
18     $S := S \setminus \{w\}$;
19 **if** S is a feasible solution **then** $S^* := S$;
20 **return** S;

The proposed *PathBreak* algorithm inputs two solutions, including a starting solution S_s and an ending solution S_e. First, we use three candidate sets to denote parts of the above solutions. In particular, the vertices that exist in S_s but not in S_e are regarded as S_{sr}; the vertices that exist in S_e but do not exist in S_s are recorded as S_{er}, and S_{same} is the same part in S_s and S_r (line 1). The candidate solution S and the temporary set S_{cr} are initialized as empty sets (line 2). If the number of S_{same} is larger than half of the number of vertices in S_s, then the candidate solution S is set to S_{sr} and S_{cr} is set to the remaining part of S_s (line 3). Otherwise, $S = S_{same}$ and $S_{cr} = S_{sr}$ (line 4). This shows that the strategy uses S to store a small part between S_{same} and S_{sr}. The strategy randomly pops a vertex v from S_{cr}, and then the vertex v is added to S with probability β until S_{cr} is empty (lines 5–8). When S_{er} is not an empty set and the size of S is smaller than S^*, the algorithm adds a random vertex v from S_{er} (lines 9–12). Subsequently, the algorithm uses a set $Conflict$ to store edges whose endpoints both belong to S (line 13). If there exist some edges in $Conflict$, the algorithm randomly picks a conflicting edge e from $Conflict$ and then among its endpoints it further selects a random endpoint w (lines 15 and 16). The corresponding conflicting set $Conflict$ should be updated (line 17), and vertex w is removed from the candidate solution S (line 18). Finally, if S is a feasible solution, which means that the algorithm obtains a better solution, then S^* is updated by S.

3.4 The local search algorithm for MIDS

The purpose of the local search is to move the current candidate solution to its neighborhood in some corresponding spaces. The proposed local search algorithm uses a tabu mechanism to overcome the cycling problem [37]. The pseudocode of *Local Search* is shown in Algorithm 4.

Algorithm 4 The *Local Search* algorithm

Input: a candidate solution S and parameters γ, s, π
Output: a current candidate solution S
1 $step := 1$, $tabu\_list := \emptyset$ and $marker := 0$;
2 **while** $step < inner\_step$ **do**
3   **if** S is a feasible solution and $|S| < |S^*|$ **then**
4         $S^* := S$, $step := 1$;
5         $marker := 1$;
6   choose a vertex $u_1 \in S$ with the highest sc value and $u_1 \notin tabu\_list$, breaking ties randomly;
7     $S := S \setminus \{u_1\}$;
8   **if** $rand()\%100 < \gamma \&\& N_2(u_1) \cap S \neq \emptyset$ **then**
9   choose a vertex $u_2 \in S$ with the highest sc value, breaking ties randomly;
10         $S := S \setminus \{u_2\}$ and $tabu\_list := \emptyset$;
11   choose a vertex v_1 from $V \setminus S$ with the highest sc value, breaking ties randomly;
12         $S := S \cup \{v_1\}$ and $tabu\_list := tabu\_list \cup \{v_1\}$;
13   **else** $tabu\_list := \emptyset$;
14   choose a vertex v_2 from $V \setminus S$ with the highest sc value, breaking ties randomly;
15     $S := S \cup \{v_2\}$ and $tabu\_list := tabu\_list \cup \{v_2\}$;
16     $pc(w) := pc(w) + 1$, for each non-dominated vertex w in V;
17     $step := step + 1$;
18   **if** $step\%s == 0$ **then**
19         $S := Perturb(S, \pi)$;
20 **if** $marker == 1$ **then return** S^*;
21 **else return** S;

The algorithm first initializes a marker variable $marker$, the number of steps $step$, and a tabu list $tabu\_list$ (line 1). The equation $marker = 1$ means that the following local search procedure finds a better solution, which is better than S^*; otherwise, $marker = 0$. The algorithm applies the local search procedure to improve the solution S until the limit of $step$ is reached, that is, $step \geqslant inner\_step$. In our work, $inner\_step$ is set to 10000. Finally, if $mark = 1$, then the best solution S^* is

1) In our algorithm, the range of values of $rand()$ is from 0 to RAND_MAX.

returned; otherwise, the algorithm returns the current candidate solution S (lines 20 and 21).

During the local search procedure, if the algorithm obtains a better solution, S^* is updated by S, *step* is set to 1, and the variable *marker* is marked as 1. Otherwise, the algorithm selects the vertex u_1 with the highest score value and inserts it into the candidate solution (lines 6 and 7). If $N_2(u_1) \cap S$ is not empty, with probability γ, the algorithm attempts to greedily remove a vertex u_2 from S (lines 9 and 10). After removing one or two vertices from S, *tabu_list* should be cleared (lines 10 and 13). In the next step, the algorithm greedily adds one vertex (i.e., v_1) or two vertices (i.e., v_1 and v_2) into S (lines 11, 12, 14, and 15). After the addition operations, these simply added vertices need to be added to *tabu_list*. The *pc* values of the corresponding vertices and *step* should be updated (lines 16 and 17). At the end of each step, if $step\%s == 0$, it means that no better candidate solution is found after s steps. Thus, the algorithm will use two perturbation methods to modify the current candidate solution (lines 18 and 19).

3.5 The perturbation framework for MIDS

In this section, we propose a perturbation procedure called *Perturb* to disturb the current candidate solution. In our work, for a great candidate solution, the *Perturb* function uses the same probability to select two different perturbation methods. Specifically, the first perturbation method aims to greedily remove some vertices from the candidate solution and then add back some other vertices by using a random addition technique based on restricted candidate lists [38]. The second perturbation method focuses on selecting vertices dominated by the candidate solution and not the candidate solution. We relax the limitation condition to add these vertices to the candidate solution without considering the independent constraint of the MIDS problem. During the addition process, we prefer to select one of these vertices that can dominate as many non-dominated vertices as possible. If there exists more than one vertex satisfying the above condition, we choose a vertex with the largest *inde* value to modify the candidate solution to a certain extent. This means that to make the candidate solution still feasible after adding it to the candidate solution, we have to remove all of its neighbors from the candidate solution. The scoring function in the second perturbation way is defined as below.

$$sc_1(v) = \sum_{u \in N[v] \wedge inde[u]=0} pc(u).$$

Based on the above scoring function, we propose a perturbation scoring rule.

Perturbation scoring rule Selecting a vertex v with $inde[v] \neq 0$ from $V \setminus S$, which has the largest sc_1 value, breaking ties by selecting the one with the largest *inde* value.

The selected vertex v has already been dominated by other vertices in the candidate solution, that is, $inde[v] \neq 0$. If the algorithm adds v to the candidate solution, the algorithm has to remove v's neighbor from the candidate solution to make the solution feasible, that is, the number of removed vertices is $inde[v]$ in total. Thus, when meeting that several vertices have the same best sc_1 value, for sufficiently disturbing the candidate solution, the algorithm picks the one among them with the highest *inde* value.

Note that the reason the algorithm uses different perturbation ways is to explore various parts of the entire search space as much as possible.

The *Perturb* function is displayed in Algorithm 5. The probability that the algorithm uses the first perturbation method is 50% (lines 1–11). The other half is called the second perturbation method (lines 12–22). During the first perturbation, the *Perturb* algorithm sets the parameter k to half the size of the candidate solution. To deal with massive graphs, the algorithm limits the value of k; thus, in our work, the maximum number of k is set to 100, which means that the algorithm removes at most k vertices from the candidate solution (lines 2–5). The algorithm computes the maximum and minimum score values of vertices from $V \setminus S$, and then sc_{rcl} is calculated based on sc_{max} and sc_{min} (lines 6 and 7). During the addition process, the algorithm adds vertices back into the candidate solution (lines 8–11). In each step, the algorithm selects a random vertex v whose score value is larger than sc_{rcl}, and the selected vertex v is added to S. sc_{max}, sc_{min}, and sc_{rcl} need to be updated accordingly. If the algorithm finds a better solution, then S^* is updated by S, and the algorithm jumps out of the adding process. During the second perturbation method, the algorithm tries to select a

Algorithm 5 The *Perturb* algorithm

Input: a candidate solution S and parameter π
Output: a current candidate solution S

1 **if** rand()%100 < 50 **then**
  // the first perturbation way
2   $k = \min(\frac{|S|}{2}, 100)$;
3   **for** $t := 0; t < k; t++$ **do**
4   choose a vertex $u \in S$ with the highest sc value, breaking ties randomly;
5     $S := S \setminus \{u\}$;
6   $sc_{max} = \max\{sc(v) \mid v \in V \setminus S\}$ and $sc_{min} = \min\{sc(v) \mid v \in V \setminus S\}$;
7   $sc_{rcl} = sc_{min} + \pi \times (sc_{max} - sc_{min})$;
8   **for** $t := 0; t < k; t++$ **do**
9   choose a random vertex v with $sc(v) \geqslant sc_{rcl}$;
10     $S := S \cup \{v\}$ and update the sc_{max}, sc_{min} and sc_{rcl};
11   **if** S is a feasible solution **then** $S^* := S$, **break** ;
12 **else**
  // the second perturbation way
13   $current\_size := |S|$;
14   find a vertex v_1 based on **Perturbation scoring rule**;
15   $S := S \cup \{v_1\}$;
16   **while** $N(v_1) \cap S \neq \emptyset$ **do**
17   select a random vertex u from $N(v_1) \cap S$;
18     $S := S \setminus \{u\}$;
19   **while** $|S| < current\_size$ **do**
20   **if** S is a feasible solution **then** $S^* := S$, **break** ;
21   find a vertex v_2 from $V \setminus S$ with the biggest sc value, breaking ties randomly;
22     $S := S \cup \{v_2\}$;
23 **return** S;

vertex not in the candidate solution with the largest sc_1 value to be added into S (lines 14 and 15). To maintain solution feasibility, the algorithm removes vertices in $N(v_1)\cap S$ (lines 16–18). To increase the size of the candidate solution, the algorithm greedily adds a vertex to the candidate solution until $|S|$ is not smaller than *current_size* (lines 19–22). At last, the perturbation solution S is returned (line 23).

4 Experiments

In this section, we evaluate the performance of the MAE-PB algorithm on a large number of benchmark instances commonly used in the literature and compare it with state-of-the-art results in the literature. We first introduce these benchmarks and experimental preliminaries. Then, we will display our parameter setting as well as the detailed results of our algorithm and all competitors. Finally, we present experiments to obtain insights into the influences of the components of the MAE-PB algorithm: a perturbation method and path-breaking.

4.1 The benchmarks

The benchmark instances of the MIDS tested in our experiments are widely used in the literature, and can be divided into two parts, including two classical benchmarks (i.e., DIMACS and BHOSLIB) and a suite of real-world massive graphs.

- DIMACS benchmark [39]: DIMACS is most commonly used for the comparison and evaluation of graph algorithms [40,41]. More specifically, the size of the DIMACS instances ranges from less than 150 vertices and 300 edges to more than 4,000 vertices and 7,900,000 edges. To test the effectiveness of the algorithm, we tested it on the complement graphs of some instances, including the sets of c-fat and p-hat. In total, 61 instances were selected.
- BHOSLIB benchmark [42]: The BHOSLIB benchmark is randomly generated based on the RB model and contains a total of 41 instances, of which a large instance named frb100-40 has 4,000 vertices and 572,774 edges. Owing to the hardness of BHOSLIB, it has been widely used as a reference benchmark for local search algorithms in recent literature [43,44].
- Real-world massive graphs [45]: In this study, we consider 187 real-world massive graphs from a network data repository online. They have recently been used in the performance of heuristic algorithms for some NP-hard problems [21,46,47]. All these massive real-world graphs have a massive number of vertices, but they all belong to sparse graphs. We ignore some massive graphs with fewer than 100,000 vertices and fewer than 1,000,000 edges. Thus, in this study, 65 instances are considered.

4.2 Experimental preliminaries

To evaluate the performance of the proposed MAE-PB algorithm, we compared it with five competitors: GRASP+PC [26], MEMETIC [27], drMIDS [30], ILPS2 [28], and ILPS3 [28], where ILPS2 and ILPS3 have different k values. All the algorithms are implemented in C++ and compiled with g++ by the -O3 option. For each instance, all algorithms independently performed 30 runs with different random seeds from 1 to 30. The time limit of all algorithms for DIMACS and BHOSLIB was set to 200 s, while the time limit for massive graphs was set to 1000 s. For each instance, *min* denotes the best size found (i.e., the minimal solution value), and *avg* denotes the average size obtained over 30 runs. The bold values in the table indicate the best solution among all the algorithms. If an algorithm fails to provide a solution within the given time limit, it is indicated by "N/A".

4.3 Parameter settings of the MAE-PB algorithm

In this section, we present the parameter adjustment experiment of the MAE-PB algorithm. Because the parameters in the experiment will affect the efficiency of the local search, the adjustment of the parameters is an indispensable and important step.

In this study, we used the automatic configuration tool irace [48] to obtain well-tuned parameters for the proposed MAE-PB algorithm, including θ, α, β, γ, s, and π. The training set was restricted to include all instances from the three benchmarks. The tuning process is given a limit of 10,000 runs with a time limit of 1,000 s per run. The results of the tuning processes are shown in Table 1. In detail, for the parameter θ involved in Algorithm 1, we assign parameter θ to 5. Specifically, after each round (i.e., every θ step), we make some adjustments to the solutions. For the parameter α involved in Algorithm 1, we assign parameter α to 0.7, which means that if the similarity of the candidate solution S_1 and S_2 is very large, then S_2 will be reconstructed. We set parameter β to 0.5, in Algorithm 3, which means that the algorithm adds vertices with a probability of β. Also, for the parameter γ involved in Algorithm 4, we set parameter γ to 0.4. With the probability of γ, the vertices are removed from the candidate solution. For parameter s also involved in Algorithm 4, we set s to 500. After every s step, we make some adjustments to the solutions. For the parameter π involved in Algorithm 5, we set the parameter π to 0.8, which is the range of the restricted

Table 1 Parameter settings of the MAE-PB algorithm

Parameter	Ranges	Description	Final values
θ	{2, 5, 8 }	The number of each round	5
α	{0.4, 0.5, 0.6, 0.7, 0.8}	The similarity of candidate solutions	0.7
β	{40%, 50%, 60%, 70%, 80%}	The probability of remove vertices	50%
γ	{40%, 50%, 60%, 70%, 80%}	The probability of remove vertices	40%
s	{200, 500, 800}	The number of each round	500
π	{0.4, 0.5, 0.6, 0.7, 0.8, 0.9}	The range of restricted candidate list	0.8

candidate list.

For all competitors, we set the same parameters as those described in the corresponding literature and optimized these parameters for the newly added massive graphs using the irace tool [48].

4.4 Results on DIMACS benchmark

In comparison, Tables 2 and 3 report that our MAE-PB algorithm finds better solutions than GRASP+PC, MEMETIC, drMIDS, ILSP2, and ILSP3 on 11, 6, 1, 7, and 7 instances, respectively. In the case of finding the same minimum value between our algorithm and the comparison algorithms, the MAE-PB algorithm finds smaller average values on 15, 6, 3, 14, and 13 instances than GRASP+PC, MEMETIC, drMIDS, ILSP2, and ILSP3, respectively. The proposed MAE-PB algorithm fails to find a better average solution value than the drMIDS algorithm on only one instance, C1000.9, and the gap between these two algorithms is small.

4.5 Results on BHOSLIB benchmark

Table 4 shows the experimental results of our algorithm and its competitors on the BHOSLIB benchmark. It is obvious from the results in the table that our algorithm yields better results than the other algorithms for most instances. In particular, we firstly compare the MAE-PB algorithm with GRASP+PC, MEMETIC and drMIDS. The MAE-PB algorithm finds better solutions than GRASP+PC, MEMETIC, and drMIDS on 32, 28, and 8 instances, respectively. The average values obtained by our algorithm are better than those of GRASP+PC, MEMETIC, and drMIDS for 8, 11, and 20 instances, respectively. In addition, the MAE-PB algorithm finds better solutions than ILSP2 and ILSP3 on 18 and 22 instances, respectively, while the average values obtained by our algorithm are better than those of ILSP2 and ILSP3 for 22 and 20 instances, respectively. However, in instance frb59-26-2, our algorithm fails to obtain the best solution value.

4.6 Results on massive graph

Comparing the MAE-PB algorithm and the competitor algorithm on a massive graph, Tables 5 and 6 report the minimum and average values of the experimental results. The MAE-PB algorithm finds the best solution for 60 instances with only three exceptions, which intuitively verifies its performance.

If we have a tie between the proposed MAE-PB and any of the other five competitors concerning solution quality, that is, the same minimal and average solution values, we compare these algorithms in terms of computation times for all the benchmarks. As shown in Fig. 2, MAE-PB can obtain the best solution in less time than the other five algorithms.

4.7 Critical difference analysis

This section evaluates the statistical difference between the proposed MAE-PB algorithm and the five competitors on the selected three benchmarks in the form of a critical difference graph. First, we use the Friedman test [49] to formulate the null hypothesis that the proposed MAE-PB algorithm and its five competitors are equivalent in terms of performance. The above results are displayed in Fig. 3 using a critical difference diagram [50]. The top line in each sub-graph is the axis where

Table 2 Experimental Results on the DIMACS benchmark I

Instance	GRASP+PC		MEMETIC		drMIDS		ILPS2		ILPS3		MAE-PB	
	min	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*
brock200_2	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
brock200_4	**6**	6.3	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
brock400_2	10	10	**9**	9.3	**9**	**9**	10	10	**9**	10	**9**	**9**
brock400_4	**9**	9.3	**9**	**9**	**9**	**9**	**9**	9.9	**9**	10	**9**	**9**
brock800_2	**8**	8.2	**8**	8.1	**8**	**8**	**8**	8.7	**8**	8.9	**8**	**8**
brock800_4	**8**	8.2	**8**	**8**	**8**	**8**	**8**	8.5	**8**	8.8	**8**	**8**
C1000.9	26	26.9	26	27.5	**25**	**25.5**	27	28	27	27.8	**25**	26
C125.9	15	15	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
C2000.5	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
C2000.9	33	33.2	33	33.5	32	32	32	33.8	32	34	**31**	**31.7**
C250.9	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**
C4000.5	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
C500.9	23	23	22	22	**21**	**21**	22	22.2	22	22.3	**21**	**21**
c-fat200-1.clq	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**	**13**
c-fat200-2.clq	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
c-fat200-5.clq	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
c-fat500-1.clq	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**
c-fat500-2.clq	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**	**14**
c-fat500-5.clq	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
DSJC1000.5	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
DSJC500.5	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
gen200_p0.9_44	**16**	16.1	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**
gen200_p0.9_55	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**
gen400_p0.9_55	21	21.2	**20**	**20**	**20**	**20**	**20**	20.1	**20**	20.3	**20**	**20**
gen400_p0.9_65	21	21.1	**20**	20.1	**20**	**20**	**20**	20.8	**20**	20.8	**20**	**20**
gen400_p0.9_75	**20**	20.4	**20**	20.3	**20**	**20**	**20**	20.8	**20**	20.6	**20**	**20**
hamming10-4	**12**	12.3	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**

Table 3 Experimental results on the DIMACS benchmark II

Instance	GRASP+PC		MEMETIC		drMIDS		ILPS2		ILPS3		MAE-PB	
	min	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*
hamming6-2	**12**	12.8	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
hamming6-4	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
hamming8-2	**32**	40.1	36	43.1	**32**	**32**	36	36	36	36	**32**	**32**
hamming8-4	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
johnson16-2-4	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**	**8**
johnson32-2-4	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**
johnson8-2-4	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
johnson8-4-4	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
keller4	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
keller5	**9**	9.4	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
keller6	17	17.6	17	17.9	**15**	17.2	17	18	18	18.3	**15**	**15.1**
MANN_a27	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**	**27**
MANN_a45	**45**	**45**	**45**	**45**	**45**	**45**	**45**	**45**	**45**	**45**	**45**	**45**
MANN_a81	**81**	**81**	**81**	**81**	**81**	**81**	**81**	**81**	**81**	**81**	**81**	**81**
MANN_a9	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
p_hat1500-1.clq	13	13.4	13	13.9	**12**	12.7	13	14.1	13	14.3	**12**	**12.4**
p_hat1500-2.clq	**7**	8	**7**	7.9	**7**	7.7	**7**	7.7	**7**	7.8	**7**	**7.2**
p_hat1500-3.clq	**3**	**3**	**3**	**3**	**3**	**3**	**3**	3.1	**3**	3.3	**3**	**3**
p_hat300-1.clq	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
p_hat300-2.clq	**5**	5.1	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
p_hat300-3.clq	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
p_hat700-1.clq	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	**11**	11.2	**11**	**11**
p_hat700-2.clq	**6**	6.5	**6**	6.3	**6**	**6**	**6**	6.6	**6**	6.4	**6**	**6**
p_hat700-3.clq	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
san1000	**4**	**4**	**4**	**4**	**4**	**4**	**4**	4.7	**4**	4.2	**4**	**4**
san200_0.7_1	7	7	**6**	**6**	**6**	**6**	**6**	6.1	**6**	6.8	**6**	**6**
san200_0.7_2	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
san200_0.9_1	16	16	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**
san200_0.9_2	**16**	16.4	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**
san200_0.9_3	**15**	15.1	**15**	**15**	**15**	**15**	**15**	15.3	**15**	15.1	**15**	**15**
san400_0.5_1	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
san400_0.7_1	**7**	7.1	**7**	**7**	**7**	**7**	**7**	7.9	8	8	**7**	**7**
san400_0.7_2	**7**	**7**	**7**	**7**	**7**	**7**	**7**	7.6	**7**	7.9	**7**	**7**
san400_0.7_3	8	8	**7**	**7**	**7**	**7**	**7**	7.8	8	8	**7**	**7**

Table 4 Experimental results on the BHOSLIB benchmark

Instance	GRASP+PC		MEMETIC		drMIDS		ILPS2		ILPS3		MAE-PB	
	min	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*
frb30-15-1	**11**	**11**	**11**	**11**	**11**	**11**	**11**	11.2	**11**	11.7	**11**	**11**
frb30-15-2	**11**	11.4	**11**	11.1	**11**	**11**	**11**	11.7	**11**	11.9	**11**	**11**
frb30-15-3	12	12	**11**	11.2	**11**	**11**	**11**	11.9	**11**	11.9	**11**	**11**
frb30-15-4	12	12	**11**	**11**	**11**	**11**	**11**	11.2	**11**	11.7	**11**	**11**
frb30-15-5	**11**	11.2	**11**	11.3	**11**	**11**	**11**	11.7	**11**	11.9	**11**	**11**
frb35-17-1	14	14	**13**	13.6	**13**	**13**	**13**	13.9	**13**	14	**13**	**13**
frb35-17-2	**13**	13.7	**13**	13.9	**13**	**13**	**13**	13.8	**13**	14	**13**	**13**
frb35-17-3	**13**	13.3	**13**	13.6	**13**	**13**	**13**	13.8	**13**	14	**13**	**13**
frb35-17-4	14	14	**13**	13.9	**13**	**13.3**	**13**	13.8	**13**	13.9	**13**	**13.3**
frb35-17-5	14	14	14	14	**13**	13.6	14	14	14	14.2	**13**	**13.4**
frb40-19-1	16	16	16	16	**15**	15.4	16	16.1	16	16.6	**15**	**15**
frb40-19-2	16	16	**15**	15.9	**15**	**15**	**15**	15.7	16	16.1	**15**	**15**
frb40-19-3	**15**	15.6	**15**	15.9	**15**	**15**	**15**	15.8	**15**	16.1	**15**	**15**
frb40-19-4	**15**	15.4	**15**	15.9	**15**	**15**	**15**	15.7	**15**	16	**15**	**15**
frb40-19-5	**15**	15.7	**15**	15.9	**15**	15.2	**15**	15.8	**15**	16	**15**	**15**
frb45-21-1	18	18	18	18.9	**17**	17.8	18	18.2	18	18.7	**17**	**17.5**
frb45-21-2	18	18	18	18.7	**17**	17.9	**17**	18	**17**	18.6	**17**	**17.6**
frb45-21-3	18	18.1	18	18.4	**17**	17.4	**17**	17.8	**17**	18.4	**17**	**17**
frb45-21-4	18	18	18	18.6	**17**	17.5	18	18.1	18	18.6	**17**	**17.1**
frb45-21-5	**17**	17.9	18	18.5	**17**	17.5	**17**	18	**17**	18.3	**17**	**17**
frb50-23-1	20	20	20	20.9	**19**	19.9	**19**	20.2	20	20.8	**19**	**19.5**
frb50-23-2	20	20.2	21	21	**19**	19.9	20	20.5	20	20.8	**19**	**19.8**

Table 4 (Continued)

Instance	GRASP+PC		MEMETIC		drMIDS		ILPS2		ILPS3		MAE-PB	
	min	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*
frb50-23-3	20	20.3	20	20.9	**19**	19.8	20	20.2	20	20.8	**19**	**19.6**
frb50-23-4	20	20.5	21	21.4	**19**	19.9	20	20.8	20	21	**19**	**19.8**
frb50-23-5	21	21	21	21.3	20	20	20	20.4	20	20.8	**19**	**19.7**
frb53-24-1	22	22.1	22	22.8	21	21.1	**20**	21.8	21	22.4	**20**	**20.8**
frb53-24-2	22	22	22	22.7	21	21.5	21	21.7	21	22.1	**20**	**21**
frb53-24-3	21	21.1	21	22.1	**20**	20.9	21	21.4	21	21.8	**20**	**20.7**
frb53-24-4	21	21.2	21	22	**20**	20.9	21	21.9	21	22.1	**20**	**20.4**
frb53-24-5	21	21.6	22	22.5	**20**	21.1	21	21.7	21	21.9	**20**	**20.9**
frb56-25-1	22	22.8	24	24.1	**21**	22.4	22	23.1	23	23.7	**21**	**22.1**
frb56-25-2	23	23.2	24	24.3	**22**	22.8	**22**	23.3	23	23.7	**22**	**22.5**
frb56-25-3	**22**	22.9	23	24	**22**	22.8	**22**	23.1	**22**	23.3	**22**	**22**
frb56-25-4	23	23.1	24	24.1	22	22.8	22	23.2	23	23.7	**21**	**22.4**
frb56-25-5	22	22.4	22	22.8	22	22.3	22	22.8	22	23.3	**21**	**21.9**
frb59-26-1	24	24.1	24	25.4	23	23.6	23	24.4	23	24.6	**22**	**23**
frb59-26-2	24	24.2	24	25.6	23	23.9	**22**	24	23	24.6	23	23.2
frb59-26-3	24	24.7	25	25.9	**23**	23.7	24	24.7	**23**	25	**23**	**23.8**
frb59-26-4	24	24.4	24	25.6	**23**	23.9	24	24.4	24	24.8	**23**	**23.6**
frb59-26-5	25	25.4	25	25.8	24	24.2	**23**	24.3	24	24.7	**23**	**23.8**
frb100-40	44	44.8	48	49.5	43	44.4	43	44.5	43	45.3	**42**	**43.4**

Table 5 Experimental results on massive graphs I

Instance	GRASP+PC		MEMETIC		drMIDS		ILPS2		ILPS3		MAE-PB	
	min	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*
bn-***0025865 ***1-bg	1198908	1199212.7	1205376	1206233.1	1197891	1200437.9	1197011	1197168.4	1197398	1197555.9	**1194499**	1196200
bn-***0025865 ***2-bg	1561094	1561624.3	1575465	1577527.3	1563155	1570989.5	1559833	1559866.6	1559492	1559696.1	**1556455**	1558023.6
ca-coauthors-dblp	49186	49250.3	52386	52494.1	44363	44989.5	44253	44288.9	44750	44772.2	**43035**	44088.8
ca-dblp-2012	94758	94938.6	110109	110325.1	87656	87812.8	87598	87689.9	110057	110196.5	**87508**	87698.1
ca-hollywood-2009	140801	141065.7	155263	155549.6	143519	146696.6	152861	153152.9	178556	178981.5	**128137**	128290.1
channel***-b050	486263	486449.8	491998	492225.5	489722	490215.8	420666	420853.7	566799	567380.7	**409978**	410258.8
dbpedia-link	N/A	N/A	8612327	8629696.2	8627747	8637426.5	8908597	8911196.6	N/A	N/A	**7533823**	7535700.8
delaunay_n22	864130	864485.6	868437	868881.7	865422	865856.5	806316	806730.5	1030255	1030610.1	**744805**	745267.7
delaunay_n23	1728925	1729304.8	1737228	1737950.7	1736327	1737009.7	1613584	1613809.2	2060811	2061390.1	**1489753**	1490376.1
delaunay_n24	3458413	3459279.8	3475398	3476268	3475650	3476635.9	N/A	N/A	N/A	N/A	**2979750**	2980281.9
friendster	4473674	4493361.4	6848800	6863274.8	6601967	6846328.6	7183568	7191256.2	7243029	7247583	**3547353**	3548971.8
hugebubbles-00020	7800496	7801942.6	7522388	7523370.2	7523382	7524471.3	6952078	6952078	N/A	N/A	**6800602**	6801761.9
hugetrace-00010	4446308	4447087.9	4285018	4285538.9	4284436	4285910.7	3962433	3963354.7	4687640	4688997.3	**3875488**	3876764.2
hugetrace-00020	5899134	5900269.8	5686663	5687271.6	5686893	5687917.5	5256729	5257882.7	6218412	6219513.8	**5142114**	5143085.2
inf-europe_osm	20767515	20768636.8	N/A	N/A	20052008	20053573.3	N/A	N/A	N/A	N/A	**18314284**	18315597.7
inf-germany_osm	4669787	4670887.1	4524573	4525242	4523638	4525116.1	4310573	4311235.7	5053976	5054966.2	**4134178**	4134983.6
inf-roadNet-CA	740604	740878.9	732830	733158.5	728927	729330.5	695837	696003.7	822287	822601.4	**662664**	662926.6
inf-roadNet-PA	412501	412731.3	408678	409058.2	401804	402169.8	386994	387294.5	458039	458370.8	**369370**	369601.5
inf-road-usa	9547166	9548928.8	9449606	9450335.3	9449603	9451604.6	9125541	9126508.7	10765734	10766635.9	**8610251**	8611245.9
rec-dating	40149	41157.5	51462	52377.3	48632	51806.4	36744	36767	36769	36790.8	**32671**	33502.1
rec-epinions	320240	368998.1	5663900	564657.2	N/A	N/A	595675	612617.7	602861	620295.4	**134669**	134715.3
rec-libimseti-dir	62046	66435.6	82520	85169.9	79938	85061.2	63429	63495.9	63483	63483	**50070**	53154.7
rgg_n_2_23_s0	858105	858435.1	867425	867715.9	865936	866444.8	736027	736356.3	954528	954785.4	**704494**	704696.2
rgg_n_2_24_s0	1656337	1656833.6	1674237	1674731.7	1673505	1674445.7	N/A	N/A	1839520	1839520	**1357335**	1357670
rt-retweet-crawl	470537	475864.1	890477	893937.5	531197	695539.5	971833	972959.2	965905	966947.8	**469708**	485375.2
sc-ldoor	68659	68718.7	70073	70123.5	67557	68547.1	68862	68962	79892	80020.7	**66770**	66846
sc-msdoor	22163	22192.2	22801	22840.9	21169	21542.3	21437	21484.2	**20912**	20939.4	21481	21517.8
sc-pwtk	6030	6046.4	6360	6389.7	4959	5065.2	5099	5126.8	5133	5164.1	**4475**	4528
sc-rel9	259632	260231.6	4237296	4262802	2110005	4090520.2	5379337	5388189.8	5382628	5392480.1	**241046**	241947
sc-shipsec1	12563	12594.7	13580	13659.1	10834	11022.9	10083	10120.5	**9696**	9743.6	10392	10443.8
sc-shipsec5	16791	16816.3	17606	17695.6	14918	15161.4	14178	14297.3	**13733**	13796.5	14533	14586.8
socfb-A-anon	1669228	1674202.7	2319836	2323819.5	2141665	2278578.5	2483752	2486549.9	2497057	2499903	**1337702**	1338843.8
socfb-B-anon	1606632	1613011.3	2271052	2276031.7	2056992	2225046.4	2428580	2431129.6	2438709	2441440.2	**1248897**	1249610.4
socfb-uci-uni	N/A	N/A	N/A	N/A	55837483	55860922	57147925	57154643.7	57162057	57167604.5	**8879317**	8879940.5

Table 6 Experimental results on massive graphs II

Instance	GRASP+PC		MEMETIC		drMIDS		ILPS2		ILPS3		MAE-PB	
	min	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*	*min*	*avg*
soc-buzznet	16427	41491.8	48972	60200.2	56933	60608.9	2571	2571.7	2573	2575.9	**1078**	2463.8
soc-delicious	257047	260709.1	375432	377337.6	229828	244509.8	410459	411412.9	400696	401662.7	**213040**	213148.9
soc-digg	464502	469247.5	592137	595356.4	541850	575911.8	620232	622005.9	628060	629787.3	**360827**	361056.8
soc-dogster	178127	187041.5	218847	222195.1	212787	220116.4	236456	236952.1	246708	247404.2	**147137**	147220.1
soc-flickr	238561	239177.3	285952	286537.5	228393	231800.9	315535	316061.4	329196	329659.9	**225706**	225986.8
soc-flickr-und	757567	759852.7	962166	963430.1	793220	847844.7	1094213	1094930.9	1133992	1134654	**712106**	712459.5
soc-flixster	1797967	1804468.4	2283393	2289703.9	2112006	2242842.7	2349351	2355308.1	2351118	2357745.3	**1446495**	1447358.9
soc-FourSquare	261585	263522.1	421911	423272.6	309284	343367.9	497910	499487.8	492209	493759.3	**254246**	263147
soc-lastfm	711394	715802.3	991546	994550.3	808133	919647.7	1049636	1055349.3	1045676	1051463	**606769**	623970.3
soc-livejournal	1556556	1557169.2	1701200	1702474.8	1569372	1610680.8	1763810	1764824.7	1888537	1889859.7	**1457679**	1458202
soc-livejournal-user-groups	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	**3935557**	3963727.7
soc-LiveMocha	27818	29799.8	46246	47163.3	25173	27104.9	19308	19326.2	19286	19312.7	**19164**	19393.4
soc-ljournal-2008	2178908	2180256.8	2392237	2393624.1	2245160	2304426.9	2471838	2472824.1	2625367	2627141.1	**2017074**	2017617.9
soc-orkut	487314	490131.8	547962	548734.1	511063	523932	571977	571977	N/A	N/A	**420253**	420702.4
soc-orkut-dir	496889	498147.2	558154	559021.8	528996	537980.2	N/A	N/A	N/A	N/A	**422147**	422761
soc-pokec	479023	479918.3	541129	541790.1	460459	473097.7	579862	580476.3	624805	625335.8	**444054**	444497.7
soc-sinaweibo	N/A	N/A	N/A	N/A	N/A	N/A	58189158	58189158	N/A	N/A	**41348112**	41348903.3
soc-twitter-higgs	136308	148028.7	187727	197706.7	194853	199584.3	64645	64781.9	64783	64838.1	**64637**	64689.4
soc-youtube	249474	252195.9	291048	294687.6	249714	263709.5	305632	306098.5	321759	322149.3	**210109**	210181.5
soc-youtube-snap	621236	628307.3	734256	736466.4	668462	696936.7	771399	772205.9	801033	801966.7	**516764**	516956.6
tech-as-skitter	504141	507896.5	807360	813966.7	700698	790524.5	999796	1001816.8	1044493	1046569	**425378**	425765.9
tech-ip	N/A	N/A	N/A	N/A	N/A	N/A	34033	34164.6	34033	34164.6	**33944**	34067
twitter_mpi	N/A	N/A	N/A	N/A	N/A	N/A	8636449	8647646.9	8674533	8687284.6	**5517459**	5518646.9
web-arabic-2005	29252	29478.1	35100	35346.4	25884	26176.7	26039	26233.0	25745	25951.4	**24497**	25286.2
web-baidu-baike	1041922	1097314.7	1281323	1281990.2	1279905	1285277.7	1339271	1340662.7	1388596	1389907.3	**892104**	892318.7
web-it-2004	67874	68537.2	80077	80201.2	62662	64220.9	82375	83130.1	67453	67454.5	**57896**	60208.1
web-uk-2005	1723	1726	1728	1729.6	1429	1432.5	1452	1530.4	1452	1528	**1427**	1427
web-wikipedia_link	N/A	N/A	N/A	N/A	N/A	N/A	1795791	1797987.3	1843338	1845944.2	**620531**	620718
web-wikipedia 2009	735795	737294.9	916510	918149.8	707187	761818.2	1032499	1033213	1097804	1098647.3	**682229**	682709.1
web-wikipedia-growth	558570	563152.9	690694	696448.4	700384	703726.3	773754	774938.2	833111	834491.2	**446746**	446931
wikipedia_link_en	24832213	24891236.9	29251560	26489886.4	26441864	26526564.8	26674651	26679001.5	26682855	26687149.9	**24841764**	24901940.4

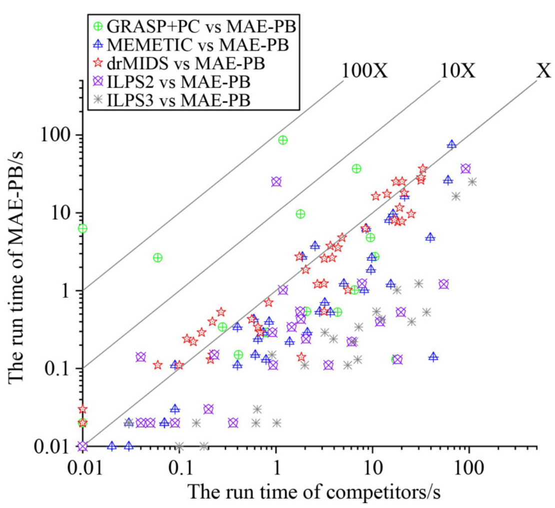

Fig. 2 Average run time of MAE-PB and competitors

the average ranks of the algorithms are plotted. The lower the ranks, the better the algorithm. If there is no significant difference between the MAE-PB algorithm and any of the five competitors, and the significance level is 0.05, then a link is established between them. It can be observed from the figure that almost all algorithms perform well on the DIMACS benchmark, and the results are relatively close. The quality of the solutions obtained by the MAE-PB algorithm under the other benchmarks was better than that of the competitors.

4.8 The effectiveness of the proposed components

In this subsection, to reflect the effectiveness of the proposed perturbation and path-breaking methods, we compare the results of the MAE-PB algorithm and the other five algorithms in the following five cases : (1) MAE-PB1 does not use any perturbation strategy; (2) MAE-PB2 only uses the first perturbation method in our algorithm; (3) MAE-PB3 only applies the second perturbation method in our algorithm; (4) MAE-PB4 only uses the original path-breaking strategy [36]; and (5) MAE-PB5 does not employ a path-breaking strategy. The comparison results of these algorithms are shown in Table 7 where #inst denotes the number of instances in each benchmark, while #better and #worse denote the number of instance families or instances where MAE-PB finds better and worse results, respectively.

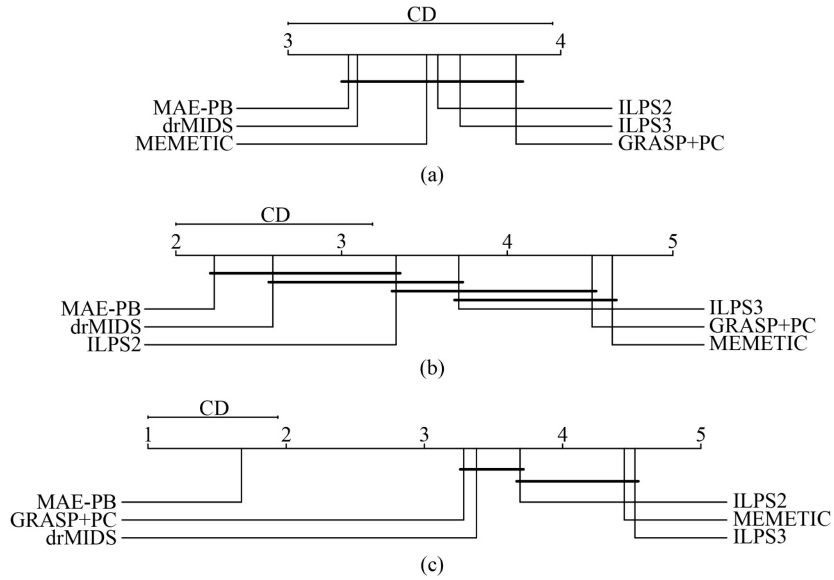

Fig. 3 Critical difference plots about MAE-PB, GRASP+PC, MEMETIC, drMIDS, ILPS2 and ILPS3 on each benchmark. (a) DIMACS; (b) BHOSLIB; (c) massive graph

Table 7 Summary results of comparing MAE-PB with its competitors on all benchmarks

Benchmark	#instance	vs. MAE-PB1		vs. MAE-PB2		vs. MAE-PB3		vs. MAE-PB4		vs. MAE-PB5	
		#better	#worse	#better	#worse	#better	#worse	#better	#worse	#better	#worse
DIMACS	61	**3**	0	**2**	0	**2**	0	**2**	0	**1**	0
BHOSLIB	41	**3**	1	**17**	0	**10**	0	**9**	0	**4**	1
massive graph	65	**45**	20	**36**	19	**35**	19	**43**	22	**40**	25
Total	167	**51**	21	**55**	19	**47**	19	**54**	22	**45**	26

From the results, it is obvious that if the algorithm does not use any perturbation or uses only one perturbation strategy, the results are not particularly good. In addition, the results demonstrate that our novel path-breaking strategy plays an important role in the performance of MAE-PB.

5 Conclusion

In this work, we introduced an improved MAE algorithm dedicated to solving the MIDS problem. First, to deeply explore the search space, the MAE-PB algorithm uses a multiple neighborhood-based local search function. Second, to enlarge the search space, the MAE-PB algorithm applies two novel perturbation methods to disturb the current candidate solution during the search process. Third, we propose a novel path-breaking strategy for solution recombination to deal with the problem of the high similarity between two candidate solutions. The experimental results show that the proposed MAE-PB performs better than the state-of-the-art MIDS heuristic algorithms in most instances.

For future work, given the success of MAE-PB in this work, we will consider if it may further improve the current algorithm for solving the MIDS problem if we combine other ideas [51–54]. Envisioned research directions regarding the proposed strategies include applying the new perturbation method to other NP-hard problems, such as k-submodular function optimization [55], the minimum vertex cover problem [56] and pseudo boolean optimization [57].

Acknowledgements This work was supported by the National Natural Science Foundation of China (Grant Nos. 61806050, 61972063, 61976050), the Fundamental Research Funds for the Central Universities (2412020FZ030, 2412019ZD013, 2412019FZ051), and Jilin Science and Technology Association (QT202005). Thanks Dr. Jianan Wang for offering the technical support of the computing server.

References

1. Samuel H, Zhuang W, Preiss B. DTN based dominating set routing for MANET in heterogeneous wireless networking. Mobile Networks and Applications, 2009, 14(2): 154–164
2. Abseher M, Musliu N, Woltran S. Improving the efficiency of dynamic programming on tree decompositions via machine learning. Journal of Artificial Intelligence Research, 2017, 58: 829–858
3. Aoun B, Boutaba R, Iraqi Y, Kenward G. Gateway placement optimization in wireless mesh networks with QoS constraints. IEEE Journal on Selected Areas in Communications, 2006, 24(11): 2127–2136
4. Potluri A, Bhagvati C. Novel morphological algorithms for dominating sets on graphs with applications to image analysis. In: Proceedings of the 15th International Workshop on Combinatorial Image Analysis. 2012, 249–262
5. Alofairi A A, Mabrouk E, Elsemman I E. Constraint-based models for dominating protein interaction networks. IET Systems Biology, 2021, 15(5): 148–162
6. Jin Y, Hao J K. General swap-based multiple neighborhood tabu search for the maximum independent set problem. Engineering Applications of Artificial Intelligence, 2015, 37: 20–33
7. Boginski V, Butenko S, Pardalos P M. Statistical analysis of financial networks. Computational Statistics & Data Analysis, 2005, 48(2): 431–443
8. Etzion T, Ostergard P R J. Greedy and heuristic algorithms for codes and colorings. IEEE Transactions on Information Theory, 1998, 44(1): 382–388

9. Akyildiz I F, Kasimoglu I H. Wireless sensor and actor networks: research challenges. Ad Hoc Networks, 2004, 2(4): 351–367
10. McLaughlan B, Akkaya K. Coverage-based clustering of wireless sensor and actor networks. In: Proceedings of IEEE International Conference on Pervasive Services. 2007, 45–54
11. Erciyes K, Dagdeviren O, Cokuslu D, Ozsoyeller D. Graph theoretic clustering algorithms in mobile ad hoc networks and wireless sensor networks. Applied and Computational Mathematics, 2007, 6(2): 162–180
12. Chen Y, Liestman A, Liu J. Clustering algorithms for ad hoc wireless networks. Ad Hoc and Sensor Networks, 2004, 28: 76−90
13. Lin C R, Gerla M. Adaptive clustering for mobile wireless networks. IEEE Journal on Selected areas in Communications, 1997, 15(7): 1265–1275
14. Basagni S. Distributed clustering for ad hoc networks. In: Proceedings of the 4th International Symposium on Parallel Architectures, Algorithms, and Networks. 1999, 310–315
15. Chen G, Nocetti F G, Gonzalez J S, Stojmenovic I. Connectivity based k-hop clustering in wireless networks. In: Proceedings of the 35th Annual Hawaii International Conference on System Sciences. 2002, 2450–2459
16. Garey M R, Johnson D S. Computers and Intractability: A Guide to the Theory of NP-Completeness. New York: W. H. Freeman, 1979
17. Gaspers S, Liedloff M. A branch-and-reduce algorithm for finding a minimum independent dominating set in graphs. In: Proceedings of the 32nd International Workshop on Graph-Theoretic Concepts in Computer Science. 2006, 78–89
18. Liu C, Song Y. Exact algorithms for finding the minimum independent dominating set in graphs. In: Proceedings of the 17th International Symposium on Algorithms and Computation. 2006, 439–448
19. Bourgeois N, Croce F D, Escoffier B, Paschos V T. Fast algorithms for min independent dominating set. Discrete Applied Mathematics, 2013, 161(4–5): 558–572
20. Liang Y, Huang H, Cai Z. PSO-ACSC: a large-scale evolutionary algorithm for image matting. Frontiers of Computer Science, 2020, 14(6): 146321
21. Wang Y, Cai S, Chen J, Yin M. SCCWalk: an efficient local search algorithm and its improvements for maximum weight clique problem. Artificial Intelligence, 2020, 280: 103230
22. Chen C, Gao L, Xie X, Wang Z. Enjoy the most beautiful scene now: a memetic algorithm to solve two-fold time-dependent arc orienteering problem. Frontiers of Computer Science, 2020, 14(2): 364–377
23. He P, Hao J K, Wu Q. Grouping memetic search for the colored traveling salesmen problem. Information Sciences, 2021, 570: 689–707
24. Wang Y, Li X, Wong K C, Chang Y, Yang S. Evolutionary multiobjective clustering algorithms with ensemble for patient stratification. IEEE Transactions on Cybernetics, 2021, doi: 10.1109/TCYB.2021.3069434
25. Liu L, Du Y. An improved multi-objective evolutionary algorithm for computation offloading in the multi-cloudlet environment. Frontiers of Computer Science, 2021, 15(5): 155503
26. Wang Y, Li R, Zhou Y, Yin M. A path cost-based grasp for minimum independent dominating set problem. Neural Computing and Applications, 2017, 28(S1): 143–151
27. Wang Y, Chen J, Sun H, Yin M. A memetic algorithm for minimum independent dominating set problem. Neural Computing and Applications, 2018, 30(8): 2519–2529
28. Haraguchi K. An efficient local search for the minimum independent dominating set problem. In: Proceedings of the 17th International Symposium on Experimental Algorithms. 2018, 13
29. Wang Y, Li C, Yin M. A two phase removing algorithm for minimum independent dominating set problem. Applied Soft Computing, 2020, 88: 105949
30. Ding J, Lü Z, Li C M, Shen L, Xu L, Glover F. A two-individual based evolutionary algorithm for the flexible job shop scheduling problem. In: Proceedings of the AAAI Conference on Artificial Intelligence. 2019, 280
31. Moalic L, Gondran A. Variations on memetic algorithms for graph coloring problems. Journal of Heuristics, 2018, 24(1): 1–24
32. Peng B, Zhang Y, Cheng T C E, Lü Z, Punnen A P. A two-individual based path-relinking algorithm for the satellite broadcast scheduling problem. Knowledge-Based Systems, 2020, 196: 105774
33. Zheng P, Zhang P, Wang J, Zhang J, Yang C, Jin Y. A data-driven robust optimization method for the assembly job-shop scheduling problem under uncertainty. International Journal of Computer Integrated Manufacturing, 2020, doi: 10.1080/0951192X.2020.1803506
34. Sun Q, Dou J, Zhang C. Robust optimization of flow shop scheduling with uncertain processing time. In: Proceedings of 2020 IEEE International Conference on Mechatronics and Automation. 2020, 512–517
35. Wang Y, Lü Z, Punnen A P. A fast and robust heuristic algorithm for the minimum weight vertex cover problem. IEEE Access, 2021, 9: 31932–31945
36. Xu Z, He K, Li C M. An iterative path-breaking approach with mutation and restart strategies for the max-sat problem. Computers & Operations Research, 2019, 104: 49–58
37. Glover F. Tabu search—part I. ORSA Journal on Computing, 1989, 1(3): 190–206
38. Feo T A, Resende M G C. Greedy randomized adaptive search procedures. Journal of Global Optimization, 1995, 6(2): 109–133
39. Trick M A, Johnson D S. Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11-13, 1993. Boston: American Mathematical Society, 1996
40. Zhou Y, Hao J K, Duval B. Reinforcement learning based local search for grouping problems: A case study on graph coloring. Expert Systems with Applications, 2016, 64: 412–422
41. Wang Y, Hao J K, Glover F, Lü Z, Wu Q. Solving the maximum vertex weight clique problem via binary quadratic programming. Journal of Combinatorial Optimization, 2016, 32(2): 531–549
42. Xu K, Boussemart F, Hemery F, Lecoutre C. Random constraint satisfaction: easy generation of hard (satisfiable) instances. Artificial Intelligence, 2007, 171(8–9): 514–534
43. Cai S, Su K, Luo C, Sattar A. NuMVC: an efficient local search algorithm for minimum vertex cover. Journal of Artificial Intelligence Research, 2013, 46: 687–716
44. Wu Q, Hao J K. A review on algorithms for maximum clique problems. European Journal of Operational Research, 2015, 242(3): 693–709
45. Rossi R A, Ahmed N K. The network data repository with interactive graph analytics and visualization. In: Proceedings of the 49th AAAI Conference on Artificial Intelligence. 2015, 4292–4293
46. Cai S. Balance between complexity and quality: local search for minimum vertex cover in massive graphs. In: Proceedings of the 24th International Conference on Artificial Intelligence. 2015, 747–753
47. Wang Y, Cai S, Yin M. Two efficient local search algorithms for maximum weight clique problem. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence. 2016, 805–811
48. López-Ibáñez M, Dubois-Lacoste J, Cáceres L P, Birattari M, Stützle T. The irace package: iterated racing for automatic algorithm configuration. Operations Research Perspectives, 2016, 3: 43–58
49. Friedman M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the American Statistical Association, 1937, 32(200): 675–701
50. Garcia S, Herrera F. An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. Journal of Machine Learning Research, 2008, 9(12): 2677–2694

51. Luo C, Cai S, Wu W, Su K. Double configuration checking in stochastic local search for satisfiability. In: Proceedings of the 28th AAAI Conference on Artificial Intelligence. 2014, 2703–2709
52. Luo C, Cai S, Wu W, Jie Z, Su K. CCLS: an efficient local search algorithm for weighted maximum satisfiability. IEEE Transactions on Computers, 2015, 64(7): 1830–1843
53. Luo C, Cai S, Su K, Huang W. CCEHC: an efficient local search algorithm for weighted partial maximum satisfiability. Artificial Intelligence, 2017, 243: 26–44
54. Liu X, Liang J, Liu D Y, Chen R, Yuan S M. Weapon-target assignment in unreliable peer-to-peer architecture based on adapted artificial bee colony algorithm. Frontiers of Computer Science, 2022, 16(1): 161103
55. Qian C, Shi J C, Tang K, Zhou Z H. Constrained monotone *k*-submodular function maximization using multiobjective evolutionary algorithms with theoretical guarantee. IEEE Transactions on Evolutionary Computation, 2018, 22(4): 595–608
56. Luo C, Hoos H H, Cai S, Lin Q, Zhang H, Zhang D. Local search with efficient automatic configuration for minimum vertex cover. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence. 2019, 1297–1304
57. Lei Z, Cai S, Luo C, Hoos H. Efficient local search for pseudo Boolean optimization. In: Proceedings of the 24th International Conference on Theory and Applications of Satisfiability Testing. 2021, 332–348

Shiwei Pan received the BS and MS degrees from the Department of Computer Science and Technology, Northeast Normal University, China in 2018 and 2021. His current research interests include heuristic search and combinatorial optimization.

Yiming Ma studied at the School of Computer Science and Information Technology, Northeast Normal University, China, with a master's degree in Computer Science, and the research direction is heuristic search, local search, algorithmic design, and combinatorial optimization.

Yiyuan Wang is Associate Professor at School of Computer Science and Information Technology, Northeast Normal University, China. He received his PhD degree from Jilin University, China. His research interests include heuristic search, local search, algorithmic design, and combinatorial optimization.

Zhiguo Zhou is Associate Professor of Northeast Normal University and received the PhD degree from the College of Computer Science and Technology, Jilin University, China in 2008. His current research interests include algorithm design and analysis.

Jinchao Ji is a Lecturer at School of Information Science and Technology, Northeast Normal University, China. He received his MS and PhD degrees in Computer Application Technology from Jilin University, China in 2010 and 2013, respectively. His research interests include machine learning, data mining, and artificial intelligence.

Minghao Yin is Professor at School of Computer Science and Information Technology, Northeast Normal University, China. He received his PhD degree in Computer Software and Theory from Jilin University, China. His research interests include heuristic search, data mining, and combinatorial optimization.

Shuli Hu is a Lecturer at Northeast Normal University, China. She received the PhD degree from the Department of Computer Science and Technology, Northeast Normal University, China in 2019. Her current research interests include heuristic search and combinatorial optimization.