Ann Oper Res (2017) 248:365–403
DOI 10.1007/s10479-016-2234-0

ORIGINAL PAPER

A multiple search operator heuristic for the max-k-cut problem

Fuda Ma[1] · **Jin-Kao Hao**[1,2]

Published online: 1 June 2016

Abstract The max-k-cut problem is to partition the vertices of an edge-weighted graph $G = (V, E)$ into $k \geq 2$ disjoint subsets such that the weight sum of the edges crossing the different subsets is maximized. The problem is referred as the max-cut problem when $k = 2$. In this work, we present a multiple operator heuristic (MOH) for the general max-k-cut problem. MOH employs five distinct search operators organized into three search phases to effectively explore the search space. Experiments on two sets of 91 well-known benchmark instances show that the proposed algorithm is highly effective on the max-k-cut problem and improves the current best known results (lower bounds) of most of the tested instances for $k \in [3, 5]$. For the popular special case $k = 2$ (i.e., the max-cut problem), MOH also performs remarkably well by discovering 4 improved best known results. We provide additional studies to shed light on the key ingredients of the algorithm.

Keywords Max-k-cut and max-cut · Graph partition · Multiple search strategies · Tabu list · Heuristics

1 Introduction

Let $G = (V, E)$ be an undirected graph with vertex set $V = \{1, \ldots, n\}$ and edge set $E \subset V \times V$, each edge $(i, j) \in E$ being associated a weight $w_{ij} \in Z$. Given $k \in [2, n]$, the max-k-cut problem is to partition the vertex set V into k (k is given) disjoint subsets $\{S_1, S_2, \ldots, S_k\}$, (i.e., $\overset{k}{\underset{i=1}{\cup}} S_i = V, S_i \neq \emptyset, S_i \cap S_j = \emptyset, \forall i \neq j$), such that the sum of

✉ Jin-Kao Hao
hao@info.univ-angers.fr

Fuda Ma
ma@info.univ-angers.fr

1 LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France

2 Institut Universitaire de France, Paris, France

weights of the edges from E whose endpoints belong to different subsets is maximized, i.e.,

$$\max \sum_{1 \leq p < q \leq k} \sum_{i \in S_p, j \in S_q} w_{ij}. \tag{1}$$

Particularly, when the number of partitions equals 2 (i.e., $k = 2$), the problem is referred as the max-cut problem. Max-k-cut is equivalent to the minimum k-partition (MkP) problem which aims to partition the vertex set of a graph into k disjoint subsets so as to minimize the total weight of the edges joining vertices in the same partition (Ghaddar et al. 2011).

The max-k-cut problem is a classical NP-hard problem in combinatorial optimization and can not be solved exactly in polynomial time (Boros and Hammer 1991; Kann et al. 1997). Moreover, when $k = 2$, the max-cut problem is one of the Karp's 21 NP-complete problems (Karp 1972) which has been subject of many studies in the literature.

In recent decades, the max-k-cut problem has attracted increasing attention for its applicability to numerous important applications in the area of data mining (Ding et al. 2001), VLSI layout design (Barahona et al. 1988; Chang and Du 1987; Chen et al. 1983; Pinter 1984; Cho et al. 1998), frequency planning (Eisenblätter 2002), sports team scheduling (Mitchell 2003), and statistical physics (Liers et al. 2004) among others.

Given its theoretical significance and large application potential, a number of solution procedures for solving the max-k-cut problem (or its equivalent MkP) have been reported in the literature. In Ghaddar et al. (2011), the authors provide a review of several exact algorithms which are based on branch-and-cut and semidefinite programming approaches. But due to the high computational complexity of the problem, only instances of reduced size (i.e., $|V| < 100$) can be solved by these exact methods in a reasonable computing time.

For large instances, heuristic and metaheuristic methods are commonly used to find "good-enough" sub-optimal solutions. In particular, for the very popular max-cut problem, many heuristic algorithms have been proposed, including simulated annealing and tabu search (Arráiz and Olivo 2009), breakout local search (Benlic and Hao 2013), projected gradient approach (Burer and Monteiro 2001), discrete dynamic convexized method (Lin and Zhu 2012), rank-2 relaxation heuristic (Burer et al. 2002), variable neighborhood search (Festa et al. 2002), greedy heuristics (Kahruman et al. 2007), scatter search (Martí et al. 2009), global equilibrium search (Shylo et al. 2012) and its parallel version (Shylo et al. 2015), memetic search (Lin and Zhu 2014; Wu and Hao 2012; Wu et al. 2015), and unconstrained binary quadratic optimization (Wang et al. 2013). Compared with max-cut, there are much fewer heuristics for the general max-k-cut problem or its equivalent MkP. Among the rare existing studies, we mention the very recent discrete dynamic convexized (DC) method of Zhu et al. (2013), which formulates the max-k-cut problem as an explicit mathematical model and uses an auxiliary function based local search to find satisfactory results.

In this paper, we partially fill the gap by presenting a new and effective heuristic algorithm for the general max-k-cut problem. We identify the contributions of the work as follows.

– In terms of algorithmic design, the main originality of the proposed algorithm is its multi-phased multi-strategy approach which relies on five distinct local search operators for solution transformations. The five employed search operators ($O_1 - O_5$) are organized into three different search phases to ensure an effective examination of the search space. The descent-based improvement phase uses the intensification operators O_1–O_2 to find a (good) local optimum from a starting solution. Then by applying two additional operators (O_3–O_4), the diversified improvement phase aims to discover promising areas around the obtained local optimum which are then further explored by the descent-based improvement phase. Finally, since the search can get trapped in local optima, the per-

turbation phase applies a random search operator (O_5) to definitively lead the search to a distant region from which a new round of the search procedure starts. This process is repeated until a stopping condition is met. To ensure a high computational efficiency of the algorithm, we employ bucket-sorting based techniques to streamline the calculations of the different search operators.
– In terms of computational results, we assess the performance of the proposed algorithm on two sets of well-known benchmarks with a total of 91 instances which are commonly used to test max-k-cut and max-cut algorithms in the literature. Computational results show that the proposed algorithm competes very favorably with respect to the existing max-k-cut heuristics, by improving the current best known results on most instances for $k \in [3, 5]$. Moreover, for the very popular max-cut problem ($k = 2$), the results yielded by our algorithm remain highly competitive compared with the most effective and dedicated max-cut algorithms. In particular, our algorithm manages to improve the current best known solutions for 4 (large) instances, which were previously reported by specific max-cut algorithms of the literature.

The rest of the paper is organized as follows. In Sect. 2, the proposed algorithm is presented. Section 3 provides computational results and comparisons with state-of-the-art algorithms in the literature. Section 4 is dedicated to an analysis of several essential parts of the proposed algorithm. Concluding remarks are given in Sect. 5.

2 Multiple search operator heuristic for max-k-cut

2.1 General working scheme

The proposed multiple operator heuristic algorithm (MOH) for the general max-k-cut problem is described in Algorithm 1 whose components are explained in the following subsections. The algorithm explores the search space (Sect. 2.2) by alternately applying five distinct search operators (O_1 to O_5) to make transitions from the current solution to a neighbor solution (Sect. 2.4). Basically, from an initial solution, the descent-based improvement phase aims, with two operators (O_1 and O_2), to reach a local optimum I (Algorithm 1, lines 10–19, descent-based improvement phase, Sect. 2.6). Then the algorithm continues to the diversified improvement phase (Algorithm 1, lines 28–38, Sect. 2.7) which applies two other operators (O_3 and O_4) to locate new promising regions around the local optimum I. This second phase ends once a better solution than the current local optimum I is discovered or when a maximum number of diversified moves ω is reached. In both cases, the search returns to the descent-based improvement phase with the best solution found as its new starting point. If no improvement can be obtained after ξ descent-based improvement and diversified improvement phases, the search is judged to be trapped in a deep local optimum. To escape the trap and jump to an unexplored region, the search turns into a perturbation-based diversification phase (Algorithm 1, lines 40–43), which uses a random operator (O_5) to strongly transform the current solution (Sect. 2.8). The perturbed solution serves then as the new starting solution of the next round of the descent-based improvement phase. This process is iterated until the stopping criterion (typically a cutoff time limit) is met.

2.2 Search space and evaluation solution

Recall that the goal of max-k-cut is to partition the vertex set V into k subsets such that the sum of weights of the edges between the different subsets is maximized. As such, we define

Algorithm 1 General procedure for the max-k-cut problem

1: **Input**: Graph $G = (V, E)$, number of partitions k, max number ω of diversified moves, max number ξ of consecutive non-improvement rounds of the descent improvement and diversified improvement phases before the perturbation phase, probability ρ for applying operator O_3, γ the perturbation strength.
2: **Output**: the best solution I_{best} found so far
3: $I \leftarrow$ Generate_initial_solution(V, k) ▷ I is a partition of V into k subsets
4: $I_{best} \leftarrow I$ ▷ I_{best} Records the best solution found so far
5: $f_{lo} \leftarrow f(I)$ ▷ f_{lo} Records the objective value of the latest local optimum reached by $O_1 \cup O_2$
6: $f_{best} \leftarrow f(I)$ ▷ f_{best} Records the best objective value found so far
7: $c_{non\_impv} \leftarrow 0$ ▷ Counter of consecutive non-improvement rounds of descent and diversified search
8: **while** stopping condition not satisfied **do**
9: /* lines 10 to 19: Descent-based improvement phase by applying O_1 and O_2, see Sect. 2.4*/
10: **repeat**
11: **while** $f(I \oplus O_1) > f(I)$ **do** ▷ Descent Phase by applying operator O_1
12: $I \leftarrow I \oplus O_1$ ▷ Perform the move defined by O_1
13: Update Δ ▷ Δ is the bucket structure recording move gains for vertices, see Sect. 2.5
14: **end while**
15: **if** $f(I \oplus O_2) > f(I)$ **then** ▷ Descent Phase by applying operator O_2
16: $I \leftarrow I \oplus O_2$
17: Update Δ
18: **end if**
19: **until** I can not be improved by operator O_1 and O_2
20: $f_{lo} \leftarrow f(I)$
21: **if** $f(I) > f_{best}$ **then**
22: $f_{best} \leftarrow f(I)$; $I_{best} \leftarrow I$ ▷ Update the best solution found so far
23: $c_{non\_impv} \leftarrow 0$ ▷ Reset counter c_{non_impv}
24: **else**
25: $c_{non\_impv} \leftarrow c_{non\_impv} + 1$
26: **end if**
27: /* lines 28 to 38: Diversified improv. phase by applying O_3 and O_4 at most ω times, see Sect. 2.4 */
28: $c_{div} \leftarrow 0$ ▷ Counter c_{div} records number of diversified moves
29: **repeat**
30: **if** $Random(0, 1) < \rho$ **then** ▷ Random(0,1) returns a random real number between 0 to 1
31: $I \leftarrow I \oplus O_3$
32: **else**
33: $I \leftarrow I \oplus O_4$
34: **end if**
35: Update H (H, λ) ▷ Update tabu list H where λ is the tabu tenure, see Sect. 2.4
36: Update Δ ▷ Update the move gains impacted by the move, see Sect. 2.5
37: $c_{div} \leftarrow c_{div} + 1$
38: **until** $c_{div} > \omega$ or $f(I) > f_{lo}$
39: /* Perturbation phase by applying O_5 if f_{best} not improved for ξ rounds of phases 1-2, see Sect. 2.8 */
40: **if** $c_{non\_impv} > \xi$ **then**
41: $I \leftarrow I \oplus O_5$ ▷ Apply random perturbation γ times, see Sect. 2.8
42: $c_{non\_impv} \leftarrow 0$
43: **end if**
44: **end while**

the search space Ω explored by our algorithm as the set of all possible partitions of V into k disjoint subsets, $\Omega = \{\{S_1, S_2, \ldots, S_k\} : \bigcup_{i=1}^{k} S_i = V, S_i \cap S_j = \emptyset, S_i \subset V, \forall i \neq j\}$, where each candidate solution is called a k-cut.

For a given partition or k-cut $I = \{S_1, S_2, \ldots, S_k\} \in \Omega$, its objective value $f(I)$ is the sum of weights of the edges connecting two different subsets:

$$f(I) = \sum_{1 \leq p < q \leq k} \sum_{i \in S_p, j \in S_q} w_{ij}. \tag{2}$$

Then, for two candidate solutions $I' \in \Omega$ and $I'' \in \Omega$, I' is better than I'' if and only if $f(I') > f(I'')$. The goal of our algorithm is to find a solution $I_{best} \in \Omega$ with $f(I_{best})$ as large as possible.

2.3 Initial solution

The MOH algorithm needs an initial solution to start its search. Generally, the initial solution can be provided by any eligible means. In our case, we adopt a randomized two step procedure. First, from k empty subsets $S_i = \emptyset, \forall i \in \{1, \ldots, k\}$, we assign each vertex $v \in V$ to a random subset $S_i \in \{S_1, S_2, \ldots, S_k\}$. Then if some subsets are still empty, we repetitively move a vertex from its current subset to an empty subset until no empty subset exists.

2.4 Move operations and search operators

Our MOH algorithm iteratively transforms the incumbent solution to a neighbor solution by applying some *move* operations. Typically, a move operation (or simply a move) changes slightly the solution, e.g., by transferring a vertex to a new subset. Formally, let I be the incumbent solution and let mv be a move, we use $I' \leftarrow I \oplus mv$ to denote the neighbor solution I' obtained by applying mv to I.

Associated to a move operation mv, we define the notion of *move gain* Δ_{mv}, which indicates the objective change between the incumbent solution I and the neighbor solution I' obtained after applying the move, i.e.,

$$\Delta_{mv} = f(I') - f(I) \tag{3}$$

where f is the optimization objective [see Formula (2)].

In order to efficiently evaluate the move gain of a move, we develop dedicated techniques which are described in Sect. 2.5. In this work, we employ two basic move operations: the *'single-transfer move'* and the *'double-transfer move'*. These two move operations form the basis of our five search operators.

- Single-transfer move (*st*): Given a k-cut $I = \{S_1, S_2, \ldots, S_k\}$, a vertex $v \in S_p$ and a target subset S_q with $p, q \in \{1, \ldots, k\}$, $p \neq q$, the 'single-transfer move' displaces vertex $v \in S_p$ from its current subset S_p to the target subset $S_q \neq S_p$. We denote this move by $st(v, S_p, S_q)$ or $v \rightarrow S_q$.
- Double-transfer move (*dt*): Given a k-cut $I = \{S_1, S_2, \ldots, S_k\}$, the 'double-transfer move' displaces vertex u from its subset S_{cu} to a target subset $S_{tu} \neq S_{cu}$, and displaces vertex v from its current subset S_{cv} to a target subset $S_{tv} \neq S_{cv}$. We denote this move by $dt(u, S_{cu}, S_{tu}; v, S_{cv}, S_{tv})$ or $dt(u, v)$, or still dt.

From these two basic move operations, we define five distinct *search operators* $O_1 - O_5$ which indicate precisely how these two basic move operations are applied to transform an incumbent solution to a new solution. After an application of any of these search operators, the move gains of the impacted moves are updated according to the dedicated techniques explained in Sect. 2.5.

- **The $\mathbf{O_1}$ search operator** applies the single-transfer move operation. Precisely, O_1 selects among the $(k-1)n$ single-transfer moves a best move $v \rightarrow S_q$ such that the induced move gain $\Delta_{(v \rightarrow S_q)}$ is maximum. If there are more than one such moves, one of them is selected at random. Since there are $(k-1)n$ candidate single-transfer moves from a given solution, the time complexity of O_1 is bounded by $O(kn)$. The proposed MOH algorithm employs this search operator as its main intensification operator which is complemented by the O_2 search operator to locate good local optima (see Algorithm 1, lines 10–19 and Sect. 2.6).

– **The O$_2$ search operator** is based on the double-transfer move operation and selects a best dt move with the largest move gain Δ_{dt}. If there are more than one such moves, one of them is selected at random.

Let $dt(u, S_{cu}, S_{tu}; v, S_{cv}, S_{tv})$ ($S_{cu} \neq S_{tu}$, $S_{cv} \neq S_{tv}$) be a double-transfer move, then the move gain Δ_{dt} of this double transfer move can be calculated by a combination of the move gains of its two underlying single-transfer moves ($\Delta_{u \to S_{tu}}$ and $\Delta_{v \to S_{tv}}$) as follows:

$$\Delta_{dt(u,v)} = \Delta_{u \to S_{tu}} + \Delta_{v \to S_{tv}} + \psi \omega_{uv} \tag{4}$$

where ω_{uv} is the weight of edge $e(u, v) \in E$ and ψ is a coefficient which is determined as follows:

$$\psi = \begin{cases} -2, & \text{if } S_{cu} = S_{cv}, S_{tu} = S_{tv} \\ 2, & \text{if } S_{tu} = S_{cv}, S_{cu} = S_{tv} \\ -1, & \text{if } S_{cu} = S_{cv}, S_{tu} \neq S_{tv} \\ 1, & \text{if } S_{cu} = S_{tv}, S_{tu} \neq S_{cv} \\ -1, & \text{if } S_{cu} \neq S_{cv}, S_{tu} = S_{tv} \\ 1, & \text{if } S_{cu} \neq S_{tv}, S_{tu} = S_{cv} \\ 0, & \text{if } S_{cu} \neq S_{cv}, S_{tu} \neq S_{cv}, S_{cu} \neq S_{tv}, S_{tu} \neq S_{tv} \end{cases} \tag{5}$$

The operator O_2 is used when O_1 exhausts its improving moves and provides a first means to help the descent-based improvement phase to escape the current local optimum and discover solutions of increasing quality. Given an incumbent solution, there are a total number of $(k-1)^2 n^2$ candidate double-transfer moves denoted as set DT. Seeking directly the best move with the maximum Δ_{dt} among all these possible moves would just be too computationally expensive. In order to mitigate this problem, we devise a strategy to accelerate the move evaluation process.

From Formula (4), one observes that among all the vertices in V, only the vertices verifying the condition $\omega_{uv} \neq 0$ and $\Delta_{dt(u,v)} > 0$ are of interest for the double-transfer moves. Note that without the condition $\omega_{uv} \neq 0$, performing a double-transfer move would actually equal to two consecutive single-transfer moves, which on the one hand makes the operator O_2 meaningless and on the other hand fails to get an increased objective gain. Thus, by examining only the endpoint vertices of edges in E, we shrink the move combinations by building a reduced subset: $DT^R = \{dt(u, v) : dt(u, v) \in DT, \omega_{uv} \neq 0, \Delta_{dt(u,v)} > 0\}$. Based on DT^R, the complexity of examining all possible double-transfer moves drops to $O(|E|)$, which is not related to k. In practice, one can examine $\phi|E|$ endpoint vertices in case $|E|$ is too large. We empirically set $\phi = 0.1/d$, where d is the highest degree of the graph.

To summarize, the O_2 search operator selects two st moves $u \to S_{tu}$ and $v \to S_{tv}$ from the reduced set DT^R, such that the combined move gain $\Delta_{dt(u,v)}$ according to Formula (4) is maximum.

– **The O$_3$ search operator**, like O_1, selects a best single-transfer move (i.e., with the largest move gain) while considering a tabu list H (Glover and Laguna 1999). The tabu list is a memory which is used to keep track of the performed st moves to avoid revisiting previously encountered solutions. As such, each time a best st move is performed to displace a vertex v from its original subset to a target subset, v becomes tabu and is forbidden to move back to its original subset for the next λ iterations (called tabu tenure). In our case, the tabu tenure is dynamically determined as follows.

$$\lambda = rand(3, n/10) \tag{6}$$

where $rand(3, n/10)$ denotes a random integer between 3 and $n/10$.
Based on the tabu list, O_3 considers all possible single-transfer moves except those forbidden by the tabu list H and selects the best st move with the largest move gain Δ_{st}. Note that a forbidden move is always selected if the move leads to a solution better than the best solution found so far. This is called aspiration in tabu search terminology (Glover and Laguna 1999).
Although both O_3 and O_1 use the single-transfer move, they are two different search operators and play different roles within the MOH algorithm. On the one hand, as a pure descent operator, O_1 is a faster operator compared to O_3 and is designed to be an intensification operator. Since O_1 alone has no any diversification capacity and always ends with the local optimum encountered, it is jointly used with O_2 to visit different local optima. On the other hand, due to the use of the tabu list, O_3 can accept moves with a negative move gain (leading to a worsening solution). As such, unlike O_1, O_3 has some diversification capacity, and when jointly used with O_4, helps the search to examine nearby regions around the input local optimum to find better solutions (see Algorithm 1, lines 28–38 and Sect. 2.7).
- **The O$_4$ search operator**, like O_2, is based on the double-transfer operation. However, O_4 strongly constraints the considered candidate dt moves with respect to two target subsets which are randomly selected. Specifically, O_4 operates as follows. Select two target subsets S_p and S_q at random, and then select two single-transfer moves $u \rightarrow S_p$ and $v \rightarrow S_q$ such that the combined move gain $\Delta_{dt(u,v)}$ according to Formula (4) is maximum.
  Operator O_4 is jointly used with operator O_3 to ensure the diversified improvement search phase.
- **The O$_5$ search operator** is based on a randomized single-transfer move operation. O_5 first selects a random vertex $v \in V$ and a random target subset S_p, where $v \notin S_p$ and then moves v from its current subset to S_p. This operator is used to change randomly the incumbent solution for the purpose of (strong) diversification when the search is considered to be trapped in a deep local optimum (see Sect. 2.8).

Among the five search operators, four of them (O_1–O_4) need to find a single-transfer move with the maximum move gain. To ensure a high computational efficiency of these operators, we develop below a streamlining technique for fast move gain evaluation and move gain updates.

2.5 Bucket sorting for fast move gain evaluation and updating

The algorithm needs to rapidly evaluate a number of candidate moves at each iteration. Since all the search operators basically rely on the single-transfer move operation, we developed a fast incremental evaluation technique based on a bucket data structure to keep and update the move gains after each move application (Cormen et al. 2001). Our streamlining technique can be described as follows: let $v \rightarrow S_x$ be the move of transferring vertex v from its current subset S_{cv} to any other subset S_x, $x \in \{1, \ldots, k\}$, $x \neq cv$. Then initially, each move gain is determined as follows:

$$\Delta_{v \rightarrow S_x} = \sum_{i \in S_{cv}, i \neq v} \omega_{vi} - \sum_{j \in S_x} \omega_{vj}, \ x \in \{1, \ldots, k\}, \quad x \neq cv \tag{7}$$

where ω_{vi} and ω_{vj} are respectively the weights of edges $e(v, i)$ and $e(v, j)$.
Suppose the move $v \rightarrow S_{tv}$, i.e., displacing v from S_{cv} to S_{tv}, is performed, the move gains can be updated by performing the following calculations:

1. for each $S_x \neq S_{cv}$, $S_x \neq S_{tv}$, $\Delta_{v \to S_x} = \Delta_{v \to S_x} - \Delta_{v \to S_{tv}}$
2. $\Delta_{v \to S_{cv}} = -\Delta_{v \to S_{tv}}$
3. $\Delta_{v \to S_{tv}} = 0$
4. for each $u \in V - \{v\}$, moving $u \in S_{cu}$ to each other subset $S_y \in S - \{S_{cu}\}$,

$$\Delta_{u \to S_y} = \begin{cases} \Delta_{u \to S_y} - 2\omega_{uv}, & \text{if } S_{cu} = S_{cv}, S_y = S_{tv} \\ \Delta_{u \to S_y} + 2\omega_{uv}, & \text{if } S_{cu} = S_{tv}, S_y = S_{cv} \\ \Delta_{u \to S_y} - \omega_{uv}, & \text{if } S_{cu} = S_{cv}, S_y \neq S_{tv} \\ \Delta_{u \to S_y} + \omega_{uv}, & \text{if } S_{cu} = S_{tv}, S_y \neq S_{cv} \\ \Delta_{u \to S_y} - \omega_{uv}, & \text{if } S_{cu} \neq S_{cv}, S_y = S_{tv} \\ \Delta_{u \to S_y} + \omega_{uv}, & \text{if } S_{cu} \neq S_{tv}, S_y = S_{cv} \\ \Delta_{u \to S_y}, & \text{if } S_{cu} \neq S_{cv}, S_{cu} \neq S_{tv}, S_y \neq S_{cv}, S_y \neq S_{tv} \end{cases} \tag{8}$$

For low-density graphs, $\omega_{uv} = 0$ stands for most cases. Hence, we only update the move gains of vertices affected by this move (i.e., the displaced vertex and its adjacent vertices), which reduces the computation time significantly.

The move gains can be stored in an vector, with which the time for finding the best move grows linearly with the number of vertices and partitions ($O(kn)$). For large problem instances, the required time to search the best move can still be quite high, which is particular true when k is large. To further reduce the computing time, we adapted the bucket sorting technique of Fiduccia and Mattheyses (1982) initially proposed for the two-way network partitioning problem to the max-k-cut problem. The idea is to keep the vertices ordered by the move gains in decreasing order in k arrays of buckets, one for each subset $S_i \in \{S_1, S_2, \ldots, S_k\}$. In each bucket array i, the jth entry stores in a doubly linked list the vertices with the move gain $\Delta_{v \to S_i}$ currently equaling j. To ensure a direct access to each vertex in the doubly linked lists, as suggested in Fiduccia and Mattheyses (1982), we maintain another array for all vertices, where each element points to its corresponding vertex in the doubly linked lists.

Figure 1 shows an example of the bucket structure for $k = 3$ and $n = 8$. The 8 vertices of the graph (Fig. 1, left) are divided to 3 subsets S_1, S_2 and S_3. The associated bucket structure (Fig. 1, right) shows that the move gains of moving vertices e, g, h to subset S_1 equal -1, then they are stored in the entry of B_1 with index of -1 and are managed as a doubly linked list. The array AI shown at the bottom of Fig. 1 manages position indexes of all vertices.

For each array of buckets, finding the best vertex with maximum move gain is equivalent to finding the first non-empty bucket from top of the array and then selecting a vertex in its doubly linked list. If there are more than one vertices in the doubly linked list, a random vertex in this list is selected. To further reduce the searching time, the algorithm memorizes the position of the first non-empty bucket (e.g., $gmax_1$, $gmax_2$, $gmax_3$ in Fig. 1). After each move, the bucket structure is updated by recomputing the move gains (see Formula (8)) of the affected vertices which include the moved vertex and its adjacent vertices, and shifting them to appropriate buckets. For instance, the steps of performing an O_1 move based on Fig. 1 are shown as follows: First, obtain the index of maximum move gain in the bucket arrays by calculating $max(gmax_1, gmax_2, gmax_3)$, which equals $gmax_3$ in this case. Second, select randomly a vertex indexed by $gmax_3$, vertex b in this case. At last, update the positions of the affected vertices a, b, d.

The complexity of each move consists in (1) searching for the vertex with maximum move gain in $O(l)$ (l being the current length of the doubly link list with the maximum gain, typically much smaller than n), (2) recomputing the move gains for the affected vertices

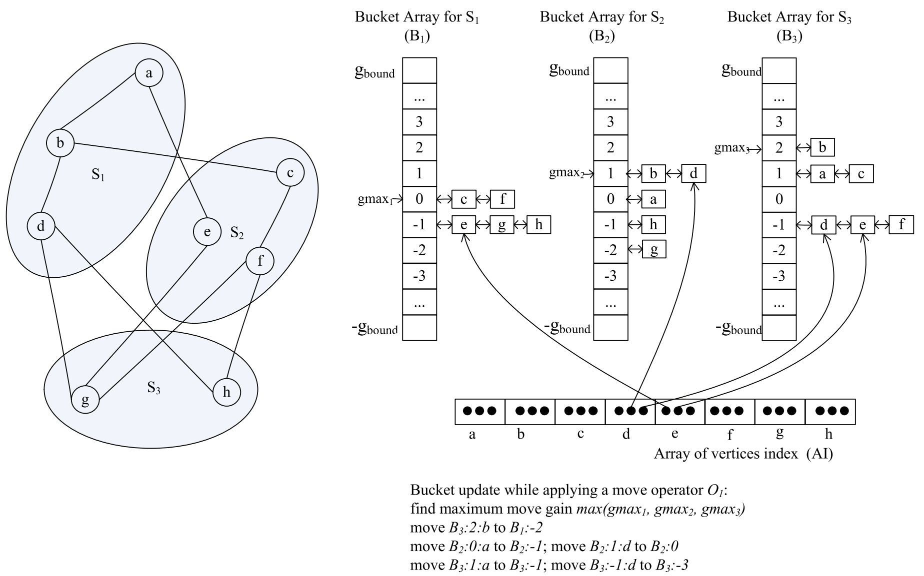

Fig. 1 An example of bucket structure for max-3-cut

in $O(kd_{max})$ (d_{max} being the maximum degree of the graph), and (3) updating the bucket structure in $O(kd_{max})$.

Bucket data structures have been previously applied to the specific max-cut and max-bisection problems (Benlic and Hao 2013; Lin and Zhu 2014; Zhu et al. 2015). This work presents the first adaptation of the bucket sorting technique to the general max-k-cut problem.

2.6 Descent-based improvement phase for intensified search

The descent-based local search is used to obtain a local optimum from a given starting solution. As described in Algorithm 1 (lines 10–19), we alternatively uses two search operators O_1 and O_2 defined in Sect. 2.4 to improve a solution until reaching a local optimum. Starting from the given initial solution, the procedure first applies O_1 to improve the incumbent solution. According to the definition of O_1 in Sect. 2.4, at each step, the procedure examines all possible single-transfer moves and selects a move $v \rightarrow S_q$ with the largest move gain $\Delta_{v \rightarrow S_q}$ subject to $\Delta_{v \rightarrow S_q} > 0$, and then performs that move. After the move, the algorithm updates the bucket structure of move gains according to the technique described in Sect. 2.5.

When the incumbent solution can not be improved by O_1 (i.e., $\forall v \in V, \forall S_q, \Delta_{v \rightarrow S_q} \leq 0$), the procedure turns to O_2 which makes one *best* double-transfer move. If an improved solution is discovered with respect to the local optimum reached by O_1, we are in a new promising area. We switch back to operator O_1 to resume an intensified search to attain a new local optimum. The descent-based improvement phase stops when no better solution can be found with O_1 and O_2. The last solution is a local optimum I_{lo} with respect to the single-transfer and double-transfer moves and serves as the input solution of the second search phase which is explained in the next section.

2.7 Diversified improvement phase for discovering promising region

The descent-based local phase described in Sect. 2.6 alone can not go beyond the best local optimum I_{lo} it encounters. The diversified improvement search phase is used 1) to jump out of this local optimum and 2) to intensify the search around this local optimum with the hope of discovering other improved solutions better than the input local optimum I_{lo}. The diversified improvement search procedure alternatively uses two search operators O_3 and O_4 defined in Sect. 2.4 to perform moves until a prescribed condition is met (see below and Algorithm 1, line 38). The application of O_3 or O_4 is determined probabilistically: with probability ρ, O_3 is applied; with $1-\rho$, O_4 is applied.

When O_3 is selected, the algorithm searches for a best single transfer move $v \rightarrow S_q$ with maximum move gain $\Delta_{v\rightarrow S_q}$ which is not forbidden by the tabu list or verifies the aspiration criterion. Each performed move is then recorded in the tabu list H and is classified tabu for the next λ (calculated by Formula (6)) iterations. The bucket structure is updated to actualize the impacted move gains accordingly. Note that the algorithm only keeps and updates the tabu list during the diversified improvement search phase. Once this second search phase terminates, the tabu list is cleared up.

Similarly, when O_4 is selected, two subsets are selected at random and a best double-transfer dt move with maximum move gain Δ_{dt} is determined from the bucket structure (break ties at random). After the move, the bucket structure is updated to actualize the impacted move gains.

The diversified improvement search procedure terminates once a solution better than the input local optimum I_{lo} is found, or a maximum number ω of diversified moves (O_3 or O_4) is reached. Then the algorithm returns to the descent-based search procedure and use the current solution I as a new starting point for the descent-based search. If the best solution founded so far (f_{best}) can not be improved over a maximum allowed number ξ of consecutive rounds of the descent-based improvement and diversified improvement phases, the search is probably trapped in a deep local optima. Consequently, the algorithm switches to the perturbation phase (Sect. 2.8) to displace the search to a distant region.

2.8 Perturbation phase for strong diversification

The diversified improvement phase makes it possible for the search to escape some local optima. However, the algorithm may still get deeply stuck in a non-promising regional search area. This is the case when the best-found solution f_{best} can not be improved after ξ consecutive rounds of descent and diversified improvement phases. Thus the random perturbation is applied to strongly change the incumbent solution.

The basic idea of the perturbation consists in applying the O_5 operator γ times. In other words, this perturbation phase moves γ randomly selected vertices from their original subset to a new and randomly selected subset. Here, γ is used to control the perturbation strength; a large (resp. small) γ value changes strongly (resp. weakly) the incumbent solution. In our case, we adopt $\gamma = 0.1|V|$, i.e., as a percent of the number of vertices. After the perturbation phase, the search returns to the descent-based improvement phase with the perturbed solution as its new starting solution.

3 Experimental results and comparisons

3.1 Benchmark instances

To evaluate the performance of the proposed MOH approach, we carried out computational experiments on two sets of well-known benchmarks with a total of 91 large instances of the literature.[1] The first set (G-set) is composed of 71 graphs with 800–20,000 vertices and an edge density from 0.02 to 6 %. These instances were previously generated by a machine-independent graph generator including toroidal, planar and random weighted graphs. These instances are available from: http://www.stanford.edu/yyye/yyye/Gset. The second set comes form (Burer et al. 2002), arising from 30 cubic lattices with randomly generated interaction magnitudes. Since the 10 small instances (with less than 1000 vertices) of the second set are very easy for our algorithm, only the results of the 20 larger instances with 1000 to 2744 vertices are reported. These well-known benchmarks were frequently used to evaluate the performance of max-bisection, max-cut and max-k-cut algorithms (Benlic and Hao 2013; Festa et al. 2002; Shylo et al. 2012, 2015; Wang et al. 2013; Wu and Hao 2012, 2013; Wu et al. 2015; Zhu et al. 2013).

3.2 Experimental protocol

The proposed MOH algorithm was programmed in C++ and compiled with GNU g++ (optimization flag "−O2"). Our computer is equipped with a Xeon E5440/2.83GHz CPU with 2GB RAM. When testing the DIMACS machine benchmark[2], our machine requires 0.43, 2.62 and 9.85 CPU time in seconds respectively for graphs r300.5, r400.5, and r500.5 compiled with g++ −O2.

3.3 Parameters

The MOH algorithm requires five parameters: tabu tenure λ, maximum number ω of diversified moves, maximum number ξ of consecutive non-improving rounds of the descent and diversified improvement phases before the perturbation phase, probability ρ for applying the operator O_3, and perturbation strength γ. For the tabu tenure λ, we adopted the recommended setting of the Breakout Local Search (Benlic and Hao 2013), which performs quite well for the benchmark graphs. For each of the other parameters, we first identified a collection of varying values and then determined the best setting by testing the candidate values of the parameter while fixing the other parameters to their default values. This parameter study was based on a selection of 10 representative and challenging G-set instances (G22, G23, G25, G29, G33, G35, G36, G37, G38 and G40). For each parameter setting, 10 independent runs of the algorithm were conducted for each instance and the average objective values over the 10 runs were recorded. If a large parameter value presents a better result, we gradually increase its value; otherwise, we gradually decrease its value. By repeating the above procedure, we determined the following parameter settings: $\lambda = rand(3, |V|/10)$, $\omega = 500$, $\xi = 1000$, $\rho = 0.5$, and $\gamma = 0.1|V|$, which were used in our experiments to report computational results.

Considering the stochastic nature of our MOH algorithm, each instance was independently solved 20 times. For the purpose of fair comparisons reported in Sects. 3.4 and 3.5, we followed most reference algorithms and used a timeout limit as the stopping criterion of the MOH algorithm. The timeout limit was set to be 30 minutes for graphs with $|V| < 5000$, 120 minutes for graphs with $10{,}000 \geq |V| \geq 5000$, 240 minutes for graphs with $|V| \geq 10{,}000$.

To fully assess the performance of the MOH algorithm, we performed two comparisons with the state-of-the-art algorithms. First, we focused on the max-k-cut problem ($k = 2, 3, 4, 5$), where we thoroughly compared our algorithm with the recent discrete dynamic convexized algorithm (Zhu et al. 2013) which provides the most competitive results

[1] Our best results are available at: http://www.info.univ-angers.fr/pub/hao/maxkcut/MOHResults.zip.

[2] dfmax:ftp://dimacs.rutgers.edu/pub/dsj/clique/.

for the general max-k-cut problem in the literature. Secondly, for the special max-cut case ($k = 2$), we compared our algorithm with seven most recent max-cut algorithms (Benlic and Hao 2013; Kochenberger et al. 2013; Shylo et al. 2012; Wang et al. 2013; Wu and Hao 2012, 2013). It should be noted that those state-of-the-art max-cut algorithms were specifically designed for the particular max-cut problem while our algorithm was developed for the general max-k-cut problem. Naturally, the dedicated algorithms are advantaged since they can better explore the particular features of the max-cut problem.

3.4 Comparison with state-of-the-art max-k-cut algorithms

In this section, we present the results attained by the MOH algorithm for the max-k-cut problem. As mentioned above, we compare the proposed algorithm with the discrete dynamic convexized algorithm (DC) (Zhu et al. 2013), which was published very recently. DC was tested on a computer with a 2.11 GHz AMD processor and 1 GB of RAM. According to the Standard Performance Evaluation Cooperation (SPEC) (www.spec.org), this computer is 1.4 times slower than the computer we used for our experiments. Note that DC is the only heuristic algorithm available in the literature, which published computational results for the general max-k-cut problem.

Tables 1, 2, 3, and 4, respectively show the computational results of the MOH algorithm ($k = 2, 3, 4, 5$) on the 2 sets of benchmarks in comparison with those of the DC algorithm. The first two columns of the tables indicate the name and the number of vertices of the graphs. Columns 3 to 6 present the results attained by our algorithm, where f_{best} and f_{avg} show the best objective value and the average objective value over 20 runs, std gives the standard deviation and $time(s)$ indicates the average CPU time in seconds required by our algorithm to reach the best objective value f_{best}. Columns 7 to 10 present the statistics of the DC algorithm, including the best objective value f_{best}, average objective value f_{avg}, the time required to terminate the run $tt(s)$ and the time $bt(s)$ to reach the f_{best} value. Considering the difference between our computer and the computer used by DC, we normalize the time of DC by dividing them by 1.4 according to the SPEC mentioned above. The entries marked as "-" in the tables indicate that the corresponding results are not available. The entries in bold indicate that those results are better than the results provided by the reference DC algorithm. The last column (gap) indicates the gap of the best objective value for each instance between our algorithm and DC. A positive gap implies an improved result.

From Table 1 on max-2-cut, one observes that our algorithm achieves a better f_{best} (best objective value) for 50 out of 74 instances reported by DC, while a better f_{avg} (average objective value) for 71 out of 74 instances. Our algorithm matches the results on other instances and there is no result worse than that obtained by DC. The average standard deviation for all 91 instances is only 2.82, which shows our algorithm is stable and robust.

From Tables 2, 3, and 4, which respectively show the comparative results on max-3-cut, max-4-cut and max-5-cut. One observes that our algorithm achieves much higher solution quality on more than 90 % of 44 instances reported by DC while getting 0 worse result. Moreover, even our *average* results (f_{avg}) are better than the *best* results reported by DC.

Note that the DC algorithm used a stopping condition of 500 generations (instead of a cutoff time limit) to report its computational results. Among the two timing statistics ($tt(s)$ and $bt(s)$), $bt(s)$ roughly corresponds to column $time$ of the MOH algorithm. Still given that the two algorithms attain solutions of quite different quality, it is meaningless to directly compare the corresponding time values listed in Tables 1, 2, 3, and 4. To fairly compare the computational efficiency of MOH and DC, we reran the MOH algorithm with the best objective value of the DC algorithm as our stopping condition and reported our timing

Table 1 Comparative results for max-2-cut between the proposed MOH algorithm and DC (Zhu et al. 2013)

Instance	$\lvert V \rvert$	MOH				DC				gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	f_{avg}	$tt(s)$	$bt(s)$	
G1	800	11,624	**11,624.00**	0.00	1.46	11,624	11,617.20	131.73	90.98	0
G2	800	11,620	**11,620.00**	0.00	4.61	11,620	11,610.00	131.38	79.96	0
G3	800	11,622	**11,622.00**	0.00	1.25	11,622	11,612.20	130.78	64.22	0
G4	800	11,646	**11,646.00**	0.00	5.23	11,646	11,633.90	133.78	48.17	0
G5	800	11,631	**11,631.00**	0.00	0.99	11,631	11,623.20	131.71	36.46	0
G6	800	2178	**2178.00**	0.00	3.03	2178	2175.90	132.08	83.88	0
G7	800	2006	**2006.00**	0.00	2.98	2006	1997.70	137.61	59.61	0
G8	800	2005	**2005.00**	0.00	5.72	2005	2000.00	139.17	31.28	0
G9	800	**2054**	**2054.00**	0.00	3.21	2049	2043.50	134.94	40.03	5
G10	800	**2000**	**2000.00**	0.00	68.09	1999	1998.40	133.26	18.34	1
G11	800	564	**564.00**	0.00	0.22	564	563.80	58.84	7.78	0
G12	800	556	**556.00**	0.00	3.52	556	555.40	58.73	17.09	0
G13	800	582	**582.00**	0.00	0.85	582	580.00	60.95	43.21	0
G14	800	**3064**	**3064.00**	0.00	251.27	3057	3054.30	82.68	56.77	7
G15	800	**3050**	**3050.00**	0.00	52.19	3044	3038.00	82.43	27.69	6
G16	800	3052	**3052.00**	0.00	93.68	3052	3039.60	81.12	15.19	0
G17	800	**3047**	**3047.00**	0.00	129.53	3043	3037.80	81.61	15.05	4
G18	800	**992**	**992.00**	0.00	112.65	989	984.00	89.05	3.73	3
G19	800	906	**906.00**	0.00	266.92	906	899.90	84.43	24.96	0
G20	800	941	**941.00**	0.00	43.71	941	938.20	86.28	15.17	0
G21	800	931	**931.00**	0.00	155.34	931	926.00	86.24	12.44	0
G22	2000	**13,359**	**13,357.00**	1.91	352.37	13,339	13,315.90	683.67	108.56	20
G23	2000	**13,344**	**13,344.00**	0.00	433.79	13,323	13,298.90	705.23	433.48	21

Table 1 continued

Instance	$\|V\|$	MOH				DC				gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	f_{avg}	$tt(s)$	$bt(s)$	
G24	2000	**13,337**	**13,336.70**	0.46	777.86	13,314	13,286.00	692.07	237.38	23
G25	2000	**13,340**	**13,335.50**	2.40	442.45	13,324	13,293.70	694.73	667.19	16
G26	2000	**13,328**	**13,325.50**	2.31	535.14	13,313	13,282.20	689.61	251.36	15
G27	2000	**3341**	**3341.00**	0.00	42.25	3326	3285.40	677.86	464.32	15
G28	2000	**3298**	**3298.00**	0.00	707.18	3292	3272.00	680.47	594.81	6
G29	2000	**3405**	**3397.85**	5.31	555.23	3390	3357.20	693.45	375.90	15
G30	2000	**3413**	**3412.15**	0.36	330.46	3398	3369.50	676.54	587.80	15
G31	2000	**3310**	**3307.85**	0.91	592.56	3295	3273.90	696.42	212.48	15
G32	2000	**1410**	**1410.00**	0.00	65.75	1408	1402.70	514.87	115.58	2
G33	2000	**1382**	**1381.60**	0.80	504.10	1378	1373.70	508.85	271.75	4
G34	2000	**1384**	**1384.00**	0.00	84.23	1378	1376.70	531.51	97.37	6
G35	2000	**7686**	**7681.65**	1.59	796.70	7647	7632.20	614.51	391.36	39
G36	2000	**7680**	**7673.60**	1.62	664.48	7625	7618.50	613.15	594.82	55
G37	2000	**7691**	**7685.75**	2.26	652.78	7640	7627.70	623.72	609.25	51
G38	2000	**7688**	**7683.60**	2.27	779.69	7641	7614.40	632.95	587.98	47
G39	2000	**2408**	**2405.35**	1.85	787.69	2375	2352.50	659.34	281.45	33
G40	2000	**2400**	**2397.35**	2.43	472.50	2384	2371.70	656.75	425.90	16
G41	2000	**2405**	**2405.00**	0.00	377.35	2377	2357.40	666.79	244.21	28
G42	2000	**2481**	**2476.35**	2.01	777.42	2469	2441.30	657.13	374.11	12
G43	1000	**6660**	**6660.00**	0.00	1.15	6657	6648.90	156.66	29.04	3
G44	1000	6650	**6650.00**	0.00	5.28	6650	6643.70	155.84	24.82	0
G45	1000	**6654**	**6654.00**	0.00	6.87	6647	6640.70	155.28	95.98	7
G46	1000	**6649**	**6648.90**	0.30	67.27	6647	6637.90	157.02	61.02	2

Table 1 continued

Instance	$\|V\|$	MOH				DC				gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	f_{avg}	$tt(s)$	$bt(s)$	
G47	1000	6657	**6657.00**	0.00	43.25	6657	6648.50	157.81	144.33	0
G48	3000	6000	6000.00	0.00	0.02	6000	6000.00	420.15	0.26	0
G49	3000	6000	6000.00	0.00	0.03	6000	6000.00	440.26	0.36	0
G50	3000	5880	5879.70	0.71	532.13	5880	5880.00	552.51	0.59	0
G51	1000	**3848**	**3848.00**	0.00	189.20	3842	3831.50	137.56	122.03	6
G52	1000	**3851**	**3851.00**	0.00	209.69	3840	3830.50	132.69	119.09	11
G53	1000	**3850**	**3849.95**	0.22	299.28	3844	3835.00	136.25	62.86	6
G54	1000	**3852**	**3851.10**	0.30	190.38	3831	3824.40	136.04	60.29	21
G55	5000	10,299	10,283.40	7.13	1230.40	–	–	–	–	–
G56	5000	4016	4007.47	6.49	990.40	–	–	–	–	–
G57	5000	3494	3486.80	2.45	1528.34	–	–	–	–	–
G58	5000	19,288	19,275.40	4.58	1522.29	–	–	–	–	–
G59	5000	6087	6077.19	7.90	2498.80	–	–	–	–	–
G60	7000	14,190	14,173.00	6.98	2945.40	–	–	–	–	–
G61	7000	5798	5782.67	5.72	6603.34	–	–	–	–	–
G62	7000	4868	4851.73	7.10	5568.63	–	–	–	–	–
G63	7000	27,033	27,019.20	6.23	6492.11	–	–	–	–	–
G64	7000	8747	8700.87	19.28	4011.10	–	–	–	–	–
G65	8000	5560	5531.93	6.43	4709.53	–	–	–	–	–
G66	9000	6360	6323.53	6.34	6061.92	–	–	–	–	–
G67	10,000	6942	6903.93	8.91	4214.28	–	–	–	–	–
G70	10,000	9544	9527.80	9.32	8732.40	–	–	–	–	–
G72	10,000	6998	6957.80	7.36	6586.64	–	–	–	–	–
G77	14,000	9928	9920.00	3.08	9863.56	–	–	–	–	–

Table 1 continued

Instance	$\lvert V \rvert$	MOH				DC				gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	f_{avg}	$tt(s)$	$bt(s)$	
G81	20,000	14,036	14,020.30	8.50	20,422.00	–	–	–	–	–
3dl101000	1000	896	**896.00**	0.00	8.35	896	888.70	113.30	48.64	0
3dl102000	1000	900	**900.00**	0.00	9.50	900	898.50	111.50	2.56	0
3dl103000	1000	**892**	**892.00**	0.00	148.25	888	884.70	112.96	23.59	4
3dl104000	1000	898	**898.00**	0.00	4.20	898	895.00	112.19	30.17	0
3dl105000	1000	**886**	**886.00**	0.00	17.00	884	882.80	115.04	14.16	2
3dl106000	1000	888	**888.00**	0.00	5.55	888	883.70	114.72	32.87	0
3dl107000	1000	**900**	**899.60**	0.80	61.10	898	892.40	114.06	39.41	2
3dl108000	1000	**882**	**882.00**	0.00	76.95	880	877.70	120.03	15.83	2
3dl109000	1000	**902**	**902.00**	0.00	21.55	902	894.40	113.64	9.72	0
3dl1010000	1000	**894**	**894.00**	0.00	12.15	894	893.40	110.87	21.37	0
3dl141000	2744	**2446**	**2445.00**	1.61	552.20	2434	2416.40	1039.73	694.21	12
3dl142000	2744	**2458**	**2457.70**	1.31	479.15	2444	2431.00	1016.16	496.31	14
3dl143000	2744	**2444**	**2439.60**	2.33	58.75	2426	2415.00	1012.31	121.79	18
3dl144000	2744	**2450**	**2448.10**	2.23	220.55	2440	2425.30	997.51	587.98	10
3dl145000	2744	**2446**	**2444.90**	2.23	372.35	2432	2422.40	999.31	277.75	14
3dl146000	2744	**2452**	**2449.60**	2.06	227.80	2438	2430.00	1035.41	930.23	14
3dl147000	2744	**2444**	**2442.70**	1.31	239.05	2428	2413.40	1022.70	556.16	16
3dl148000	2744	**2448**	**2446.40**	1.50	405.35	2432	2424.40	1030.67	954.38	16
3dl149000	2744	**2428**	**2424.70**	2.12	112.05	2418	2403.70	1020.11	832.95	10
3dl1410000	2744	**2458**	**2455.70**	2.63	286.35	2438	2429.30	1018.15	466.77	20
Better		50/74/91	71/74/91							
Equal		24/74/91	3/74/91							
Worse		0/74/91	0/74/91							

Table 2 Comparative results for max-3-cut between the proposed MOH algorithm and DC Zhu et al. (2013)

Instance	$\|V\|$	MOH				DC			*gap*
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	$tt(s)$	bt(s)	
G1	800	**15,165**	15,164.90	0.36	557.25	15,127	508.34	339.41	38
G2	800	**15,172**	15,171.20	0.99	333.25	15,159	497.49	228.37	13
G3	800	**15,173**	15,173.00	0.00	269.60	15,149	506.45	205.06	24
G4	800	15,184	15,181.40	2.46	300.55	–	–	–	–
G5	800	15,193	15,193.00	0.00	98.15	–	–	–	–
G6	800	2632	2631.95	0.22	307.30	–	–	–	–
G7	800	2409	2408.40	1.07	381.00	–	–	–	–
G8	800	2428	2427.55	0.67	456.50	–	–	–	–
G9	800	2478	2475.85	2.52	282.00	–	–	–	–
G10	800	2407	2406.40	0.86	569.30	–	–	–	–
G11	800	**669**	667.80	0.75	143.80	660	240.99	132.51	9
G12	800	**660**	658.95	0.50	100.70	655	212.56	59.09	5
G13	800	**686**	685.40	0.58	459.35	679	230.20	111.53	7
G14	800	**4012**	4009.45	1.88	88.20	3984	271.47	190.40	28
G15	800	**3984**	3982.40	0.58	80.30	3960	271.88	183.92	24
G16	800	**3991**	3986.30	1.87	1.30	3958	272.44	75.02	33
G17	800	3983	3981.00	1.05	7.80	–	–	–	–
G18	800	1207	1205.60	1.56	0.30	–	–	–	–
G19	800	1081	1078.05	2.38	0.20	–	–	–	–
G20	800	1122	1115.00	4.05	13.25	–	–	–	–
G21	800	1109	1106.75	2.30	55.75	–	–	–	–
G22	2000	**17,167**	17,157.80	7.62	28.45	17008	2121.42	986.19	159
G23	2000	**17,168**	17,156.70	6.40	45.05	17021	2190.36	1208.18	147
G24	2000	**17,162**	17,152.10	4.98	16.30	17037	2230.09	1385.32	125
G25	2000	17,163	17,155.20	3.44	64.75	–	–	–	–
G26	2000	17,154	17,146.30	4.61	44.80	–	–	–	–
G27	2000	4020	4013.80	3.33	53.15	–	–	–	–
G28	2000	3973	3966.45	5.10	38.85	–	–	–	–
G29	2000	4106	4097.30	5.40	68.15	–	–	–	–
G30	2000	4119	4109.90	5.34	150.40	–	–	–	–
G31	2000	4003	3999.20	6.69	124.70	–	–	–	–
G32	2000	**1653**	1651.85	0.73	160.05	1635	1274.91	905.73	18
G33	2000	**1625**	1622.30	0.95	62.55	1603	1215.13	664.57	22
G34	2000	**1607**	1604.00	1.00	88.85	1589	1303.88	827.79	18
G35	2000	**10,046**	10,039.90	2.59	66.15	9965	1793.30	1048.97	81
G36	2000	**10,039**	10,034.40	3.81	74.25	9945	1822.04	1196.02	94
G37	2000	**10,052**	10,047.80	1.96	3.35	9952	1845.20	1288.13	100
G38	2000	10,040	10,035.50	3.26	116.60	–	–	–	–
G39	2000	2903	2890.05	6.75	8.95	–	–	–	–
G40	2000	2870	2850.65	8.08	82.80	–	–	–	–
G41	2000	2887	2862.90	9.77	87.70	–	–	–	–

Table 2 continued

Instance	$\lvert V \rvert$	MOH				DC			gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	$tt(s)$	bt(s)	
G42	2000	2980	2964.30	5.99	2.45	–	–	–	–
G43	1000	**8573**	8573.00	0.00	380.30	8510	512.48	112.20	63
G44	1000	**8571**	8569.60	2.35	616.80	8526	491.34	47.87	45
G45	1000	**8566**	8564.85	1.11	186.20	8515	504.19	44.00	51
G46	1000	8568	8564.60	2.01	215.30	–	–	–	–
G47	1000	8572	8568.70	2.72	239.35	–	–	–	–
G48	3000	**6000**	6000.00	0.00	0.40	5998	2591.27	293.30	2
G49	3000	6000	6000.00	0.00	0.90	6000	2653.42	1587.05	0
G50	3000	**6000**	6000.00	0.00	119.15	5998	2547.78	279.78	2
G51	1000	5037	5031.35	1.90	47.90	–	–	–	–
G52	1000	5040	5037.50	0.81	0.65	–	–	–	–
G53	1000	5039	5038.00	1.05	223.85	–	–	–	–
G54	1000	5036	5033.55	2.29	133.95	–	–	–	–
G55	5000	12,429	12,423.70	2.61	383.10	–	–	–	–
G56	5000	4752	4741.90	7.84	569.20	–	–	–	–
G57	5000	4083	4079.00	1.55	535.60	–	–	–	–
G58	5000	25,195	25,182.10	8.89	576.00	–	–	–	–
G59	5000	7262	7246.70	9.20	27.50	–	–	–	–
G60	7000	17,076	17,067.00	4.40	683.00	–	–	–	–
G61	7000	6853	6842.10	5.26	503.10	–	–	–	–
G62	7000	5685	5681.50	1.43	242.40	–	–	–	–
G63	7000	35,322	35,301.60	10.35	658.50	–	–	–	–
G64	7000	10,443	10,408.80	25.23	186.90	–	–	–	–
G65	8000	6490	6485.80	2.04	324.70	–	–	–	–
G66	9000	7416	7411.50	2.42	542.50	–	–	–	–
G67	10,000	8086	8083.50	2.29	756.70	–	–	–	–
G70	10,000	9999	9999.00	0.00	7.80	–	–	–	–
G72	10,000	8192	8186.70	3.35	271.20	–	–	–	–
G77	14,000	11,578	11,568.90	4.01	154.90	–	–	–	–
G81	20,000	16,321	16,313.00	4.05	331.20	–	–	–	–
3dl101000	1000	**1067**	1066.10	0.54	150.40	1043	333.45	179.20	24
3dl102000	1000	**1072**	1071.95	0.22	669.50	1044	339.38	188.68	28
3dl103000	1000	**1065**	1063.60	0.66	142.85	1042	326.69	114.20	23
3dl104000	1000	**1071**	1070.30	0.46	160.20	1045	341.58	109.75	26
3dl105000	1000	**1064**	1061.90	0.77	4.40	1039	320.88	178.88	25
3dl106000	1000	**1063**	1061.80	0.60	120.00	1032	353.75	23.96	31
3dl107000	1000	**1075**	1074.40	0.58	414.05	1053	335.95	157.18	22
3dl108000	1000	**1071**	1069.95	0.38	78.55	1049	325.50	209.77	22
3dl109000	1000	**1079**	1078.20	0.81	208.85	1052	328.38	232.87	27
3dl1010000	1000	**1070**	1069.50	0.50	478.65	1044	346.13	184.91	26
3dl141000	2744	**2924**	2919.75	2.45	25.00	2845	2527.70	1496.07	79

Table 2 continued

Instance	$\|V\|$	MOH				DC			gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	$tt(s)$	bt(s)	
3dl142000	2744	**2935**	2929.25	2.53	55.95	2856	2556.83	1408.24	79
3dl143000	2744	**2912**	2909.50	1.40	110.25	2829	2658.27	1659.44	83
3dl144000	2744	**2924**	2919.90	2.41	81.15	2861	2490.92	1759.67	63
3dl145000	2744	**2914**	2911.25	1.92	67.50	2839	2515.36	1764.88	75
3dl146000	2744	**2913**	2909.00	2.00	22.05	2834	2541.43	1529.38	79
3dl147000	2744	**2913**	2909.30	1.73	70.05	2834	2554.19	1748.39	79
3dl148000	2744	**2925**	2919.40	4.05	73.95	2845	2495.00	1440.25	80
3dl149000	2744	**2906**	2901.50	2.62	6.35	2823	2476.52	1699.97	83
3dl1410000	2744	**2933**	2927.65	2.22	29.90	2851	2519.16	1476.52	82
Better		43/44/91							
Equal		1/44/91							
Worse		0/44/91							

Table 3 Comparative results for max-4-cut between the proposed MOH algorithm and DC Zhu et al. (2013)

Instance	$\|V\|$	MOH				DC			gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	$tt(s)$	bt(s)	
G1	800	**16,803**	16,801	0.86	26.45	16,740	450.16	290.51	63
G2	800	**16,809**	16,808	1.12	268.55	16,735	455.81	388.76	74
G3	800	**16,806**	16,804.7	0.78	138.25	16,752	431.86	245.50	54
G4	800	16,814	16,811.2	1.49	146.65	–	–	–	–
G5	800	16,816	16,815.8	0.36	577.45	–	–	–	–
G6	800	2751	2748.45	1.07	89.95	–	–	–	–
G7	800	2515	2513.75	0.54	57.15	–	–	–	–
G8	800	2525	2523.35	0.65	78.6	–	–	–	–
G9	800	2585	2583.35	0.96	16.45	–	–	–	–
G10	800	2510	2507.6	1.24	79.85	–	–	–	–
G11	800	**677**	676	0.32	20.3	675	171.27	152.04	2
G12	800	**664**	662.25	0.54	41.25	660	179.99	117.52	4
G13	800	**690**	689.1	0.44	198.7	685	187.54	127.56	5
G14	800	**4440**	4435.35	1.93	55.95	4402	243.08	159.14	38
G15	800	**4406**	4403.4	0.8	89.55	4373	249.66	129.21	33
G16	800	**4415**	4414.05	1.02	392.45	4378	246.11	75.89	37
G17	800	4411	4406.45	2.27	0.2	–	–	–	–
G18	800	1261	1253.9	3.06	0.3	–	–	–	–
G19	800	1121	1115.35	3.69	1.2	–	–	–	–
G20	800	1168	1160.95	3.12	0.4	–	–	–	–
G21	800	1155	1148.25	3.74	54.7	–	–	–	–
G22	2000	**18,776**	18,765.7	5.67	107.25	18,615	1988.31	1314.45	161
G23	2000	**18,777**	18,765.8	5.71	73.7	18,612	1941.85	1775.80	165

Table 3 continued

Instance	$\|V\|$	MOH				DC			gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	$tt(s)$	bt(s)	
G24	2000	**18,769**	18,763.6	3.75	26.4	18,620	1822.82	407.66	149
G25	2000	18,775	18,767.6	4.36	75.65	–	–	–	–
G26	2000	18,767	18,761.2	4.49	96.55	–	–	–	–
G27	2000	4201	4188.5	4.6	45.35	–	–	–	–
G28	2000	4150	4138.85	5.91	24.95	–	–	–	–
G29	2000	4293	4281.65	5.68	87.4	–	–	–	–
G30	2000	4305	4296.4	4.12	33.5	–	–	–	–
G31	2000	4171	4164.4	6.46	107.8	–	–	–	–
G32	2000	**1669**	1667.85	1.01	120.9	1659	1140.66	736.15	10
G33	2000	**1638**	1634.65	1.15	0	1629	1052.38	870.96	9
G34	2000	**1616**	1611.7	1.65	0.05	1604	1105.02	1016.31	12
G35	2000	**11,111**	11,106.2	2.14	17.2	11,007	1890.32	1764.52	104
G36	2000	**11,108**	11,101.4	2.9	17.25	10,993	1738.64	1634.13	115
G37	2000	**11,117**	11,112.5	2.33	36.05	11023	1754.17	115.08	94
G38	2000	11,108	11,101.1	3.16	48.4	–	–	–	–
G39	2000	3006	2998.7	3.91	1.15	–	–	–	–
G40	2000	2976	2955.65	8.99	48.7	–	–	–	–
G41	2000	2983	2970.3	6.91	1.8	–	–	–	–
G42	2000	3092	3084.05	4.8	16.9	–	–	–	–
G43	1000	**9376**	9373.95	1.2	84.15	9306	422.97	62.38	70
G44	1000	**9379**	9373.55	2.52	67.9	9315	430.52	43.88	64
G45	1000	**9376**	9375.1	0.94	249.5	9312	463.45	319.58	64
G46	1000	9378	9375.35	1.96	139.75	–	–	–	–
G47	1000	9381	9377.05	2.04	60.5	–	–	–	–
G48	3000	6000	6000	0	0	6000	1673.79	0.48	0
G49	3000	6000	6000	0	0	6000	1675.56	0.49	0
G50	3000	6000	6000	0	0	6000	1678.91	0.50	0
G51	1000	5571	5567.65	1.93	14.6	–	–	–	–
G52	1000	5584	5581.15	1.74	20.9	–	–	–	–
G53	1000	5574	5571.85	1.19	6.85	–	–	–	–
G54	1000	5579	5576.25	1.58	0.7	–	–	–	–
G55	5000	12,498	12,498	0	0.9	–	–	–	–
G56	5000	4931	4917.1	6.49	424.6	–	–	–	–
G57	5000	4112	4110.5	1.12	298.1	–	–	–	–
G58	5000	27,885	27,870.9	8.68	435.4	–	–	–	–
G59	5000	7539	7515.1	15.09	969.3	–	–	–	–
G60	7000	17,148	17,148	0	2.3	–	–	–	–
G61	7000	7110	7104.6	5.08	1305.2	–	–	–	–
G62	7000	5743	5738.7	2.69	385.5	–	–	–	–
G63	7000	39,083	39,063.5	9.18	660.2	–	–	–	–
G64	7000	10,814	10,797.4	13.28	910.5	–	–	–	–

Table 3 continued

Instance	$\|V\|$	MOH				DC			gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	$tt(s)$	bt(s)	
G65	8000	6534	6525.4	4.48	1.5	–	–	–	–
G66	9000	7474	7467.8	4.24	2.2	–	–	–	–
G67	10,000	8155	8142.5	5.57	3	–	–	–	–
G70	10,000	9999	9999	0	0.5	–	–	–	–
G72	10,000	8264	8254.6	7.36	3.1	–	–	–	–
G77	14,000	11,674	11,658.9	10.08	6.4	–	–	–	–
G81	20,000	16,470	16,454.3	8.5	27.9	–	–	–	–
3dl101000	1000	**1103**	1100.6	0.86	64.5	1073	304.44	187.92	30
3dl102000	1000	**1102**	1100	0.95	1.5	1070	351.27	301.64	32
3dl103000	1000	**1108**	1106.4	0.86	22.8	1072	340.99	249.06	36
3dl104000	1000	**1103**	1101.65	0.65	87.7	1076	323.51	276.29	27
3dl105000	1000	**1098**	1096.3	0.78	58.6	1074	334.38	294.70	24
3dl106000	1000	**1097**	1095.15	0.91	94.05	1063	358.27	307.91	34
3dl107000	1000	**1114**	1112.2	1.08	108.3	1093	308.31	101.66	21
3dl108000	1000	**1105**	1103	0.77	28.9	1079	276.09	260.12	26
3dl109000	1000	**1115**	1113.45	0.8	108.35	1086	271.29	60.70	29
3dl1010000	1000	**1109**	1106.1	0.89	54.9	1088	277.18	257.21	21
3dl141000	2744	**3016**	3012.05	1.91	57.05	2893	1990.54	1511.84	123
3dl142000	2744	**3026**	3019.8	2.04	18.45	2893	2007.26	464.84	133
3dl143000	2744	**3006**	3001.7	2.88	37.2	2892	1956.09	1339.53	114
3dl144000	2744	**3012**	3007.85	1.85	47.8	2897	1980.32	1923.14	115
3dl145000	2744	**3006**	3001.2	2.16	58.1	2882	1972.18	1866.67	124
3dl146000	2744	**3005**	3001.35	1.46	14	2888	1948.91	1892.88	117
3dl147000	2744	**3007**	3001.95	2.31	30.5	2879	1995.73	1983.25	128
3dl148000	2744	**3018**	3014.5	1.96	165.45	2883	1982.66	1914.45	135
3dl149000	2744	**2999**	2993.95	2.62	20	2877	2024.45	1769.77	122
3dl1410000	2744	**3023**	3021.15	1.68	389.4	2904	2007.36	2003.40	119
Better		41/44/91							
Equal		3/44/91							
Worse		0/44/91							

statistics in Table 5. One observes that our algorithm needs at most 16 seconds (less than 1 second for most cases) to attain the best objective value reported by the DC algorithm, while the DC algorithm requires at least 44 seconds and up to more than 2000 seconds for several instances. More generally, as shown in Tables 1, 2, 3, and 4, except the last 17 instances of the very competitive max-2-cut problem for which the results of DC are not available, the MOH algorithm requires rarely more than 1000 seconds to attain solutions of much better quality.

We conclude that the proposed algorithm for the general max-k-cut problem dominates the state-of-the-art reference DC algorithm both in terms of solution quality and computing time.

Table 4 Comparative results for max-5-cut between the proposed MOH algorithm and DC Zhu et al. (2013)

Instance	$\|V\|$	MOH				DC			*gap*
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	$tt(s)$	bt(s)	
G1	800	**17,703**	17,700.80	1.18	76.40	17,627	532.14	376.14	76
G2	800	**17,706**	17,702.50	1.63	122.20	17,636	537.26	288.13	70
G3	800	**17,701**	17,699.20	1.47	210.20	17,623	525.92	357.24	78
G4	800	17,709	17,706.50	1.75	141.20	–	–	–	–
G5	800	17,710	17,708.60	1.66	269.70	–	–	–	–
G6	800	2781	2776.00	2.26	146.20	–	–	–	–
G7	800	2533	2530.75	2.00	56.50	–	–	–	–
G8	800	2535	2532.75	1.13	105.00	–	–	–	–
G9	800	2601	2598.65	1.28	6.55	–	–	–	–
G10	800	2526	2520.00	4.18	143.70	–	–	–	–
G11	800	**677**	675.40	0.58	0.00	670	239.03	147.55	7
G12	800	**662**	661.40	0.49	153.10	660	240.87	191.89	2
G13	800	**689**	688.40	0.49	317.15	687	222.88	177.50	2
G14	800	**4639**	4634.60	1.83	37.65	4597	297.49	63.30	42
G15	800	**4606**	4599.90	1.79	80.05	4571	293.47	99.68	35
G16	800	**4613**	4610.30	1.31	94.60	4579	291.25	243.93	34
G17	800	4603	4600.85	1.01	96.50	–	–	–	–
G18	800	1268	1261.85	3.48	0.05	–	–	–	–
G19	800	1132	1122.45	7.08	0.10	–	–	–	–
G20	800	1172	1163.90	4.73	0.35	–	–	–	–
G21	800	1162	1153.50	5.34	0.05	–	–	–	–
G22	2000	**19,553**	19547.00	3.64	42.40	19,413	2429.87	1685.57	140
G23	2000	**19,558**	19549.20	4.04	85.40	19,413	2422.00	2248.13	145
G24	2000	**19,555**	19547.20	2.93	88.55	19,423	2255.39	1668.64	132
G25	2000	19,554	19547.80	3.18	140.35	–	–	–	–
G26	2000	19,552	19545.00	2.80	85.00	–	–	–	–
G27	2000	4236	4224.30	6.23	143.10	–	–	–	–
G28	2000	4182	4171.45	6.84	65.10	–	–	–	–
G29	2000	4327	4317.50	4.25	72.85	–	–	–	–
G30	2000	4340	4329.75	4.44	50.45	–	–	–	–
G31	2000	4211	4196.40	7.89	37.40	–	–	–	–
G32	2000	**1670**	1666.45	1.94	0.75	1647	1304.51	1272.00	23
G33	2000	**1638**	1635.05	1.20	0.20	1615	1194.92	678.48	23
G34	2000	**1615**	1610.20	2.84	0.40	1594	1232.62	629.56	21
G35	2000	**11,605**	11,595.20	4.15	68.80	11,521	2030.16	961.14	84
G36	2000	**11,601**	11,593.80	3.03	12.25	11,516	2074.70	510.45	85
G37	2000	**11,603**	11,599.40	2.46	70.15	11,532	2026.00	1661.50	71
G38	2000	11,601	11,596.20	3.19	163.65	–	–	–	–
G39	2000	3022	3014.35	5.32	70.15	–	–	–	–
G40	2000	2986	2967.20	9.45	0.50	–	–	–	–
G41	2000	2986	2972.85	7.84	20.05	–	–	–	–

Table 4 continued

Instance	$\|V\|$	MOH				DC			*gap*
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	$tt(s)$	bt(s)	
G42	2000	3109	3099.15	5.29	0.60	–	–	–	–
G43	1000	**9770**	9767.30	1.38	56.50	9700	583.20	76.61	70
G44	1000	**9772**	9768.05	1.60	16.85	9702	518.05	482.50	70
G45	1000	**9771**	9768.10	1.30	25.60	9708	502.37	470.51	63
G46	1000	9774	9769.55	1.66	47.80	–	–	–	–
G47	1000	9775	9770.05	1.86	60.70	–	–	–	–
G48	3000	6000	6000.00	0.00	0.00	6000	1871.21	0.50	0
G49	3000	6000	6000.00	0.00	0.00	6000	1864.70	0.48	0
G50	3000	6000	6000.00	0.00	0.00	6000	1887.36	0.50	0
G51	1000	5826	5822.30	2.05	0.75	–	–	–	–
G52	1000	5837	5832.35	1.68	4.90	–	–	–	–
G53	1000	5829	5825.90	1.09	55.75	–	–	–	–
G54	1000	5830	5826.70	1.42	28.40	–	–	–	–
G55	5000	12,498	12,498.00	0.00	0.00	–	–	–	–
G56	5000	4971	4957.90	8.75	243.70	–	–	–	–
G57	5000	4111	4108.70	1.19	293.50	–	–	–	–
G58	5000	29,105	29,090.70	9.28	272.10	–	–	–	–
G59	5000	7566	7541.20	19.22	120.40	–	–	–	–
G60	7000	17,148	17,148.00	0.00	0.00	–	–	–	–
G61	7000	7188	7174.50	7.74	437.60	–	–	–	–
G62	7000	5744	5736.90	2.88	4.20	–	–	–	–
G63	7000	40,786	40,767.50	10.50	420.80	–	–	–	–
G64	7000	10,896	10,851.50	23.04	48.60	–	–	–	–
G65	8000	6540	6528.90	4.93	8.50	–	–	–	–
G66	9000	7476	7470.60	4.74	10.90	–	–	–	–
G67	10,000	8165	8151.60	7.32	8.20	–	–	–	–
G70	10,000	9999	9999.00	0.00	0.10	–	–	–	–
G72	10,000	8266	8256.00	6.74	8.60	–	–	–	–
G77	14,000	11,687	11,672.10	11.41	21.10	–	–	–	–
G81	20,000	16,501	16,480.20	10.06	271.50	–	–	–	–
3dl101000	1000	**1106**	1102.95	1.50	38.00	1073	321.44	79.97	33
3dl102000	1000	**1106**	1103.50	1.12	51.95	1067	358.55	78.05	39
3dl103000	1000	**1111**	1106.95	1.86	74.10	1072	343.13	106.00	39
3dl104000	1000	**1108**	1105.65	0.91	44.00	1076	330.08	223.84	32
3dl105000	1000	**1098**	1096.15	1.01	76.90	1074	327.13	197.17	24
3dl106000	1000	**1099**	1097.55	0.92	48.25	1071	329.38	304.61	28
3dl107000	1000	**1119**	1115.85	1.62	48.80	1084	321.82	230.50	35
3dl108000	1000	**1113**	1110.70	1.27	126.30	1077	333.74	147.03	36
3dl109000	1000	**1119**	1117.30	0.84	17.85	1089	327.09	186.92	30
3dl1010000	1000	**1115**	1114.10	0.83	336.95	1081	330.26	301.70	34
3dl141000	2744	**3029**	3022.00	3.51	4.15	2912	2416.83	1114.20	117

Table 4 continued

Instance	$\|V\|$	MOH				DC			*gap*
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	$tt(s)$	bt(s)	
3dl142000	2744	**3033**	3025.75	3.73	58.40	2916	2665.55	1512.49	117
3dl143000	2744	**3015**	3007.75	5.23	100.10	2891	2568.33	706.35	124
3dl144000	2744	**3021**	3015.95	2.65	30.85	2914	2658.98	2066.46	107
3dl145000	2744	**3014**	3005.25	2.90	7.45	2897	2405.89	2252.09	117
3dl146000	2744	**3013**	3010.05	2.22	102.50	2906	2363.11	2227.79	107
3dl147000	2744	**3016**	3009.55	4.17	85.60	2900	2536.90	257.75	116
3dl148000	2744	**3027**	3022.70	2.12	12.85	2920	2376.40	2127.40	107
3dl149000	2744	**3005**	2994.15	4.15	0.25	2901	2711.61	2687.12	104
3dl1410000	2744	**3033**	3023.25	3.78	17.75	2917	2432.17	1767.87	116
Better		41/44/91							
Equal		3/44/91							
Worse		0/44/91							

3.5 Comparison with state-of-the-art max-cut algorithms

Our algorithm was designed for the general max-k-cut problem for $k \geq 2$. The assessment of the last section focused on the general case. In this section, we further evaluate the performance of the proposed algorithm for the special max-cut problem ($k = 2$).

Recall that max-cut has been largely studied in the literature for a long time and there are many powerful heuristics which are specifically designed for the problem. These state-of-the-art max-cut algorithms constitute thus relevant references for our comparative study. In particular, we adopt the following 7 best performing sequential algorithms published since 2012.

1. Global equilibrium search (GES) (2012) (Shylo et al. 2012)—an algorithm sharing ideas similar to simulated annealing and utilizing accumulated information of search space to generate new solutions for the subsequent stages. The reported results of GES were obtained on a PC with a 2.83GHz Intel Core QUAD Q9550 CPU and 8.0GB RAM.
2. Breakout local search (BLS) (2013) (Benlic and Hao 2013)—a heuristic algorithm integrating a local search and adaptive perturbation strategies. The reported results of BLS were obtained on a PC with 2.83GHz Intel Xeon E5440 CPU and 2GB RAM.
3. Two memetic algorithms respective for the max-cut problem (MACUT) (2012) (Wu and Hao 2012) and the max-bisection problem (MAMBP) (2013) (Wu and Hao 2013)—integrating a grouping crossover operator and a tabu search procedure. The results reported in the two papers were obtained on a PC with a 2.83GHz Intel Xeon E5440 CPU and 2GB RAM.
4. GRASP-Tabu search algorithm (2013) (Wang et al. 2013)—a method converting the max-cut problem to the UBQP problem and solving it by integrating GRASP and tabu search. The reported results were obtained on a PC with a 2.83GHz Intel Xeon E5440 CPU and 2GB RAM.
5. Tabu search (TS-UBQP) (2013) (Kochenberger et al. 2013)—a tabu search algorithm designed for UBQP. The evaluation of TS-UBQP were performed on a PC with a 2.83GHz Intel Xeon E5440 CPU and 2GB RAM.

Table 5 Average computing time needed by the MOH algorithm (MOH(tavg)) to attain the best objective value of the DC algorithm (Zhu et al. 2013). The time required by DC (DC(t)) to reach the same objective value is also included

Instance	max-3-cut		max-4-cut		max-5-cut	
	DC(t)	MOH(tavg)	DC(t)	MOH(tavg)	DC(t)	MOH(tavg)
G1	339.41	0.16	290.51	0.18	376.14	0.01
G2	228.37	2.05	388.76	0.12	288.13	0.01
G3	205.06	0.35	245.50	0.24	357.24	0.01
G11	132.51	0.11	152.04	6.67	147.55	8.39
G12	59.09	2.11	117.52	6.65	191.89	16.02
G13	111.53	0.29	127.56	0.68	177.50	0.29
G14	190.40	0.09	159.14	0.13	63.30	0.01
G15	183.92	0.12	129.21	0.16	99.68	0.00
G16	75.02	0.08	75.89	0.09	243.93	0.01
G22	986.19	0.06	1314.45	0.09	1685.57	0.01
G23	1208.18	0.05	1775.80	0.08	2248.13	0.01
G24	1385.32	0.10	407.66	0.10	1668.64	0.01
G32	905.73	0.37	736.15	0.36	1272.00	2.00
G33	664.57	0.27	870.96	1.50	678.48	5.16
G34	827.79	0.31	1016.31	1.64	629.56	1.58
G35	1048.97	0.24	1764.52	0.10	961.14	0.00
G36	1196.02	0.13	1634.13	0.09	510.45	0.00
G37	1288.13	0.09	115.08	0.13	1661.50	0.00
G43	112.20	0.06	62.38	0.05	76.61	0.01
G44	47.87	0.09	43.88	0.08	482.50	0.01
G45	44.00	0.07	319.58	0.07	470.51	0.01
G48	293.30	0.52	0.48	0.01	0.50	0.00
G49	1587.05	0.53	0.49	0.01	0.48	0.00
G50	279.78	4.36	0.50	0.01	0.50	0.00
sg3dl101000	179.20	0.06	187.92	0.06	79.97	0.05
sg3dl102000	188.68	0.05	301.64	0.05	78.05	0.03
sg3dl103000	114.20	0.09	249.06	0.05	106.00	0.03
sg3dl104000	109.75	0.07	276.29	0.05	223.84	0.05
sg3dl105000	178.88	0.07	294.70	0.10	197.17	0.06
sg3dl106000	23.96	0.03	307.91	0.04	304.61	0.05
sg3dl107000	157.18	0.08	101.66	0.17	230.50	0.05
sg3dl108000	209.77	0.06	260.12	0.10	147.03	0.05
sg3dl109000	232.87	0.07	60.70	0.07	186.92	0.06
sg3dl1010000	184.91	0.05	257.21	0.14	301.70	0.04
sg3dl141000	1496.07	0.14	1511.84	0.05	1114.20	0.07
sg3dl142000	1408.24	0.14	464.84	0.04	1512.49	0.07
sg3dl143000	1659.44	0.11	1339.53	0.07	706.35	0.06
sg3dl144000	1759.67	0.25	1923.14	0.05	2066.46	0.09
sg3dl145000	1764.88	0.15	1866.67	0.05	2252.09	0.08
sg3dl146000	1529.38	0.12	1892.88	0.05	2227.79	0.07

Table 5 continued

Instance	max-3-cut		max-4-cut		max-5-cut	
	DC(t)	MOH(tavg)	DC(t)	MOH(tavg)	DC(t)	MOH(tavg)
sg3dl147000	1748.39	0.12	1983.25	0.05	257.75	0.07
sg3dl148000	1440.25	0.13	1914.45	0.05	2127.40	0.10
sg3dl149000	1699.97	0.14	1769.77	0.06	2687.12	0.11
sg3dl1410000	1476.52	0.11	2003.40	0.06	1767.87	0.07

6. Tabu search based hybrid evolutionary algorithm (TSHEA) (2016) (Wu et al. 2015)—a very recent hybrid algorithm integrating a distance-and-quality guided solution combination operator and a tabu search procedure based on neighborhood combination of one-flip and constrained exchange moves. The results were obtained on a PC with 2.83GHz Intel Xeon E5440 CPU and 8GB RAM.

One notices that except GES, the other five reference algorithms were run on the same computing platform. Nevertheless, it is still difficult to make a fully fair comparison of the computing time, due to the differences on programming language, compiling options, and termination conditions, etc. Our comparison thus focuses on the best solution achieved by each algorithm. Recall that for our algorithm, the timeout limit was set to be 30 minutes for graphs with $|V| < 5000$, 120 minutes for graphs with $1000 \geq |V| \geq 5000$, 240 minutes for graphs with $|V| \geq 10{,}000$. Our algorithm employed thus the same timeout limits as (Wu and Hao 2012) on the graphs $|V| < 10{,}000$, but for the graphs $|V| \geq 10{,}000$, we used 240 minutes to compare with BLS Benlic and Hao (2013).

Table 6 gives the comparative results on the 91 instances of the two benchmarks. Columns 1 and 2 respectively indicate the instance name and the number of vertices of the graphs. Columns 3 shows the current best known objective value f_{pre} reported by any existing max-cut algorithm in the literature including the latest *parallel* GES algorithm (Shylo et al. 2015). Columns 4 to 10 give the best objective value obtained by the reference algorithms: GES (Shylo et al. 2012), BLS (Benlic and Hao 2013), MACUT (Wu and Hao 2012), TS-UBQP (Kochenberger et al. 2013), GRASP-TS/PM (Wang et al. 2013), MAMBP (Wu and Hao 2013) and TSHEA (Wu et al. 2015). Note that MAMBP is designed for the max-bisection problem (i.e., balanced max-cut), however it achieves some previous best known max-cut results. The last column 'MOH' recalls the best results of our algorithm from Table 1. The rows denoted by 'Better', 'Equal' and 'Worse' respectively indicate the number of instances for which our algorithm obtains a result of better, equal and worse quality relative to each reference algorithm. The entries are reported in the form of x/y/z, where z denotes the total number of the instances tested by our algorithm, y is the number of the instances tested by a reference algorithm and x indicates the number of instances where our algorithm achieved 'Better', 'Equal' or 'Worse' results. The results in bold mean that our algorithm has improved the best known results. The entries marked as "–" in the table indicate that the results are not available.

From Table 6, one observes that the MOH algorithm is able to improve the current best known results in the literature for 4 instances, and match the best known results for 74 instances. For 13 cases (in italic), even if our results are worse than the current best known results achieved by the latest *parallel* GES algorithm (Shylo et al. 2015), they are still better than the results of other existing algorithms, except for 4 instances if we refer to the most recent

Table 6 Comparative results of the proposed MOH algorithm with 7 state-of-the-art max-cut algorithms

Instance	$\|V\|$	f_{pre}	GES (Shylo et al. 2012)	BLS (Benlic and Hao 2013)	MACUT (Wu and Hao 2012)	TS-UBQP (Kochenberger et al. 2013)	TS/PM (Wang et al. 2013)	MAMBP (Wu and Hao 2013)	TSHEA (Wu et al. 2015)	MOH
G1	800	11,624	11,624	11,624	11,624	11,624	11,624	11,624	11,624	11,624
G2	800	11,620	11,620	11,620	11,620	11,620	11,620	11,617	11,620	11,620
G3	800	11,622	11,622	11,622	11,622	11,620	11,620	11,621	11,622	11,622
G4	800	11,646	11,646	11,646	–	11,646	11,646	11,646	11,646	11,646
G5	800	11,631	11,631	11,631	–	11,631	11,631	11,631	11,631	11,631
G6	800	2178	2178	2178	–	2178	2178	2177	2178	2178
G7	800	2006	2006	2006	–	2006	2006	2002	2006	2006
G8	800	2005	2005	2005	–	2005	2005	2004	2005	2005
G9	800	2054	2054	2054	–	2054	2054	2052	2054	2054
G10	800	2000	2000	2000	–	2000	2000	1998	2000	2000
G11	800	564	564	564	564	564	564	564	564	564
G12	800	556	556	556	556	556	556	556	556	556
G13	800	582	582	582	582	580	582	582	582	582
G14	800	3064	3064	3064	3064	3061	3063	3062	3064	3064
G15	800	3050	3050	3050	3050	3050	3050	3050	3050	3050
G16	800	3052	3052	3052	3052	3052	3052	3052	3052	3052
G17	800	3047	3047	3047	–	3046	3047	3047	3047	3047
G18	800	992	992	992	–	991	992	992	992	992
G19	800	906	906	906	–	904	906	905	906	906
G20	800	941	941	941	–	941	941	941	941	941
G21	800	931	931	931	–	930	931	930	931	931
G22	2000	13,359	13,359	13,359	13,359	13,359	13,349	13,359	13,359	13,359
G23	2000	13,344	13,342	13,344	13,344	13,342	13,332	13,344	13,344	13,344

Table 6 continued

Instance	$\|V\|$	f_{pre}	GES (Shylo et al. 2012)	BLS (Benlic and Hao 2013)	MACUT (Wu and Hao 2012)	TS-UBQP (Kochenberger et al. 2013)	TS/PM (Wang et al. 2013)	MAMBP (Wu and Hao 2013)	TSHEA (Wu et al. 2015)	MOH
G24	2000	13,337	13,337	13,337	13,337	13,337	13,324	13,336	13,337	13,337
G25	2000	13,340	13,340	13,340	–	13,332	13,326	13,340	13,340	13,340
G26	2000	13,328	13,328	13,328	–	13,328	13,313	13,328	13,328	13,328
G27	2000	3341	3341	3341	–	3336	3325	3341	3341	3341
G28	2000	3298	3298	3298	–	3295	3287	3298	3298	3298
G29	2000	3405	3405	3405	–	3391	3394	3403	3405	3405
G30	2000	3413	3413	3412	–	3403	3402	3412	3413	3413
G31	2000	3310	3310	3309	–	3288	3299	3309	3310	3310
G32	2000	1410	1410	1410	1410	1406	1406	1410	1410	1410
G33	2000	1382	1382	1382	1382	1378	1374	1382	1382	1382
G34	2000	1384	1384	1384	1384	1378	1376	1384	1384	1384
G35	2000	7687	7686	7684	7686	7678	7661	7686	7687	7687
G36	2000	7680	7680	7678	7679	7670	7660	7678	7680	7680
G37	2000	7691	7691	7689	7690	7682	7670	7689	7691	7691
G38	2000	7688	7687	7687	–	7683	7670	7688	7688	7688
G39	2000	2408	2408	2408	–	2397	2397	2408	2408	2408
G40	2000	2400	2400	2400	–	2390	2392	2400	2400	2400
G41	2000	2405	2405	2405	–	2400	2398	2405	2405	2405
G42	2000	2481	2481	2481	–	2469	2474	2481	2481	2481
G43	1000	6660	6660	6660	6660	6660	6660	6659	6660	6660
G44	1000	6650	6650	6650	6650	6639	6649	6650	6650	6650
G45	1000	6654	6654	6654	6654	6652	6654	6654	6654	6654

Table 6 continued

Instance	$\|V\|$	f_{pre}	GES (Shylo et al. 2012)	BLS (Benlic and Hao 2013)	MACUT (Wu and Hao 2012)	TS-UBQP (Kochenberger et al. 2013)	TS/PM (Wang et al. 2013)	MAMBP (Wu and Hao 2013)	TSHEA (Wu et al. 2015)	MOH
G46	1000	6649	6649	6649	–	6649	6649	6649	6649	6649
G47	1000	6657	6657	6657	–	6656	6656	6657	6657	6657
G48	3000	6000	6000	6000	6000	6000	6000	6000	6000	6000
G49	3000	6000	6000	6000	6000	6000	6000	6000	6000	6000
G50	3000	5880	5880	5880	5800	5880	5880	5880	5880	5880
G51	1000	3848	3848	3848	–	3847	3847	3847	3848	3848
G52	1000	3851	3851	3851	–	3849	3850	3851	3851	3851
G53	1000	3850	3850	3850	–	3848	3848	3850	3850	3850
G54	1000	3852	3852	3852	–	3851	3850	3851	3852	3852
G55	5000	10,299	–	10,294	10,299	10,236	–	10,299	10,299	10,299
G56	5000	4017	–	4012	4016	3934	–	4016	4017	*4016*
G57	5000	3494	–	3492	–	3460	–	3488	3494	3494
G58	5000	19,293	–	19,263	–	19,248	–	19,276	19,276	*19,288*
G59	5000	6086	–	6078	–	6019	–	6085	6085	**6087**
G60	7000	14,188	–	14,176	14,186	14,057	–	14,186	14,186	**14,190**
G61	7000	5796	–	5789	–	5680	–	5796	5796	**5798**
G62	7000	4870	–	4868	–	4822	–	4866	4866	*4868*
G63	7000	27,045	–	26,997	–	26,963	–	26,754	27,018	*27,033*
G64	7000	8751	–	8735	–	8610	–	8731	8735	*8747*
G65	8000	5562	–	5558	5550	5518	–	5556	5560	*5560*
G66	9000	6364	–	6360	6352	6304	–	6352	6364	*6360*

Table 6 continued

Instance	$\|V\|$	f_{pre}	GES (Shylo et al. 2012)	BLS (Benlic and Hao 2013)	MACUT (Wu and Hao 2012)	TS-UBQP (Kochenberger et al. 2013)	TS/PM (Wang et al. 2013)	MAMBP (Wu and Hao 2013)	TSHEA (Wu et al. 2015)	MOH
G67	10,000	6950	–	6940	6934	6894	–	6934	6944	*6942*
G70	10,000	9591	–	9541	–	9458	–	9580	9548	*9544*
G72	10,000	7006	–	6998	–	6922	–	6990	6990	*6998*
G77	14,000	9938	–	9926	–	–	–	9900	9902	*9928*
G81	20,000	14,048	–	14,030	–	–	–	13,978	14,010	*14,036*
3dl101000	1000	896	896	–	–	–	–	–	896	896
3dl102000	1000	900	900	–	–	–	–	–	900	900
3dl103000	1000	892	892	–	–	–	–	–	892	892
3dl104000	1000	898	898	–	–	–	–	–	898	898
3dl105000	1000	886	886	–	–	–	–	–	886	886
3dl106000	1000	888	888	–	–	–	–	–	888	888
3dl107000	1000	900	900	–	–	–	–	–	900	900
3dl108000	1000	882	882	–	–	–	–	–	882	882
3dl109000	1000	902	902	–	–	–	–	–	902	902
3dl1010000	1000	894	894	–	–	–	–	–	894	894
3dl141000	2744	2446	2446	–	–	–	–	–	2446	2446
3dl142000	2744	2458	2458	–	–	–	–	–	2458	2458
3dl143000	2744	2442	2442	–	–	–	–	–	2442	**2444**
3dl144000	2744	2450	2450	–	–	–	–	–	2450	2450
3dl145000	2744	2446	2446	–	–	–	–	–	2446	2446
3dl146000	2744	2452	2452	–	–	–	–	–	2452	2452

Table 6 continued

Instance	$\|V\|$	f_{pre}	GES (Shylo et al. 2012)	BLS (Benlic and Hao 2013)	MACUT (Wu and Hao 2012)	TS-UBQP (Kochenberger et al. 2013)	TS/PM (Wang et al. 2013)	MAMBP (Wu and Hao 2013)	TSHEA (Wu et al. 2015)	MOH
3dl147000	2744	2444	2444	–	–	–	–	–	2444	2444
3dl148000	2744	2448	2448	–	–	–	–	–	2448	2448
3dl149000	2744	2428	2426	–	–	–	–	–	2428	2428
3dl1410000	2744	2460	2458	–	–	–	–	–	2460	*2458*
Better		4/91/91	4/74/91	20/71/91	7/30/91	47/69/91	29/54/91	33/71/91	11/91/91	
Equal		74/91/91	70/74/91	51/71/91	23/30/91	22/69/91	25/54/91	37/71/91	75/91/91	
Worse		13/91/91	0/74/91	0/71/91	0/30/91	0/69/91	0/54/91	1/71/90	5/91/91	

TSHEA algorithm (Wu et al. 2015). Note that the results of the parallel GES algorithm were achieved on a more powerful computing platform (Intel CoreTM i7-3770 CPU @3.40GHz and 8GB RAM) and with longer time limits (4 parallel processes at the same time and 1 hour for each process).

Such a performance is remarkable given that we are comparing our more general algorithm designed for max-k-cut with the best performing specific max-cut algorithms. The experimental evaluations presented in this section and last section demonstrate that our algorithm not only performs well on the general max-k-cut problem, but also remains highly competitive for the special case of the popular max-cut problem.

4 Discussion

In this section, we investigate the role of several important ingredients of the proposed algorithm, including the bucket sorting data structure, the descent improvement search operators O_1 and O_2 and the diversified improvement search operators O_3 and O_4.

4.1 Impact of the bucket sorting technique

As described in Sect. 2.5, the bucket sorting technique is utilized in the MOH algorithm for the purpose of quickly identifying a suitable move with the best objective gain. To verify its effectiveness, we implemented another MOH version where we replaced the bucket sorting data structure with a simple vector and conducted an experimental comparison on the max-3-cut problem. For this experiment, we used 20 representative Gxx instances and ran 20 times both MOH versions to solve each chosen instance with a time limit of 300 seconds.

Table 7 reports the average of the best objective values and the total number of iterations of each MOH version for each instance. From Table 7, we observe that the MOH algorithm using the bucket sorting structure conducted 3.3 times more iterations on average than using the vector structure within the given time span. Moreover, the former is able to find better results for 16 instances and only one worse result. In conclusion, this experiment confirms that using the devised bucket sorting technique is able to considerably improve the computational efficiency and search capacity of the MOH algorithm.

4.2 Impact of the descent improvement search operators

As described in Sect. 2.6, the proposed algorithm employs operators O_1 and O_2 for its descent improvement phase to obtain local optima. To analyze the impact of these two operators, we implement three variants of our algorithm, the first one using the operator O_1 alone, the second one using the union $O_1 \cup O_2$ such that the descent search procedure always chooses the best move among the O_1 *and* O_2 moves (Lü et al. 2011), the third one using operator $rand(O_1, O_2)$ where the descent procedure applies randomly and with equal probability O_1 or O_2, while keeping all the other ingredients and parameters fixed as described in Sect. 3.3. The strategy used by our original algorithm, detailed in Sect. 2.6, is denoted as $O_1 + O_2$.

This study was based on the max-cut problem and the same 10 challenging instances used for parameter tuning of Sect. 3.3. Each selected instance was solved 10 times by each of these variants and our original algorithm. The stopping criterion was a timeout limit of 30 minutes. The obtained results are presented in Table 8, including the best objective value f_{best}, the average objective value f_{avg} over the 10 independent runs, as well as the CPU times in seconds to reach f_{best}. To evaluate the performance, we display in Fig. 2a the gaps between the best

Table 7 Computational assessment of bucket sorting compared to an implementation using a vector applied to the max-3-cut problem

Instance	Bucket sorting structure		Vector structure		Differences	
	f_{bss}	$iter_{bss}$	f_{vs}	$iter_{vs}$	$f_{bss} - f_{vs}$	$iter_{bss}/iter_{vs}$
G22	17,135.65	87,068,095.55	17,132.7	55,940,769.45	2.95	1.56
G26	17,128.1	89,044,944.75	17,121.65	50,698,801.15	6.45	1.76
G28	3943.4	81,621,472.45	3942.9	49,226,453.00	0.5	1.66
G30	4091.95	89,369,709.35	4095.85	52,714,888.95	−3.9	1.70
G32	1654.85	212,255,042.05	1652.75	59,712,070.05	2.1	3.55
G34	1605.4	216,409,597.50	1604.2	51,582,268.90	1.2	4.20
G36	10,024.1	136,113,904.60	10,015	48,257,118.45	9.1	2.82
G38	10,027.1	147,998,869.05	10,021.5	53,182,934.85	5.6	2.78
G40	2841.85	137,242,801.85	2831.75	53,555,508.15	10.1	2.56
G44	8556.75	99,472,399.80	8557.1	102,758,227.95	−0.35	0.97
G46	8555.1	100,453,139.40	8555.35	100,251,434.60	−0.25	1.00
G54	5028.65	170,660,709.15	5026.9	98,723,794.70	1.75	1.73
G56	4709.05	105,834,778.80	4662.45	14,561,723.95	46.6	7.27
G58	25,144.4	88,340,858.10	25,092.5	14,574,161.75	51.9	6.06
G60	17,019.6	37,339,981.15	16,963.55	8,873,616.55	56.05	4.21
G62	5685.7	101,427,430.65	5656.7	9,955,135.45	29	10.19
G64	10,318.1	68,975,406.10	10,175.75	8,846,430.90	142.35	7.80
G66	7417.3	92,758,417.20	7353.45	7,508,205.95	63.85	12.35
G70	9999	4,336,200.40	9999	4,046,618.05	0	1.07
G72	8189.35	77,034,721.40	8109.9	6,998,747.65	79.45	11.01

Table 8 Comparative results for max-cut with varying combination strategies of O_1 and O_2

Instance	O_1			$O_1 \cup O_2$		
	f_{best}	f_{avg}	$time$ (s)	f_{best}	f_{avg}	$time$ (s)
G22	13,359	13,357.6	381.6	13,359	13,355.8	357.3
G23	13,344	13,343.6	473.4	13,344	13,344	550.9
G25	13,338	13,334	442.8	13,339	13,335.8	690.4
G29	3405	3398.22	211.1	3405	3396.4	254.2
G33	1382	1381.4	553.5	1382	1382	716.5
G35	7686	7681.3	755.4	7684	7679.1	449.6
G36	7680	7672	1367.1	7677	7672.5	408.1
G37	7690	7685.5	1039.2	7689	7683.4	1099.0
G38	7688	7684	135.2	7688	7681.2	177.8
G40	2400	2384.7	453.5	2396	2381.6	427.2
Instance	$rand(O_1, O_2)$			$O_1 + O2$		
	f_{best}	f_{avg}	$time$ (s)	f_{best}	f_{avg}	$time$ (s)
G22	13,359	13,356	365.3	13,359	13,357	438.2
G23	13,344	13,343.9	584.9	13,344	13,344	302.1
G25	13,340	13,336.4	408.8	13,340	13,335.5	451.5
G29	3405	3398.4	403.9	3405	3398.1	569.9
G33	1382	1381.8	585.2	1382	1381.4	667.4
G35	7686	7683.1	628.0	7687	7684.3	968.3
G36	7680	7672	944.8	7680	7675.3	1075.6
G37	7688	7681.7	1078.3	7691	7687.5	1133.2
G38	7688	7680.8	153.6	7688	7685.7	333.0
G40	2395	2388.8	412.4	2400	2385.2	467.1

objective values obtained by different strategies and the best objective values by our original algorithm. We also show in Fig. 2b the box and whisker plots which indicate, for different O_1 and O_2 combination strategies, the distribution and the ranges of the obtained results for the 10 tested instances. The results are expressed as the additive inverse of percent deviation of the averages results from the best known objective values obtained by our original algorithm.

From Fig. 2a, one observes that for the tested instances, other combination strategies obtain fewer best known results compared to the strategy $O_1 + O_2$, and produce large gaps to the best known results on some instances. From Fig. 2b, we observe a clear difference in the distribution of the results with different strategies. For the results with the strategies of $O_1 + O_2$, the plot indicates a smaller mean value and significantly smaller variation compared to the results obtained by other strategies. We thus conclude that the strategy used by our algorithm ($O_1 + O_2$) performs better than other strategies.

4.3 Impact of the diversified improvement search operators

As described in Sect. 2.7, the proposed algorithm employs two diversified operator O_3 and O_4 to enhance the search power of the algorithm and make it possible for the search to visit new promising regions. The diversified improvement procedure uses probability ρ to select

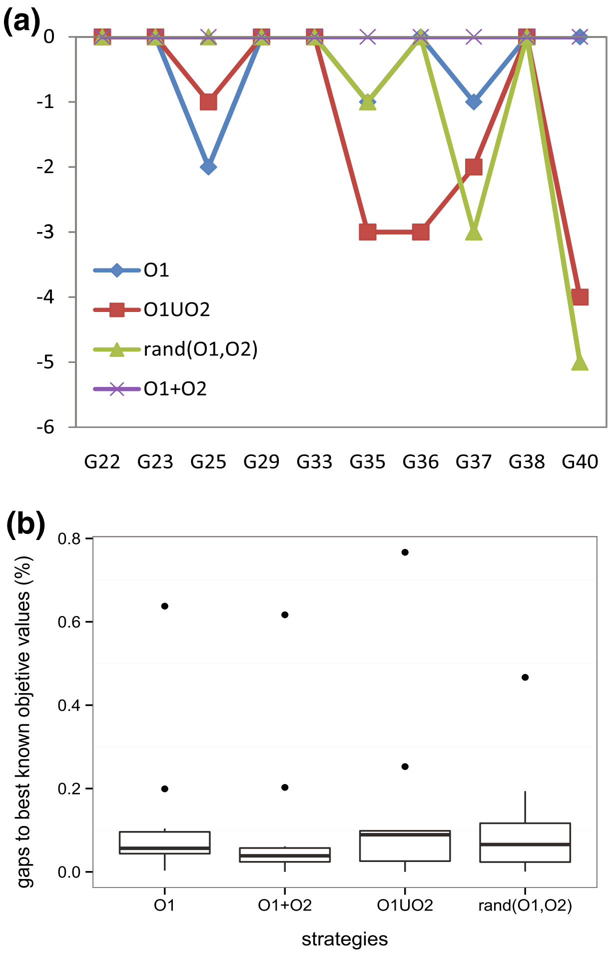

Fig. 2 Analysis of the move operators O_1 and O_2. **a** $f_{best-strategy} - f_{bestknown}$, gaps to best known objective values. **b** $(f_{bestknown} - f_{avg-strategy})/f_{bestknown} \times 100\%$, gaps to best known objective values

O_3 or O_4. To analyze the impact of operators O_3 and O_4, we tested our algorithm with $\rho = 1$ (using the operator O_3 alone), $\rho = 0.5$ (equal application of O_3 and O_4 used in our original MOH algorithm), $\rho = 0$ (using the operator O_4 alone), while keeping all the other ingredients and parameters fixed as described before. The stopping criterion was a timeout limit of 30 minutes. We then independently solved each selected instance 10 times with those different values of ρ. The obtained results on the max-cut problem for the 10 challenging instances used for parameter tuning of Sect. 3.3 are presented in Table 9, including the best objective value f_{best}, the average objective value f_{avg} over the 10 independent runs, as well as the CPU times in seconds to reach f_{best}. To evaluate the performance, we again calculate the gaps between different best objective values shown in Fig. 3a and average objective values shown in Fig. 3b, where the set of values f_{best}, f_{avg}, when $\rho = 0.5$, are set as the reference values.

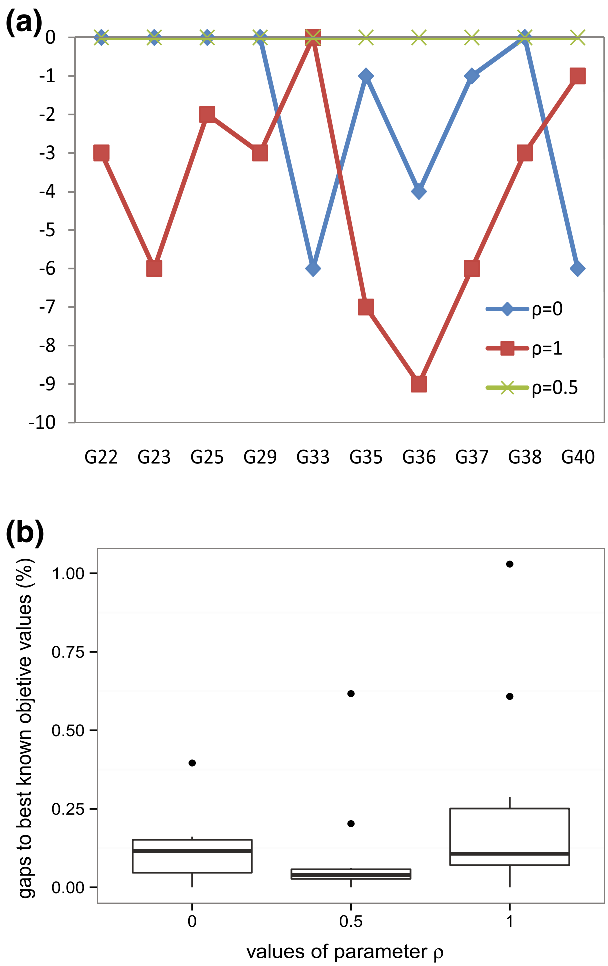

Fig. 3 Analysis of the move operators O_3 and O_4. **a** $f_{best} - \rho - f_{bestknown}$, gaps between f_{best} obtained with different ρ values to best known objective values. **b** $(f_{bestknown} - f_{avg} - \rho)/f_{bestknown} \times 100\%$, gaps to best known objective values

As in Sect. 4.2, to evaluate the performance, we show in Fig. 3a the gaps between the best objective values obtained with different values of ρ and the best objective values by our original MOH algorithm ($\rho = 0.5$). We also show in Fig. 3b the box and whisker plots which indicates, for different values of ρ, the distribution and the ranges of the obtained results for the 10 tested instances. The results are expressed as the additive inverse of percent deviation of the averages results from the best known objective values obtained by our original algorithm.

Figure 3a discloses that using O_3 or O_4 alone obtains fewer best known results than using them jointly and achieves significantly worse results on some particular instances. From Fig. 3b, we observe a visible difference in the distribution of the results with different strategies. For the results with the parameter $\rho = 0.5$, the plot indicates a smaller mean value and significantly smaller variation compared to the results obtained by other strategies. We thus conclude that jointly using O_3 and O_4 with $\rho = 0.5$ is the best choice since it produces better results in terms of both best and average results.

Table 9 Comparative results for max-cut with varying parameter ρ

Instance	$\rho = 1$			$\rho = 0$			$\rho = 0.5$		
	f_{best}	f_{avg}	$time$ (s)	f_{best}	f_{avg}	$time$ (s)	f_{best}	f_{avg}	$time$ (s)
G22	13,359	13,350.1	352.7	13,356	13,355.2	440.6	13,359	13,357	438.2
G23	13,344	13,344	441.4	13,338	13,335.6	340.1	13,344	13,344	302.1
G25	13,339	13,335.1	426.1	13,337	13,333.5	412.9	13,340	13,335.5	451.5
G29	3405	3395.2	614.5	3402	3399.8	593.5	3405	3398.1	569.9
G33	1376	1373.6	519.9	1382	1382	609.2	1382	1381.4	667.7
G35	7686	7680.7	832.1	7680	7678.2	850.8	7687	7684.3	968.3
G36	7676	7669.2	1540.8	7671	7667.6	1304.8	7680	7675.3	1075.6
G37	7690	7681.2	1167.8	7685	7679.6	1053.8	7691	7687.5	1133.2
G38	7688	7681.4	275.1	7685	7679	257.3	7688	7685.7	333.0
G40	2394	2375.3	453.0	2399	2390.5	529.8	2400	2385.2	467.1

5 Conclusion

Our multiple search operator algorithm (MOH) for the general max-k-cut problem achieves a high level performance by including five distinct search operators which are applied in three search phases. The descent-based improvement phase aims to discover local optima of increasing quality with two intensification-oriented operators. The diversified improvement phase combines two other operators to escape local optima and discover promising new search regions. The perturbation phase is applied as a means of strong diversification to get out of deep local optimum traps. To obtain an efficient implementation of the proposed algorithm, we developed streamlining techniques based on bucket sorting.

We demonstrated the effectiveness of the MOH algorithm both in terms of solution quality and computation efficiency by a computational study on the two sets of well-known benchmarks composed of 91 instances. For the general max-k-cut problem, the proposed algorithm is able to improve 90 percent of the current best known results available in the literature. Moreover, for the very popular special case with $k = 2$, i.e., the max-cut problem, MOH also performs extremely well by discovering 4 improved best results which were never reported by any max-cut algorithm of the literature. We also investigated the importance of the bucket sorting technique as well as alternative strategies for combing search operators and justified the combinations adopted in the proposed MOH algorithm.

Given that most ideas of the proposed algorithm are general enough, it is expected that they can be useful to design effective heuristics for other graph partitioning problems.

Acknowledgements We are grateful to the reviewers of the paper which helped us to improve the work. The work was supported by the PGMO (2014-0024H) project from the Jacques Hadamard Mathematical Foundation, National Natural Science Foundation of China (Grant No. 71501157) and China Postdoctoral Science Foundation (Grant No. 2015M580873). Support for Fuda Ma from the China Scholarship Council is also acknowledged.

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.

Barahona, F., Grötschel, M., Jünger, M., & Reinelt, G. (1988). An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, *36*(3), 493–513.

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

Boros, E., & Hammer, P. (1991). The max-cut problem and quadratic 0–1 optimization: Polyhedral aspects, relaxations and bounds. *Annals of Operations Research*, *33*, 151–180.

Burer, S., & Monteiro, R. D. (2001). A projected gradient algorithm for solving the maxcut SDP relaxation. *Optimization Methods and Software*, *15*(3–4), 175–200.

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

Chang, K. C., & Du, D. H. C. (1987). Efficient algorithms for layer assignment problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *6*(1), 67–78.

Chen, R. W., Kajitani, Y., & Chan, S. P. (1983). A graph-theoretic via minimization algorithm for two-layer printed circuit boards. *IEEE Transactions on Circuits and Systems*, *30*(5), 284–299.

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.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to algorithms* (Vol. 6). Cambridge: MIT Press.

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 IEEE international conference on data mining, 2001. ICDM 2001, pp. 107–114. IEEE.

Eisenblätter, A. (2002). The semidefinite relaxation of the k-partition polytope is strong. In: Cook, W. J., & Schulz, A. S. (Eds.), *Integer Programming and Combinatorial Optimization* (pp. 273–290). Berlin: Springer.

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

Fiduccia, C. M., & Mattheyses, R. M. (1982). A linear-time heuristic for improving network partitions. In: *19th Conference on design automation, 1982*, pp. 175–181. IEEE.

Ghaddar, B., Anjos, M. F., & Liers, F. (2011). A branch-and-cut algorithm based on semidefinite programming for the minimum k-partition problem. *Annals of Operations Research*, *188*(1), 155–174.

Glover, F., & Laguna, M. (1999). *Tabu search*. Berlin: Springer.

Kahruman, S., Kolotoglu, E., Butenko, S., & Hicks, I. V. (2007). On greedy construction heuristics for the max-cut problem. *International Journal of Computational Science and Engineering*, *3*(3), 211–218.

Kann, V., Khanna, S., Lagergren, J., & Panconesi, A. (1997). On the hardness of approximating max k-cut and its dual. C*hicago Journal of Theoretical Computer Science, 2*.

Karp, R. M. (1972). *Reducibility among combinatorial problems*. Berlin: Springer.

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.

Liers, F., Jünger, M., Reinelt, G., & Rinaldi, G. (2004). Computing exact ground states of hard ising spin glass problems by branch-and-cut. In: Hartmann, A., & Rieger, H. (Eds.), *New Optimization Algorithms in Physics* (pp. 47–68). London: Wiley.

Lin, G., & Zhu, W. (2012). A discrete dynamic convexized method for the max-cut problem. *Annals of Operations Research*, *196*(1), 371–390.

Lin, G., & Zhu, W. (2014). An efficient memetic algorithm for the max-bisection problem. *IEEE Transactions on Computers*, *63*(6), 1365–1376.

Lü, Z., Glover, F., Hao, & J. K. (2011). Neighborhood combination for unconstrained binary quadratic problems. In *MIC post-conference book*, pp. 49–61.

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 (NRL)*, *50*(7), 683–701.

Pinter, R. Y. (1984). Optimal layer assignment for interconnect. *Journal of VLSI and Computer Systems*, *1*(2), 123–137.

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 Analysis*, *51*(1), 16–24.

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

Wang, Y., Lü, Z., Glover, F., & Hao, J. K. (2013). Probabilistic grasp-tabu search algorithms for the UBQP problem. *Computers & Operations Research*, *40*(12), 3100–3107.

Wu, Q., & Hao, J. K. (2012). A memetic approach for the max-cut problem. In: Coello Coello, C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., & Pavone, M. (Eds.), *Parallel Problem Solving from Nature* - PPSN XII, Lecture Notes in Computer Science (vol. 7492, pp. 297–306). Berlin: Springer.

Wu, Q., & Hao, J. K. (2013). Memetic search for the max-bisection problem. *Computers & Operations Research*, *40*(1), 166–179.

Wu, Q., Wang, Y., & Lü, Z. (2015). A tabu search based hybrid evolutionary algorithm for the max-cut problem. *Applied Soft Computing*, *34*(1), 827–837.

Zhu, W., Lin, G., & Ali, M. M. (2013). Max-k-cut by the discrete dynamic convexized method. *INFORMS Journal on Computing*, *25*(1), 27–40.

Zhu, W., Liu, Y., & Lin, G. (2015). Speeding up a memetic algorithm for the max-bisection problem. *Numerical Algebra*, *5*(2), 151–168.