Expert Systems With Applications 209 (2022) 118077

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Short communication

A memetic algorithm based on edge-state learning for max-cut

Zhi-zhong Zeng [a], Zhi-peng lü [b,*], Xin-guo Yu [a], Qing-hua Wu [c], Yang Wang [d], Zhou Zhou [a]

[a] *National Engineering Research Center for E-Learning, Faculty of Artificial Intelligence in Education, Central China Normal University, Wuhan, 430079, China*
[b] *School of Computer Science and Technology, Huazhong University of Science and Technology, 430074, China*
[c] *School of Management, Huazhong University of Science and Technology, 430074, China*
[d] *School of Management, Northwestern Polytechnical University, Xi'an, 710072, China*

ARTICLE INFO

Keywords:
Max-cut
Combinatory optimization
Metaheuristics
Memetic algorithm
EDA algorithms
Path relinking

ABSTRACT

Max-cut is one of the most classic NP-hard combinatorial optimization problems. The symmetry nature of it leads to special difficulty in extracting meaningful configuration information for learning; none of the state-of-the-art algorithms has employed any learning operators. This paper proposes an original learning method for max-cut, namely *post-flip edge-state learning* (PF-ESL). Different from previous algorithms, PF-ESL regards edge-states (cut or not cut) rather than vertex-positions as the critical information of a configuration, and extracts their statistics over a population for learning. It is based on following observations. 1) Edges are the only factors considered by the objective function. 2) Edge-states keep invariant when rotating a local configuration to its symmetry position, but vertex-positions do not. These suggest that edge-states contain more meaningful information about a configuration than vertex-positions do. It is impossible to set the state of an edge without influencing some other edges' states due to their dependencies. Therefore, instead of setting edge-states directly, PF-ESL samples the flips on vertices. Flips on vertices are sampled according to their capacities in increasing the similarity on edge-states between the given solution and a population. PF-ESL is employed in an EDA (Estimation of Distribution Algorithm) perturbation operator and a path-relinking operator. Experimental results show that our algorithm is competitive, and show that edge-state learning is value-added for both the two operators.

The main contributions of this paper are as follows. Firstly, previous state-of-the-art evolutionary algorithms for max-cut focus on vertex positions in their evolutionary operation, this paper proposes a new and more reasonable perspective suggesting that edge-states are the critical information of divided graphs rather than vertex positions, and introduces a novel method to measure and utilize their similarities based on it. Such a perspective is fundamental to learning based algorithms design for max-cut and other graph partitioning problems, and can shed lights on future researches. Furthermore, since max-cut is one of the most classic and fundamental NP hard problems, many real-world problems involve dividing graph data into different parts to optimize certain functions, this new perspective may inspire related or similar problems. Secondly, besides the original edge-states based perspective, and the post-flip edge-states learning (PFESL) operator based on it, our memetic algorithm also incorporates a novel evolutionary framework which alternates between EDA based Iterated Tabu search (ITS) and path relinking based genetic algorithm. Finally, the proposed algorithm provides competitive results on two mostly used benchmark sets and improves the best-known results of 6 most challenging instances.

1. Introduction

Combinatorial optimization is a kind of mathematical optimization, it is aimed to find an optimal solution of a problem within its finite and discrete solution space subject to a given objective function. Some of the combinatorial optimization problems, such as most of the NP-hard problems, are well-known to be extremely difficult to solve. Therefore, researchers often resort to heuristic algorithms to find best possible solutions of them with reasonable resources. As one of the most classic NP-hard problems (Karp, 1972), max-cut involves dividing the vertices of a graph into two disjoint subsets to maximize the weighted sum of the edges crossing the two subsets. Solutions to

* Corresponding author.
E-mail addresses: zzzeng@mail.ccnu.edu.cn (Z.-z. Zeng), zhipeng.lv@hust.edu.cn (Z.-p. lü), xgyu@mail.ccnu.edu.cn (X.-g. Yu), qinghuawu1005@gmail.com (Q.-h. Wu), yangw@nwpu.edu.cn (Y. Wang), zz099@mails.ccnu.edu.cn (Z. Zhou).

https://doi.org/10.1016/j.eswa.2022.118077
Received 26 May 2020; Received in revised form 30 May 2022; Accepted 3 July 2022
Available online 22 July 2022

this problem and its close variations (e.g., min-cut, min–max-cut, k-max-cut, etc.) can be used in data mining (Ding et al., 2001), cloud computing (Bansal et al., 2014), VLSI layout (Cho et al., 1998), wireless communication-frequency scheduling (Eisenblätter, 2002), football team scheduling (Mitchell, 2003), and statistical physics (Liers et al., 2004). On the other hand, methods to solve such classic and extremely challenging problems can inspire researchers on many other similar or related problems. The problem can be formulated as follows:

Given an undirected graph $G = (V, E)$, $|V| = n$ and $|E| = m$. V is the set of vertices, $E \subseteq V \times V$ is the set of edges. Each edge $e(i,j) \in E$ connects two different vertices v_i and v_j, and is associated with a weight w_{ij}. The problem is to divide the vertex set V into two disjoint subsets V_0 and V_1, in order to maximize the sum of the weights of edges connecting vertices belonging to different subsets. The objective function of the problem is as follows:

$$f(G) = \max \sum_{i \in V_0, j \in V_1} w_{ij} \tag{1}$$

A solution S to a max-cut problem can be represented as a vector $VP_S = (vp_1, vp_2, \ldots\ldots, vp_n)$, vp_k $(1 \le k \le n)$ represents the position of the kth vertex, i.e., which subset (V_0 or V_1) it belongs to. Due to the NP-Complete nature of max-cut (Karp, 1972), exact algorithms are only applicable to some small or special structured instances so far (Della Croce et al., 2007; Kneis & Rossmanith, 2005). As for large-scale and general instances, it is intractable to solve them exactly up to now. Therefore, researchers usually resort to approximation algorithms and heuristic algorithms. The approximation algorithms use relaxation methods (Goemans & Williamson, 1995; Krishnan & Mitchell, 2006) to provide good approximate solutions to max-cut. They often can guarantee the lower-bound of their algorithms, but they usually cannot apply to large scale instances efficiently (Burer et al., 2002). Although heuristic algorithms often are inefficient for special cases such like graphs mainly consists of cycles or Hamilton paths, and cannot guarantee the lower-bound, they can provide high quality approximate solutions with reasonable time for large-scale general cases. The heuristic algorithms can be further divided into two classes: the specialized heuristic algorithms and the metaheuristic algorithms. The specialized heuristic algorithms are customized for the max-cut problem, such as the rank-two relaxation heuristic (Burer et al., 2002) and the global equilibrium search algorithms (Shylo et al., 2015, 2012). The metaheuristic algorithms first employ powerful general-purpose metaheuristic frameworks, then design the elements of the metaheuristic based on their insights into the max-cut problem. These elements include neighborhoods (moves), perturbation method, evolutionary method, and so on. They also often integrate several metaheuristics together to better utilize their distinct features regarding intensification and diversification. The metaheuristic algorithms include the local search algorithms based on Tabu search (Arráiz & Olivo, 2009; Kochenberger et al., 2013), Variable Neighborhood search (Festa et al., 2002), and Simulated Annealing (Arráiz & Olivo, 2009); the evolutionary or memetic algorithms based on scatter search (Martí et al., 2009), GRASP (Festa et al., 2002; Wu et al., 2015), and path relinking (Festa et al., 2002; Wang et al., 2012); iterated local search based algorithms (Benlic & Hao, 2013; Ma & Hao, 2017; Ma et al., 2017); etc. Recently, a two-phase metaheuristic framework has become a research focus. It includes an intensification phase and a diversification phase. It uses sophisticate local search metaheuristic such as Tabu search or (and) Variable Neighborhood search to find good local-optimum solution within their intensification phase, and uses metaheuristics such as iterated local search, grasp, or path-relinking to construct diversified and promising initial solutions for the intensification phase during the diversification phase. The whole process alternates between these two phases repeatedly. Many of such algorithms have shown great effectiveness (Ma & Hao, 2017; Ma et al., 2017; Wu et al., 2015) since they can utilize the complementary search capacity of the two phases properly. Based on its prominent feature, we also adopt this

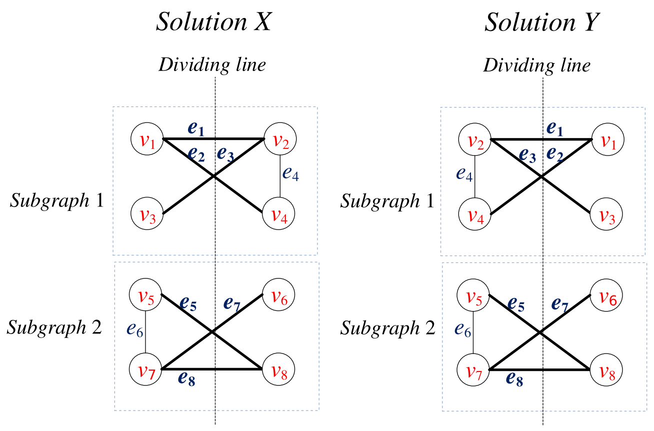


Fig. 1. An example on local configuration symmetry issue. The cutting edges are bolded. Although solution X and Y are quite different with regard to vertex positions, they have same cutting edges. Therefore, they are exactly one solution from the perspective of the objective function. This confliction is caused by the symmetry of local configurations. If one rotates the sub-graph 1 of solution Y around the dividing line to its mirror position, then all the vertex positions of solution X and Y will be the same.

iterative two-phase framework in our metaheuristic-based algorithm. Although previous two-phase metaheuristic algorithms have provided state-of-the-art results on max-cut, none of them has employed any learning-based methods in their diversification phase, which have already shown great effectiveness on many other optimization problems regarding providing best solution within reasonable time (Fu & Hao, 2017; Khalil et al., 2017; Zhou et al., 2018; Zhou & Hao, 2017). This might be caused by the difficulties in designing effective learning operators for the max-cut problem: previous learning methods usually utilize the statistics on element positions, but as illustrated by the following example, one cannot use such statistics of max-cut solutions reasonably and easily.

Learning based methods such as Estimation of Distribution Algorithms (EDAs) (Larrañaga & Lozano, 2001) and Opposite Based Learning (OBL) (Tizhoosh, 2005) have been a hot research area of evolutionary computation since the beginning of this century. Generally, they use the statistic information of a population or previous searches to construct promising new solutions or guide future searches. There are two key issues involved in designing effective learning operators: what to learn and how to learn. What to learn prescribes what statistic information one should extract from a population; how to learn determines how to construct or partly reconstruct a solution according to these statistic information. A solution to a problem is often represented as a vector, and learning operators usually make use of the statistic information on each component of the vector independently (Fu & Hao, 2017; Zhou et al., 2018). Therefore, which kind of elements one used to represent a solution can well-define what statistic information to learn. As described above, the max-cut problem has two kinds of elements: vertices and edges. Previous state-of-the-art algorithms use the positions of vertices to represent a solution and whereupon measure the similarity between two solutions for crossover (Wang et al., 2012; Wu et al., 2015). We think it may be not an effective choice in some cases. Consider the example in Fig. 1, the vertices of solutions X and Y are divided into two sets by two dividing lines respectively. Both solutions X and Y are cutting their edges e_1, e_2, e_3, e_5, e_7 and e_8 (the bolded edges in Fig. 1). They are the same with regard to cutting edges, which are the only things calculated by the objective function (formula (1)). However, they are quite different according to their vertex positions. They cannot be encoded into one representation according to their vertex positions. Specifically, solution X can be encoded as 01010101 or 10101010 depending on which vertex set we choose as V_0. Similarly, solution Y can be encoded as 10100101 or 01011010. Solution X and

Y are quite different no matter which encoding results one chooses. This situation is caused by the symmetry of local configurations. If one rotates the local configuration on subgraph 1 in solution Y around the dividing line to its mirror position, then all the vertex positions of solution X and Y will be the same. One can observe that vertex positions are not a good criterion for measuring the similarity between two solutions in this case, and sometimes they cannot provide critical information on what a solution really is. In many real-world applications, graphs consist of high-coherent, low-coupling subgraphs. In these cases, solutions can suffer similar local-configuration symmetry issues, and the statistics on vertex positions over a population will not make any sense. This difficulty may be the reason why learning operators are not employed in any previous state-of-the-art algorithms.

To deal with this issue, we use the edge-state based encoding method for learning. It encodes a solution by its edge states rather than vertex positions. A solution S hence can be represented in another form: $ES_S = (es_1, es_2, \ldots\ldots, es_m)$, es_k $(1 \le k \le m)$ represents the state of its kth edge: if it is a cutting edge, it is set to 1; otherwise 0. By this means, solution X and solution Y are both encoded into the same representation: 11101011. This suggests that edge-states may be a more proper criterion to measure the similarity between two solutions, and they can provide more meaningful information about a solution than vertex positions do. Therefore, the statistic information on edge states may be more relevant than that on vertex positions for such structural cases.

Although the edge-state encoding is more meaningful on such structural instances, it also has posed a challenge to designing effective learning operators. Unlike vertex positions, one cannot set the state of an edge without influencing the states of some other edges in most occasions. For example, in Fig. 1, if one switches the state of e_1 from its current cutting state to non-cutting state, it will also change the states of edges $\{e_2\}$ or $\{e_3, e_4\}$. Due to this dependency issue, one cannot set the states of edges one by one independently just like some other learning operators (Fu & Hao, 2017; Zhou et al., 2018) perform on their solution components. To overcome this difficulty, we propose a method named *post-flip edge-state learning* (PF-ESL). Rather than setting edge states directly, it samples the flips on vertices. The flip on a vertex k is sampled according to its capacity in increasing the similarity on edge states between the solution and a population. By iteratively sampling such flips, the solution will become more and more similar to the population with regard to edge states.

The main contributions of this paper are as follows. Firstly, previous state-of-the-art evolutionary algorithms for max-cut focus on vertex positions in their evolutionary operation, this paper proposes a new and more reasonable perspective suggesting that edge-states are the critical information of divided graphs rather than vertex positions, and introduces a novel method to measure and utilize their similarities based on it. Such a perspective is fundamental to learning based algorithms design for max-cut and other graph partitioning problems, and can shed lights on future researches. Furthermore, since max-cut is one of the most classic and fundamental NP hard problems, many real-world problems involve dividing graph data into different parts to optimize certain functions, this new perspective may inspire related or similar problems. Secondly, besides the original edge-states based perspective, and the post-flip edge-states learning (PFESL) operator based on it, our memetic algorithm also incorporates a novel evolutionary framework which alternates between EDA based Iterated Tabu search (ITS) and path relinking based genetic algorithm. Finally, the proposed algorithm provides competitive results on two mostly used benchmark sets and improves the best-known results of 6 most challenging instances.

The rest of our paper is organized as follows. Section 2 presents a detailed description on the proposed algorithm. Section 3 provides the experimental results of our algorithm comparing with other state-of-the-art algorithms, and analyzes the key features of our algorithm. Finally, Section 4 concludes the paper.

2. Solution method

This section describes our solution in detail. Section 2.1 first provides the main framework of our solution method; the other subsection then describes its components in detail. We also list all the variables used in the solution method and their descriptions as follows for reference:

P: Sections 2.1–2.2; the population.

p: Sections 2.1–2.2; the size of the population.

ζ: Section 2.1; a constant value prescribes the possibility of using Path_relinking to construct a new solution from the population at each iteration.

Policy_flag: Section 2.1; a temporary variable, its value is generated from range (0, 1) at each iteration randomly. It is used to compare with ζ to determine whether to use Path_Relinking or EDA procedure to construct a new solution.

Pop_stats: Sections 2.1–2.3; the statistics on edge states, it records the ratio of each edge that being "cut" within the population, i.e., *Pop_stats* $[i]$ = (number of solutions in the population that cut edge i) / (number of solutions in the population).

η: Section 2.3.1; perturbation strength of EDA operator, it defines the number of flips (move a vertex from its current vertex set to the other vertex set) will be performed in the EDA operator.

d_{jm}: Section 2.3.1; the distance between the two input solutions S_j and S_m, it measures at least how many vertices one needs to flip on solution S_j to make the two input solutions identical.

μ: Section 2.3.2; Path-relinking length, it defines the number of flips will be performed in the Path-relinking operator.

Current solution: the solution to perform current one-flip or edge-swap move.

The Δ value array: Section 2.4.1; the move-gain (increase of the objective function value after a move) of performing each one-flip move on current solution.

The δ value array: Section 2.4.2; the move-gain of performing each edge-swap move on current solution.

Current best solution: the best solution ever found so far during the execution of the Tabu search procedure.

tt: Section 2.4; Tabu tenure, a constant prescribes how long a vertex will be forbidden to be involved in any future moves after it has been involved in current move.

ρ: Section 2.4; a constant prescribes the possibility to employ *edge-swap* move when local search get stuck at a local optimum.

ic: Section 2.4; a constant prescribes the improvement cutoff of TS; i.e., how many consecutive iterations without improvement is allowed in Tabu search. If the number of consecutive iterations without improvement exceeds this value, the Tabu search procedure ends.

2.1. Main scheme

Our PF-ESL memetic algorithm follows the two-phase (intensification and diversification) framework mentioned above. It is composed of four components: population initialization, the hybrid reconstruction procedure, the Tabu search procedure, and the population-updating operator. The Tabu search procedure corresponds to the intensification phase, the hybrid reconstruction procedure and the population-updating operator form the diversification phase. At first, the first component initializes the population for the rest of the algorithm. The other three components then form a key repeated process alternating between intensification phase (the third component) and the diversification phase (the second and the fourth component): at each iteration of the repeated process, the hybrid reconstruction procedure generates a promising initial solution based on the population (diversification phase), which is further optimized by the Tabu search procedure (intensification phase); the population-updating operator then uses the result of the Tabu search procedure to update the population (diversification phase).

The variables involved in the algorithm are described as follows:

P: the population.

p: the size of the population.

ζ: a constant value prescribes the possibility of using Path_relinking to construct a new solution from the population at each iteration.

Policy_flag: A temporary variable, its value is generated from range (0, 1) at each iteration randomly. It is used to compare with ζ to determine whether to use Path_Relinking or EDA procedure to construct a new solution.

Pop_stats: the statistics on edge states, it records the ratio of each edge that being "cut" within the population, i.e., *Pop_stats*[i]= (number of solutions in the population that cut edge i) / (number of solutions in the population).

Algorithm 1: PF-ESL memetic algorithm.

Input: the size of the population (p); the number of vertices n; the $n \times n$ matrix of edge weights.

Output: the best solution ever found.

```
1: /*Lines 2–7 initialize the population (Population initialization, component 1), section 2.2*/
2: $P=\{S_1,..., S_p\} \leftarrow$ randomly generate $p$ initial solutions as the population /* section 2.2 */
3: for $i$ in $\{1,..., p\}$ do
4:    $S_i \leftarrow Tabu\_Search(S_i)$ /* Tabu Search procedure, section 2.4 */
5: end for
6: $Pop\_stats$ $\leftarrow$ Calculate the statistics on edge states over the population /* section 2.2 */
7: Mark each solution pair $(S_i, S_j)$ $(i \neq j,\ i, j \in [1, p])$ as linkable /* section 2.2 */
8: /*Lines 9–24 form the key repeated process (component 2 - 4)*/
9: repeat
10:    /*Lines 11–21 belong to the hybrid reconstruction procedure (component 2), section 2.3 */
11:    $Policy\_flag \leftarrow$ randomly chooses a value from (0, 1)
12:    if $Policy\_flag < \zeta$ and there are at least one pair of linkable solutions then
13:       /* the path relinking operator is activated*/
14:       Choose a linkable solution pair $(S_j, S_m)$ which have largest difference
15:       $S_0 \leftarrow Path\_Relinking\ (S_j, S_m)$ /* Section 2.3.2 */
16:       Mark $(S_j, S_m)$ as unlinkable
17:    else
18:       /* The EDA operator is activated */
19:       Randomly choose an individual $S_j$ from $P$
20:       $S_0 \leftarrow$ EDA $(S_j, Pop\_stats)$ /* Section 2.3.1 */
21:    end if
22:    $S_0 \leftarrow$ Tabu_Search $(S_0)$ /*Tabu Search procedure (component 3), section 2.4 */
23:    $\{S_1,...,S_p\} \leftarrow$ Population_Updating$(S_1,..., S_p, j, S_0, Policy\_flag, Pop\_stats)$ /* component 4, Section 2.5 */
24: until a stop criterion is met
```

Algorithm 1 provides the pseudo code of the algorithm. In the beginning, the first component initializes the population randomly, uses Tabu search to optimize its individuals one by one, calculate the statistics on edge states over the population, and mark each two different solutions as linkable (lines 2–7). Then the key repeated process (lines 9–24) starts. The repeated process begins with the hybrid reconstruction procedure (lines 11–21). It chooses the PF-ESL based path-relinking operator or the PF-ESL based EDA operator randomly according to a prescribed probability ζ. The path-relinking operator chooses two linkable solutions S_j and S_m that are most different with regard to edge states from the population (line 14), then combines them based on their edge states to generate an offspring solution S_0 (line 15). The EDA operator randomly chooses an individual S_j from the population (line 19), and then reconstructs it partly to generate an offspring solution S_0 according to the statistic information of the population on edge-states (line 20). When the offspring solution S_0 is generated, the Tabu search procedure is employed to optimize it (line 22). Finally, Population-Updating operator updates the population according to the quality of the optimized offspring and the reconstruction policy used (line 23).

The EDA operator (line 20) is a kind of perturbation operator. Hence, the whole algorithm can be regarded as a hybrid of a path relinking based genetic algorithm and an EDA based Iterated Tabu search. We will describe the four components of our algorithm sequentially in the following.

2.2. Population initialization

Population initialization is the first component of our algorithm. It prepares the initial population and related variables for the following key repeated process. It first generates p solutions randomly. For each solution, each of its n vertices is assigned to V_0 or V_1 randomly. After the p solutions are generated, they are further optimized by the Tabu search procedure one by one. Subsequently, the initialization procedure calculates the statistics on edge states over the initial population, i.e., for each edge, how many solutions in the population are now cutting it. Finally, each pair of solutions is initialized as linkable. Only two linkable solutions can be used by the path-relinking operator to generate an offspring. Notice that for two solutions S_i and S_j, (S_i, S_j) and (S_j, S_i) are regarded as two different solution pairs.

2.3. The hybrid reconstruction procedure

The hybrid reconstruction procedure is the second component of our algorithm. It is the key component of our algorithm. It generates a promising initial solution for the following Tabu search procedure to further optimize. To make the generated solution more promising, it needs to extract meaningful information out of the population, and then make effective use of it. Previous algorithms use the information on vertex positions to construct an offspring solution. They use the positions of vertices as the "genes" of two parent solutions, and then crossover them to construct an offspring accordingly (Wang et al., 2012; Wu et al., 2015). Based on the importance of edge states in the objective function, and the observation that edge states are immune to local configuration symmetry issue, we think the information on edge states is more meaningful than that on vertex positions. Hence, our hybrid reconstruction procedure regards edge states as the genes of two parent solutions for path relinking, and makes statistics on edge states over the population for EDA based learning. In the following, we first introduce our EDA operator that learns from the population. Our path relinking operator is in fact a special EDA operator that learns from a temporary population with just one solution (a parent solution).

2.3.1. EDA operator

The EDA operator uses the statistic information on edge states over an input population to partly reconstruct an input solution. It repeatedly samples the flips on vertices. The flip on a vertex k is sampled according to its capacity in increasing the similarity between the input solution and the population with regard to edge states, which is measured by the total or average *post-flip-statistic* of k. In the following, we will first introduce the concepts *post-flip-statistic*, total *post-flip-statistic*, and average *post-flip-statistic* sequentially. Then we will describe our EDA operator.

Definition 1 (*Post-Flip-Statistic*)**.** Given a vertex k of solution S and a population P, the post-flip-statistic on edge $e(k, t)$ is the ratio of solutions in population P whose edges $e(k, t)$ are in the same state as the edge $e(k, t)$ of S is after flipping vertex k of S. Represented as post-flip-statistic$_{e(k,t)}(k, S, P)$.

For example, let solution X in Fig. 1 in a population P with 10% of its solutions cutting edge e_1, 65% cutting edge e_2. Then if one flips v_1, e_1 and e_2 will both switch from cutting state into non-cutting state. After flipping vertex v_1 of X, there are 1%–10% = 90% solutions within the population whose edge e_1 are in the same state as the edge e_1 of solution X is, and 1%–65% = 35% solutions within the population whose edge e_2 are in the same state as the edge e_2 of solution X is. So *post-flip-statistic*$_{e_1}(v_1, X, P) = 0.9$, *post-flip-statistic*$_{e_2}(v_1, X, P) = 0.35$. *Post-flip-statistic*$_{e(k,t)}(k, S, P)$ measures how similar to a population P an edge $e(k, t)$ in solution S will become after flipping vertex k. After flipping a vertex, usually more than one edge will change their states; to evaluate a flip, we need to measure the total *post-flip-statistic* on all the edges connected to the flipped vertex.

Definition 2 (*Total Post-Flip-Statistic*)**.** Given a vertex k of solution S and a population P, the total *post-flip-statistic* of k is the weighted sum of the *post-flip-statistics* on all the edges connected to k. Represented as *post-flip-statistic*$_\Sigma$ (k, S, P):

$$post\text{-}flip\text{-}statistic_{\Sigma}(k,S,P) = \sum_t |w_{kt}|\, post-flip-statistic_{e(k,t)}(k,S,P)$$

For example, in Fig. 1, if e_1 has a weight of 2, and e_2 has a weight of -1, then *post-flip-statistic*$_\Sigma(v_1, X, P) = |2| * 0.9 + |-1| * 0.35 = 2.15$. The edges connected to a vertex k can be regarded as a local configuration. Therefore, *post-flip-statistic*$_\Sigma(k, S, P)$ measures how similar to population P the local configuration connected to vertex k will become after flipping vertex k. Notice that dense vertices that have many edges connected to them are prone to have larger total *post-flip-statistics*, even if each edge connected to it only has a small *post-flip-statistic*. Therefore, we present another concept average *post-flip-statistic* to eliminate the influences of vertex degrees.

Definition 3 (*Average Post-Flip-Statistic*)**.** Given a vertex k of solution S and a population P, the average *post-flip-statistic* of k is the weighted average value of the *post-flip-statistics* on all the edges connected to k:

$$\overline{post\text{-}flip\text{-}statistic}\,(k,S,P) = \frac{post\text{-}flip\text{-}statistic_{\Sigma}(k,S,P)}{\sum_t |w_{kt}|}$$

For example, in Fig. 1, $\overline{post\text{-}flip\text{-}statistic}\left(v_1, X, P\right) = post\text{-}flip\text{-}statistic_{\Sigma}\left(v_1, X, P\right) / (|2| + |-1|) = 0.7167$.

$\overline{post\text{-}flip\text{-}statistic}\,(k,S,P)$ measures how similar to population P the local configuration connected to vertex k will become on the average after flipping vertex k.

At each iteration, the EDA operator samples a vertex k from S_j according to following probability distribution, and then flips it:

$$P(k) \propto post\text{-}flip\text{-}statistic_{\Sigma}\left(k, S_j, P\right) \tag{2}$$

The sampling process is repeated for η times, η is a parameter that defines the number of vertices to be flipped in EDA. We use total $post-flip-statistic$ rather than average $post-flip-statistic$ in the sampling process, because total $post-flip-statistic$ will give more priority to dense vertices. Flipping such a vertex can make more edges to become similar to the population at one time. What is more, dense vertices can be regarded as critical elements, Lü and Hao (Lü & Hao, 2009) revealed that changing the states of the critical elements in the random perturbation process of Iterated Tabu search can provide better results. The pseudo-code is provided in Algorithm 2.

Algorithm 2: The EDA procedure.

Input: The input solution S that needs to be transformed; The statistics records that what is the ratio within the population that each edge is cut
Output: The transformed solution of S

```
1: for $i = 1$ to $\eta$ /* $\eta$ is a predefined parameter */ do
2:     sample a vertex $k$ according to the possibility post-flip-statistic$_\Sigma(k, S, Pop\_stats)$
3:     move vertex $k$ in $S$ from its current vertex set to the other vertex set
4: end for
5: return $S$
```

2.3.2. Path relinking

The path relinking operator first determines the distance d_{jm} between the two input solutions S_j and S_m. The distance measures at least how many vertices one needs to flip to make the two input solutions identical. Due to the symmetry nature of the problem, the number will never exceed $|V|/2$. After the distance d_{jm} is calculated, path relinking then determines the number μ of vertices to be flipped accordingly. μ is randomly chosen from $[d_{jm}/3, 2d_{jm}/3]$ in our implementation. Finally, it samples and flips the μ vertices from S_j one by one according to the following probability distribution:

$$P(k) \propto \overline{post\text{-}flip\text{-}statistic}\left(k, S_j, \{S_m\}\right) \tag{3}$$

The pseudo-code is provided in Algorithm 3.

Algorithm 3: The Path-Relinking procedure.

Input: The input solution S_j that needs to be transformed; solution S_m is used as the reference of the transformation
Output: The transformed solution of S_j

```
1: determines the distance $d_{jm}$ between the two input solutions $S_j$ and $S_m$
2: $\eta \leftarrow$ random choose an integer with in $[d_{jm}/3, d_{jm}/3]$
3: for $i = 1$ to $\eta$ do
4:     sample a vertex $k$ according to the possibility $\overline{post\text{-}flip\text{-}statistic}(k, S_j, \{S_m\})$
5:     move vertex $k$ in $S_j$ from its current vertex set to the other vertex set
6: end for
7: return $S_j$
```

2.4. Tabu search

After the offspring solution is generated, the Tabu Search procedure further optimizes the offspring by means of intensive exploration. Tabu Search is a powerful local search metaheuristic (Glover & Laguna, 1998). Our Tabu Search procedure is a simple adaptation of the Tabu Search procedure proposed in (Ma et al., 2017) to max-cut. Specifically, it uses the one-flip move in the best improvement search sub procedure. When getting stuck at a local optimum, it samples one-flip move and edge-swap move according to a predefined probability ρ. The Tabu search procedure ends when no improvement can be obtained within ic iterations. ρ and ic are predefined constants. We first introduce the one-flip move and edge-swap move respectively, then analyze the time complexity of a Tabu search iteration, and finally provide the pseudo-code of the Tabu search procedure.

2.4.1. One-flip move

Given a solution $S = \{V_0, V_1\}$, one-flip move moves a vertex v_i from its current vertex set (e.g., V_0) to the other (e.g., V_1). For each Tabu search iteration, there are $|V|$ possible one-flip moves. The Tabu search procedure always chooses the best not tabued move at each iteration. The chosen move is marked as tabued and will not be chosen again for tt iterations to avoid endless loops. The tabued status of a one-flip move is ignored if it can lead to a solution better than the best solution ever found during the current execution of the Tabu search procedure, this is the so-called aspiration rule. If several untabued one-flip moves can lead to solutions with the same objective function value, it always chooses the one that has been chosen most recently. The move-gain of a one-flip move represents the increase in the objective function value after a move is performed. To accelerate the calculation of move-gains, we use the technique first proposed in the unconstrained binary quadratic optimization problem (Glover & Hao, 2010). Specifically, the Tabu search procedure maintains an array Δ representing the move-gain for each one-flip move. The array is initialized as follows:

$$\Delta_i = \begin{cases} \sum_{j\in v_0} w_{ij} - \sum_{j\in v_1} w_{ij} & (i \in V_0) \\ \sum_{j\in v_1} w_{ij} - \sum_{j\in v_0} w_{ij} & (i \in V_1) \end{cases} \tag{4}$$

After performing the one-flip move on a vertex i (moving vertex i from its current vertex set to the other vertex set), the array is updated as follows (function vertexset() returns the vertex set that a vertex belongs to):

$$\Delta_k = \begin{cases} -\Delta_k & (k = i) \\ \Delta_k - 2w_{ik} & (k \neq i, vertexset(k) = vertexset(i)) \\ \Delta_k + 2w_{ik} & (k \neq i, vertexset(k) \neq vertexset(i)) \end{cases} \tag{5}$$

All the neighboring vertices of a vertex i (the vertices that sharing edges with i) can be stored in a list. When vertex i is flipped, only the vertices stored in its list need to update their delta values. Therefore, the averaging updating time is only $O(|E|/|V|)$. In most cases, the graph is sparse, and $|E|/|V|$ is a small value, the updating process can be very efficient.

Special data structures, such as bucket-sorting can be used to accelerate the computing process of max-cut problem. For each possible Δ value, bucket-sorting holds two doubly-linked list, one stores all the untabued vertices that have such one-flip Δ value, the other stores all the tabued vertices that have such one-flip Δ value. There is also an array with each of its element corresponds to such a possible Δ value and points to its corresponding two doubly linked lists. Two variables indicate the current largest Δ value of all the untabued vertices, and the current largest Δ value of all the tabued vertices respectively. Fig. 2 provides an example of the array of doubly-linked lists in bucket-sorting. The value stored in each array element is the corresponding Δ value, and the first doubly-linked list stores all the untabued vertices that have this one-flip Δ value, and the second doubly-linked list stores all the tabued vertices that have this one-flip Δ value. Variables $Bestuntabued$ and $Besttabued$ indicate the current largest Δ values of all the untabued vertices and tabued vertices respectively. By means of such data structure, in each iteration, the time for the Tabu search procedure to retrieve the desired vertex to perform one-flip move can be reduced to only $O(1)$. The maintenance of the data structure also can be very efficient. Updating the Δ value of a vertex needs to remove it from its current doubly-linked list and insert it at the beginning of the doubly-linked list corresponding to its new Δ value. Both can be processed with the time complexity $O(1)$ by utilizing the data structure efficiently. Therefore, the time complexity of maintain the data structure at each iteration is $O(|E|/|V|)$.

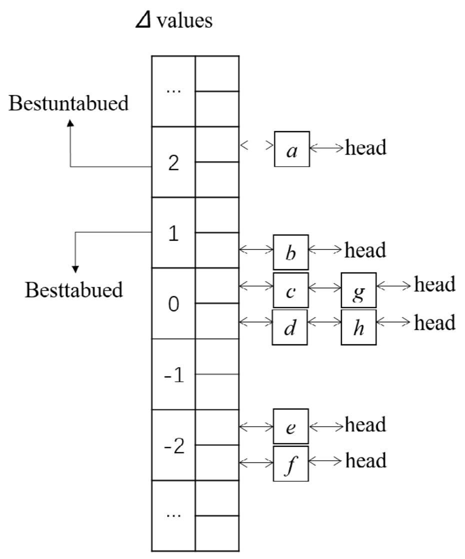


Fig. 2. An example of the array of doubly-linked lists in bucket-sorting.

2.4.2. Edge-swap move

Edge-swap move was proposed by Wu et al. (2015). Given a solution $S = \{V_0, V_1\}$, $G = (V = V_0 \cup V_1, E)$, edge-swap move moves two vertices v_i and v_j ($v_iv_j \in E$, v_i and v_j lie in different vertex set) from their current vertex sets to the other vertex sets respectively. For each Tabu search iteration, there are $|E|$ possible one-flip moves. When employing edge-swap move, the Tabu search procedure always chooses the best not tabued edge-swap move at each iteration. The edge-swap move is only employed when the local search got stuck at a local optimum and is sampled with a low possibility ρ. It can be regarded as perform two separate one-flip move: flipping vertex i and then flipping vertex j. Therefore, the Tabu rules and aspiration rules mentioned above remain the same: (1) an edge-swap move is Tabued if and only if one of the involved vertices is tabued; (2) when an edge-swap move is performed, the involved two vertices are marked as Tabued and are forbidden to be used by any moves (one-flip or edge-swap) within tt iterations; (3) The tabued status of a move (one-flip or edge-swap) is ignored if it can lead to a solution better than the best solution ever found during the current execution of the Tabu search procedure.

The move-gain value δ_{ij} of performing the edge-swap move on edge ij can be calculated as follows (Wu et al., 2015):

$$\delta_{ij} = \Delta_k + \Delta_k + 2w_{ij} \tag{6}$$

After performing the edge-swap move on an edge ij:

$$\Delta_k = \begin{cases} -\Delta_k - 2w_{ij} & (k = i \quad or \quad k = j) \\ \Delta_k - 2w_{ik} + 2w_{jk} & (k \neq i, k \neq j, vertexset(k) = vertexset(i)) \\ \Delta_k + 2w_{ik} - 2w_{jk} & (k \neq i, k \neq j, vertexset(k) = vertexset(j)) \end{cases} \tag{7}$$

We also use the bucket-sorting data structure discussed above to accelerate the calculation for edge-swap move too. It can be easily observed that the time complexity for the Tabu search procedure to retrieve the desired edge to perform edge-swap move is also $O(1)$.

When vertex i is flipped (by means of one-flip move or edge-swap move), not only the Δ values of all its neighboring vertices need to be updated, all the δ values of the edges associated with these neighboring vertices also need to be updated. The average updating time complexity is increased from $O(|E|/|V|)$ to $O(|E|^2/|V|^2)$. The time complexity of maintenance the bucket-sorting data structure is also increased from $O(|E|/|V|)$ to $O(|E|^2/|V|^2)$ due to the same reason.

2.4.3. Time complexity of a tabu search iteration

The time spent in each Tabu search iteration is mainly composed of three parts: first, retrieve the desired vertex or edge; second, update the Δ array and δ array; third, update the data structure. Based on above analysis of the two kinds of Tabu search iteration, the time complexity of a Tabu search iteration is $O(O(1) + O(|E|^2/|V|^2) + O(|E|^2/|V|^2)) = O(|E|^2/|V|^2)$.

2.4.4. Pseudo-code

Now we present the pseudo code of the Tabu search procedure in Algorithm 4. The variables involved in the algorithm are described as follows:

Current solution: the solution to perform current one-flip or edge-swap move.

The Δ value array: The move-gain of performing each one-flip move on current solution.

The δ value array: The move-gain of performing each edge-swap move on current solution.

Current best solution: the best solution ever found so far during the execution of the Tabu search.

tt: Tabu tenure, a constant prescribes how long a vertex will be forbidden to be involved in any future moves after it has been involved in current move.

bucket-sorting data structure: the arrays of doubly linked lists explained above to accelerate the calculation.

$Probabilityvalue$: a temporary variable, its value is randomly chosen from (0, 1) at each iteration, and then compared with the constant ρ explained below to determine which kind of move to be used in current iteration.

ρ: a constant prescribes the possibility to employ edge-swap move when local search get stuck at a local optimum.

ic: a constant prescribes the improvement cutoff of TS; i.e., how many consecutive iterations without improvement is allowed in Tabu search. If the number of consecutive iterations without improvement exceeds this value, the Tabu search procedure ends.

The Tabu search procedure first initializes the current solution and the current best solution using the input solution (line 1 and line 2), then initialize the Δ values, the δ values and the bucket-sorting data structure based on current solution (line 3). Then the main loop (line 5–30) of the Tabu search begins. It alternates between two part: the best-improvement search part and the breakthrough part. The best-improvement search part (line 7–12) uses the one-flip move 2.4.1 in a best improvement search process. When it gets stuck in a local optimum, the breakthrough part (line 14–30) samples one-flip move (line 16–18) and edge swap move (line 19–23, Section 2.4.2) according to a predefined probability ρ. When a further improvement is obtained (line 25), the process switches back to the loop of the best-improvement search (line 27). The whole process ends when no improvement to the current best solution can be obtained within ic consecutive iterations.

2.5. Population updating

After the offspring solution is optimized by Tabu search, population-updating decides whether the offspring solution should be accepted and which solution in the population it should replace. It first checks whether the offspring is the same as one of the solutions in the population. If it is, it will be discarded. If no solution in the population is the same as the offspring, the operator will update the population according to the quality of the offspring and the reconstruction strategy used to generate it. If the employed reconstruction operator is path relinking, it will compare the offspring with the worst solution in the population; if it is better than the worst solution, it will replace the worst solution with the offspring. If the employed reconstruction operator is EDA, population updating will compare the offspring with its parent solution that has generated it; if the offspring is better than its parent solution, population-updating will replace its parent solution with it. Otherwise the offspring will be discarded. As the strategy proposed in (Wang et al., 2012; Wu et al., 2015), once the offspring solution is inserted into the population, all the solutions will be set to be linkable to it and it is set to be linkable to all the other solutions.

3. Experiments and analysis

To assess the proposed PF-ESL based memetic algorithm, we implemented it in Visual C++ language and performed the following experiments. The computer we used to run the experiments is a PC with an AMD FX 8350 CPU (4 GHZ), 4 GB RAM. The operation system is Windows 7, 32 bit. The important parameter setting of our algorithm for the two test sets are listed in Table 1.

Algorithm 4: The Tabu search procedure.

Input: The input solution $\{V_0, V_1\}$ (V_0 and V_1 are the two disjoint vertex sets), the graph $G = (V = V_0 \cup V_1, E)$
Output: The best solution found during the search.

```
1:  $currentsolution \leftarrow$ The input solution
2:  $currentbestsolution \leftarrow$ The input solution
3:  initialize the $\Delta$ values, the $\delta$ values and the bucket-sorting data
    structure based on the current solution /* Sections 2.4.1 and 2.4.2 */
4:  /* main process of the Tabu search */
5:  **repeat**
6:    /* Lines 7-12 uses the one-flip move 2.4.1 in a best improvement
      search process */
7:    **while** the largest $\Delta$ value of current solution is positive **do**
8:      move the vertex $i$ that has the largest $\Delta$ value from its current
        vertex set to the other vertex set (if several untabued one-flip
        moves can lead to solutions with the same objective function
        value, chooses the one that has been chosen most recently)
9:      current best solution $\leftarrow$ current solution
10:     Mark vertex $i$ as Tabued for $tt$ iterations
11:     Update the array of the $\Delta$ values, the $\delta$ values and the
        bucket-sorting data structure accordingly
12:   **end while**
13:   /* when former best-improvement search gets stuck, it samples
      one-flip 2.4.1 move and edge swap move 2.4.2 according to a
      predefined probability $\rho$. It is called the breakthrough part */
14:   **while** (1) **do**
15:     $Probabilityvalue \leftarrow$ randomly choose a value from (0, 1)
16:     **if** Probabilityvalue $> \rho$ **then**
17:       move the not tabued vertex $i$ that has the largest $\Delta$ value
          from its current vertex set to the other vertex set (if several
          not tabued one-flip moves can lead to solutions with the
          same objective function value, chooses the one that has been
          chosen most recently)
18:       Mark vertex $i$ as Tabued for $tt$ iterations
19:     **else**
20:       Find the not tabued edge $ij$ ($v_i$ and $v_j$ should lie in different
          vertex set) that has the largest $\delta$ value
21:       moves the two vertices $v_i$ and $v_j$ from their current vertex
          sets to the other vertex sets respectively
22:       Mark vertex $i$ and $j$ as Tabued for $tt$ iterations
23:     **end if**
24:     Update the array of the $\Delta$ values, the $\delta$ values and the bucket-
        sorting data structure accordingly /* Sections 2.4.1 and 2.4.2 */
25:     **if** current solution is better than the current best solution /*
        has got out from the local optimum */ **then**
26:       Current best solution $\leftarrow$ current solution
27:       **break** /*resume to the best-improvement search*/
28:     **end if**
29:   **end while**
30: **until** the current best solution cannot be improved for $ic$
    consecutive iterations /*stop criteria*/
31: **return** the current best solution
```

The first experiment is aimed to evaluate the "discovery capability" (Grosso et al., 2007) of our algorithm, i.e., the ability to find the best possible results with reasonable time. The G-set and cubic lattices set are the mostly used benchmark sets of max-cut. Many important max-cut algorithms were tested on one or two of these two benchmark sets (Benlic & Hao, 2013; Ma & Hao, 2017; Ma et al., 2017; Shylo

Table 1
Settings of important parameters on the two benchmark sets.

Parameter	Section	Description	Values	
			G-set	Cubic lattice
p	2.1–2.2	Population size	640	160
ζ	2.1	Probability of choosing Path-relinking	0.33	0.33
η	2.3	Perturbation strength of EDA operator	$rand[\lvert V\rvert/6,\dots,\lvert V\rvert/4]$	$rand[200,\dots,400]$
μ	2.3	Path-relinking length	$rand[d_{jm}/3,\dots,2d_{jm}/3]$	$rand[d_{jm}/3,\dots,2d_{jm}/3]$
ρ	2.4	Probability of choosing swap moves in TS	0.15	0.15
tt	2.4	Tabu tenure	$rand[6,\dots,0.2\lvert V\rvert]$	$rand[6,\dots,0.2\lvert V\rvert]$
ic	2.4	Improvement cutoff of TS	10,000	10,000

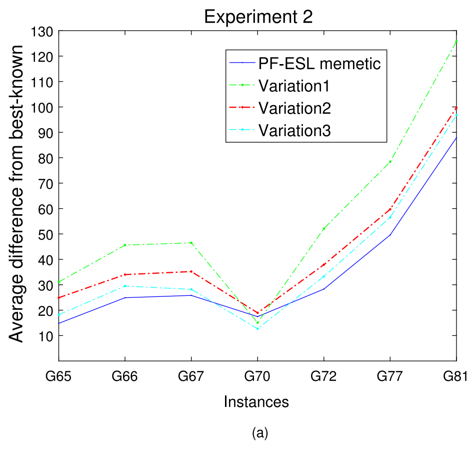
Table 2
Computational comparisons between TSHEA, MOH, and PF-SEL on the largest G-set and cubic lattice instances.

Instance	$\lvert V\rvert$	f_{pre}	TSHEA		MOH		PF-SEL	
			f_{best}	f_{avg} (σ)	f_{best}	f_{avg} (σ)	f_{best}	f_{avg} (σ)
G-set								
G60	7000	14,190	14,186	14,173.50(7.20)	14,190	14,173.00(6.98)	14187	14174.85(5.50)
G61	7000	5798	5796	5776.00(8.95)	5798	5782.67(5.72)	5792	5785.75(4.60)
G62	7000	4868	4866	4860.20(1.66)	4868	4851.73(7.10)	4868	4865.7(1.49)
G63	7000	27,033	27,018	26,993.60(9.14)	27,033	27,019.20(6.23)	26980	26966.5(7.40)
G64	7000	8747	8735	8717.95(9.45)	8747	8700.87(19.28)	8726	8709.2(10.00)
G65	8000	5560	5560	5555.40(3.03)	5560	5531.93(6.43)	**5562**	5556(2.90)
G66	9000	6364	6364	6353.70(5.87)	6360	6323.53(6.34)	6360	6355.7(2.36)
G67	10,000	6944	6944	6937.30(3.30)	6942	6903.93(8.91)	**6946**	6939.7(2.77)
G70	10,000	9582	9548	9539.60(6.43)	9544	9527.80(9.32)	**9587**	9580.8(2.82)
G72	10,000	6998	6990	6979.70(6.58)	6998	6957.80(7.36)	**7000**	6995.8(2.75)
G77	14,000	9928	9902	9890.80(6.40)	9928	9920.00(3.08)	**9930**	9921.6(3.98)
G81	20,000	14,036	14,010	13,993.20(8.19)	14,036	14,020.30(8.50)	**14038**	14030.1(3.97)
Cubic lattice								
sg3dl141000	2744	2446	2446	2446	2446	2445.00(1.61)	2446	2444.1(1.99)
sg3dl142000	2744	2458	2458	2458	2458	2457.70(1.31)	2458	2457.7(0.73)
sg3dl143000	2744	2444	2442	2442	2444	2439.60(2.33)	2442	2440(1.12)
sg3dl144000	2744	2450	2450	2449.40(0.91)	2450	2448.10(2.23)	2450	2447.3(1.34)
sg3dl145000	2744	2446	2446	2446	2446	2444.90(2.23)	2446	2445.2(1.51)
sg3dl146000	2744	2452	2452	2451.40(0.91)	2452	2449.60(2.06)	2452	2449.7(0.98)
sg3dl147000	2744	2444	2444	2444	2444	2442.70(1.31)	2444	2441.6(1.39)
sg3dl148000	2744	2448	2448	2447.60(0.80)	2448	2446.40(1.50)	2448	2445.2(1.36)
sg3dl149000	2744	2428	2428	2426.30(0.71)	2428	2424.70(2.12)	2428	2425.1(1.51)
sg3dl1410000	2744	2460	2460	2458.40(0.80)	2458	2455.70(2.63)	2460	2455.6(2.11)

et al., 2015, 2012; Wu et al., 2015). We perform our algorithm on the 11 (respectively 10) largest instances of G-set (respectively the cubic lattices set) and compare our results with those of two state-of-the-art serial algorithms and the best known results provided by all other serial algorithms so far. The two compared serial algorithms TSHEA (Wu et al., 2015) and MOH (Ma & Hao, 2017) had provided or matched most previous best results on max-cut. Each instance is solved for 20 independent runs with random seeds. The parameter settings are listed in Table 1. The time limit of each run is 3 h for the G-set instances, and half hour for the cubic lattice instances. The time limit is comparable with what the reference state-of-the-art algorithms use (Ma & Hao, 2017; Wu et al., 2015). For the largest 11 instances of G-set, they use time limits of 2 h and about 2-6 h for each run respectively, 20 independently runs for each instance. For the 10 largest instances of the cubic lattice set, they use time limits of half hour and about 10 min for each run respectively, 20 independently runs for each instance. The computational results are listed in Table 2. Columns 1 to 3 corresponds to the instance names (Instance), the number of vertices ($|V|$), and the best known results provided by other serial algorithms to the best of our knowledge (f_{prev}). Column 4–5, 6–7 and 8–9 reported the computational statistics of TSHEA, MOH and our PF-ESL algorithm respectively, including their best results (f_{best}), average results (f_{avg}) and Standard Deviations (σ). The results of PF-ESL which are better than the best known results provided by other serial algorithms so far are bolded.

From the experimental results, one can observe that our algorithm is competitive. For the 12 instances in the first test set, our algorithm improves 6 of the corresponding best-known results, i.e., half of the best-known results. If compare our algorithm with the two state-of-the-art algorithm TSHEA and MOH one by one, its advantage is more significant on this benchmark set. For the second test set, it matched 9 and only missed 1 of the 10 best known results. Our algorithm also provides better average results and significant smaller Standard Deviation on most instances of the G-set benchmark. One may also observe that our algorithm is more competitive on larger instances. For the 7 instances with $|V| \geq 8000$, it improves 6 of the best known results by other serial algorithms, and only misses one; for the 5 instances with $|V| \geq 10000$, it improves all the best known results by other serial algorithms. It may be caused by the following reasons. Firstly, the solution spaces of the large instances are tremendous; the search process of an algorithm can only cover a very small part of it. Therefore, in these cases, it is very important to use a large population and good learning operators to cover enough promising search areas for effective diversification, and this is exactly what our PF-ESL method performs. For the smaller instances, the solution space is much smaller. Therefore, the search processes of an algorithm can cover relative larger ratio of the solution space. For these cases, rather than using large population and good learning operators to the end of effective diversification, it is more important to use small population, sophisticated local search procedure and small perturbation to provide better intensification. These are what the other state-of-the-art serial algorithms have accomplished. Furthermore, large graphs are prone to be sparser, which are prone to be disconnected or weakly connected. As illustrated by the example in Fig. 1, the advantage of our edge-states based learning will be more prominent on disconnected or weakly connected cases.

The results of this experiment suggest that our edge-state learning based memetic algorithm has a good discovery capacity for large instances.



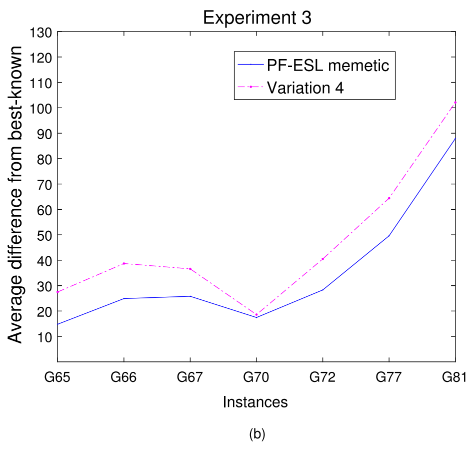


Fig. 3. Computational results of experiment 2 and 3. Variation 1 disable the guidance of PF-ESL in both the path relinking and the EDA perturbation operators of our algorithm. Variation 2 only activate the guidance of PF-ESL in the EDA perturbation operator of our algorithm. Variation 3 only activate the guidance of PF-ESL in the path relinking operator of our algorithm. Variation 4 replaces the path relinking operator of our algorithm with the solution combination operator proposed by the latest state-of-the-art memetic algorithm (Wu et al., 2015).

To further evaluate whether the *post-flip edge-state learning* method is important for challenging instances under a limited time, we further carry out the second and the third experiments. They are ablation studies that aimed to analyze whether and which novelties of our algorithms are value-added. To this end, they replace each novelty of our algorithm to previous state-of-the-art counterpart to see the change in efficiency. In the second experiment, we evaluate the learning method by comparing our algorithm with its three variations. Specifically, for the first variation, we disable the guidance of PF-ESL in both the path relinking and the EDA perturbation operators. For the second variation, we only activate the guidance of PF-ESL in the EDA perturbation operator. For the third variation, we only activate the guidance of PF-ESL in the path relinking operator. The path relinking operator without the guidance of PF-ESL is transformed into a common crossover operator, it crossovers two solutions based on the similarity on vertex positions rather than edge states just as the other state of the art memetic algorithms (Wang et al., 2012; Wu et al., 2015) do; the EDA based perturbation operator without the guidance of PF-ESL is transformed into a random perturbation operator just as the state-of-the-art Iterated Tabu search algorithm do (Ma et al., 2017). As for the third experiment, we further implement a variation (variation 4) by replacing the path-relinking operator of our algorithm with the solution-combination operator proposed by the latest state-of-the-art memetic algorithm (Wu et al., 2015), and compare our algorithm with it. For each experiment, we run our algorithm and the corresponding variations on the seven largest instances with $|V| \geq 8000$ under a time limit of 30 min for 20 independent runs, and then compare the average results obtained by each algorithm. Fig. 3 shows the results of the two experiments; each result is represented by $f_{best} - f_{avg}$, i.e., the difference between the corresponding best result in Table 1 and the average result over corresponding algorithm's 20 runs.

After the two experiments, we carry out Mann–Whitey U *test* to compare the results of our algorithm and its variations. For the second experiment, we found that for 6 (respectively 6 and 6) out of the 7 instances, the difference between the computational results obtained by our algorithm and the three variations is above 99.7% confidence significant. These results suggest that our edge-states based learning is value-added for both the path-relinking operator and the perturbation operator. As for the third experiment, we found that for 6 out of the 7 instances, the difference between the computational results obtained by our algorithm and the last variation is above 99.7% confidence significant. These results suggest that our edge-states learning based path-relinking operator is competitive comparing with the recent state-of-the-art solution-combination operator. From these two experiments, one can observe that our *post-flip edge-state learning* is important.

It is noticeable that all the test instances are sparse graphs. In many practical situations, graphs are sparse. For example, in social networks, people can only have social relationships with a limited number of people due to the limitation of human cognitive ability; in crystals, an atom can only interact with a few other atoms. But there are practical cases where graphs are dense. As analyzed in Section 2.4.3, the time complexity of each Tabu search iteration is $O(|E|^2/|V|^2)$. When the graph is sparse, the time consumption of each iteration is reduced to only $O(1)$. But when the graph is dense, the time consumption of each iteration can deteriorate to $O(|V|^2)$. This can make the execution of our algorithm very slow. Furthermore, when the weights of edges are diversified rather than chosen from a few candidate values such as 1 and -1, auxiliary data structure such as heaps will be needed to organize the headers of the doubly-linked lists mentioned in 2.4.1 and 2.4.2, the time complexity of each iteration can slightly deteriorate from $O(|E|^2/|V|^2)$ to $O(|E|^2/|V|^2 log|V|)$.

4. Conclusions

This paper has proposed an original learning method named post-flip edge-state learning (PF-ESL) for the max-cut problem. Different from previous state-of-the-art works, it regards edge states rather than vertex positions as the key information of a solution, and calculates the statistics on them over a population for its learning operations. The idea is based on the observations that edge states and edge weights are the only factors involved in the objective function, and edge states are consistent between any two symmetry local configurations but vertex positions are not. To overcome the difficulties caused by the dependencies between connected edges, the learning method samples the flips on vertices instead of setting edge-states directly. The flip on a vertex is sampled according to its capacity in increasing the similarity on edge states between the given solution and a given population. PF-ESL is used to construct an EDA perturbation operator and a path-relinking operator in our memetic algorithm. Experimental results demonstrate that our algorithm is competitive, especially for relative larger and more challenging instances. Comparative experiments also suggest that edge-states based learning is important for both perturbation and path relinking.

Graph optimization problems are a basic class of combinatory optimization problems with important applications. Many combinatorial optimization problems involve dividing graph data into different parts to satisfy certain constraints or maximize (or minimize) certain functions. This paper has posed a new perspective to what is the meaningful information of a divided graph, and introduced a novel method to measure and utilize the similarities between divided graphs whereupon.

Such a perspective is fundamental for learning based algorithms design for maxcut and other graph partitioning problems, and may inspire other related or similar problems.

Declaration of competing interest

One or more of the authors of this paper have disclosed potential or pertinent conflicts of interest, which may include receipt of payment, either direct or indirect, institutional support, or association with an entity in the biomedical field which may be perceived to have potential conflict of interest with this work. For full disclosure statements refer to https://doi.org/10.1016/j.eswa.2022.118077. Zhizhong Zeng reports financial support was provided by National Natural Science Foundation of China.

Acknowledgments

This work is supported by National Natural Science Foundation of China (Grant No. 61877026); National Natural Science Foundation of China for Outstanding Youth (Grant No. 72122006); Self-determined Research Funds of CCNU from the colleges' basic research and operation of MOE (Grant No. CCNU19ZN001); Natural Science Basic Research Program of Shaanxi (Grant No. 2020JM-089). We would give our special thanks to the anonymous referees for their professional comments and valuable suggestions that improve this paper significantly.

References

Arráiz, E., & Olivo, O. (2009). Competitive simulated annealing and tabu search algorithms for the max-cut problem. In *Proceedings of the 11th annual conference on genetic and evolutionary computation* (pp. 1797–1798). ACM.

Bansal, N., Feige, U., Krauthgamer, R., Makarychev, K., Nagarajan, V., Seffi, J., & Schwartz, R. (2014). Min-max graph partitioning and small set expansion. *SIAM Journal on Computing*, *43*(2), 872–904.

Benlic, U., & Hao, J.-K. (2013). Breakout local search for the max-cutproblem. *Engineering Applications of Artificial Intelligence*, *26*(3), 1162–1173.

Burer, S., Monteiro, R., & Zhang, Y. (2002). Rank-two relaxation heuristics for MAX-CUT and other binary quadratic programs. *SIAM Journal on Optimization*, *12*(2), 503–521.

Cho, J.-D., Raje, S., & Sarrafzadeh, M. (1998). Fast approximation algorithms on maxcut, k-coloring, and k-color ordering for VLSI applications. *IEEE Transactions on Computers*, *47*(11), 1253–1266.

Della Croce, F., Kaminski, M. J., & Paschos, V. T. (2007). An exact algorithm for MAX-CUT in sparse graphs. *Operations Research Letters*, *35*(3), 403–408.

Ding, C. H., He, X., Zha, H., Gu, M., & Simon, H. D. (2001). A min-max cut algorithm for graph partitioning and data clustering. In *Proceedings 2001 IEEE international conference on data mining* (pp. 107–114). IEEE.

Eisenblätter, A. (2002). The semidefinite relaxation of the k-partition polytope is strong. In *International conference on integer programming and combinatorial optimization* (pp. 273–290). Springer.

Festa, P., Pardalos, P., Resende, M., & Ribeiro, C. (2002). Randomized heuristics for the Max-Cut problem. *Optimization Methods & Software*, *17*(6), 1033–1058.

Fu, Z.-H., & Hao, J.-K. (2017). Knowledge-guided local search for the prize-collecting Steiner tree problem in graphs. *Knowledge-Based Systems*, *128*, 78–92.

Glover, F., & Hao, J.-K. (2010). Efficient evaluations for solving large 0-1 unconstrained quadratic optimisation problems. *International Journal of Metaheuristics*, *1*(1), 3–10.

Glover, F., & Laguna, M. (1998). Tabu search. In *Handbook of combinatorial optimization* (pp. 2093–2229). Springer.

Goemans, M. X., & Williamson, D. P. (1995). Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, *42*(6), 1115–1145.

Grosso, A., Locatelli, M., & Schoen, F. (2007). A population-based approach for hard global optimization problems based on dissimilarity measures. *Mathematical Programming*, *110*(2), 373–404.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations* (pp. 85–103). Springer.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., & Song, L. (2017). Learning combinatorial optimization algorithms over graphs. In *Advances in neural information processing systems* (pp. 6348–6358).

Kneis, J., & Rossmanith, P. (2005). *A new satisfiability algorithm with applications to max-cut*. Citeseer.

Kochenberger, G. A., Hao, J.-K., Lü, Z., Wang, H., & Glover, F. (2013). Solving large scale max cut problems via tabu search. *Journal of Heuristics*, *19*(4), 565–571.

Krishnan, K., & Mitchell, J. E. (2006). A semidefinite programming based polyhedral cut and price approach for the maxcut problem. *Computational Optimization and Applications*, *33*(1), 51–71.

Larrañaga, P., & Lozano, J. A. (2001). *Estimation of distribution algorithms: A new tool for evolutionary computation (vol. 2)*. Springer Science & Business Media.

Liers, F., Jünger, M., Reinelt, G., & Rinaldi, G. (2004). Computing exact ground states of hard ising spin glass problems by branch-and-cut. *New Optimization Algorithms in Physics*, 47–69.

Lü, Z., & Hao, J.-K. (2009). A critical element-guided perturbation strategy for iterated local search. In *European conference on evolutionary computation in combinatorial optimization* (pp. 1–12). Springer.

Ma, F., & Hao, J.-K. (2017). A multiple search operator heuristic for the max-k-cut problem. *Annals of Operations Research*, *248*(1–2), 365–403.

Ma, F., Hao, J.-K., & Wang, Y. (2017). An effective iterated tabu search for the maximum bisection problem. *Computers & Operations Research*, *81*, 78–89.

Martí, R., Duarte, A., & Laguna, M. (2009). Advanced scatter search for the max-cut problem. *INFORMS Journal on Computing*, *21*(1), 26–38.

Mitchell, J. E. (2003). Realignment in the national football league: Did they do it right? *Naval Research Logistics*, *50*(7), 683–701.

Shylo, V., Glover, F., & Sergienko, I. (2015). Teams of global equilibrium search algorithms for solving the weighted maximum cut problem in parallel. *Cybernetics and Systems*, *51*(1), 16–24.

Shylo, V., Shylo, O., & Roschyn, V. (2012). Solving weighted max-cut problem by global equilibrium search. *Cybernetics and Systems*, *48*(4), 563–567.

Tizhoosh, H. R. (2005). Opposition-based learning: a new scheme for machine intelligence. In *Computational intelligence for modelling, control and automation, 2005 and international conference on intelligent agents, web technologies and internet commerce, international conference on (vol. 1)* (pp. 695–701). IEEE.

Wang, Y., Lü, Z., Glover, F., & Hao, J.-K. (2012). Path relinking for unconstrained binary quadratic programming. *European Journal of Operational Research*, *223*(3), 595–604.

Wu, Q., Wang, Y., & Lü, Z. (2015). A tabu search based hybrid evolutionary algorithm for the max-cut problem. *Applied Soft Computing*, *34*, 827–837.

Zhou, Y., Duval, B., & Hao, J.-K. (2018). Improving probability learning based local search for graph coloring. *Applied Soft Computing*, *65*, 542–553.

Zhou, Y., & Hao, J.-K. (2017). Frequency-driven tabu search for the maximum s-plex problem. *Computers & Operations Research*, *86*, 65–78.