An effective iterated tabu search for the maximum bisection problem

Fuda Ma[a], Jin-Kao Hao[a,b,*], Yang Wang[c]

[a] *LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*
[b] *Institut Universitaire de France, Paris, France*
[c] *School of Management, Northwestern Polytechnical University, 127 Youyi West Road, 710072 Xi'an, China*

Computers and Operations Research
http://dx.doi.org/10.1016/j.cor.2016.12.012

Abstract

Given an edge weighted graph $G = (V, E)$, the maximum bisection problem involves partitioning the vertices of V into two disjoint subsets of equal cardinality such that the weight sum of the edges crossing the two subsets is maximized. In this study, we present an Iterated Tabu Search (ITS) algorithm to solve the problem. ITS employs two distinct search operators organized into three search phases to effectively explore the search space. Bucket sorting is used to ensure a high computational efficiency of the ITS algorithm. Experiments based on 71 well-known benchmark instances of the literature demonstrate that ITS is highly competitive compared to state-of-the-art approaches and discovers improved best-known results (new lower bounds) for 8 benchmark instances. The key ingredients of the algorithm are also investigated.

Keywords: Max-bisection, graph partition, multiple search strategies, tabu search, heuristics.

1. Introduction

Given an undirected graph $G = (V, E)$ with a set of vertices $V = \{1, \ldots, n\}$, a set of edges $E \subset V \times V$ and a set of edge weights $\{w_{ij} \in Z : \{i, j\} \in E\}$ ($w_{ij} = 0$ if $\{i, j\} \notin E$). The maximum bisection problem (max-bisection for short) aims to partition the vertex set V into two disjoint subsets S_1 and S_2 of equal cardinality (i.e., $S_1 \cup S_2 = V, S_1 \cap S_2 = \emptyset, |S_1| = |S_2|$), such that the weight sum of the edges whose endpoints belong to different subsets is maximized, i.e.,

*Corresponding author
Email addresses: ma@info.univ-angers.fr (Fuda Ma), hao@info.univ-angers.fr (Jin-Kao Hao), yangw@nwpu.edu.cn (Yang Wang)

$$\max \sum_{i \in S_1, j \in S_2} w_{ij}. \tag{1}$$

Max-bisection is a classical NP-hard combinatorial optimization problem and cannot be solved exactly in polynomial time unless $P = NP$ [31]. Moreover, when the equal cardinality constraint is relaxed, the problem is referred to as the maximum cut problem (max-cut) whose decision version is one of Karp's 21 NP-complete problems [22]. Both max-bisection and max-cut have been subject of many studies in the literature.

In particular, max-bisection has attracted increasing attention in recent decades due to its relevance to numerous applications like VLSI layout design [2, 8, 9], data clustering [14] and sports team scheduling [15] among others.

Unfortunately, like max-cut, max-bisection is a computationally challenging problem. To solve the problem, a number of exact and heuristic procedures have been reported in the literature. Examples of exact algorithms based on branch-and-cut and semidefinite programming are described in [5] and [21]. But due to the high computational complexity of the problem, only instances with no more than a few hundred of vertices can be solved by these exact methods in a reasonable computing time.

For large instances, heuristic methods are commonly used to find sub-optimal solutions of good quality within an acceptable time frame. In particular, for the very popular max-cut problem, many heuristic algorithms have been proposed, including simulated annealing and tabu search [1], breakout local search [4], projected gradient approach [6], discrete dynamic convexized method [24], rank-2 relaxation heuristic [7], variable neighborhood search [16], greedy heuristics [20], scatter search [30], global equilibrium search [35, 36], unconstrained binary quadratic optimization [38] and memetic search [39]. On the other hand, unlike the intensively investigated max-cut problem, there are only few heuristics for the max-bisection problem including in particular a Lagrangian net algorithm [41], a deterministic annealing algorithm [12], a variable neighborhood search algorithm [26] and three recent memetic algorithms [25, 40, 42]. These memetic algorithms have produced the best computational results on a set of max-bisection benchmark instances and were used as the main references for our comparative studies.

In this study, we introduce an effective heuristic algorithm for the max-bisection problem based on the iterated local search (ILS) framework [28], which has been applied with success to a number of difficult combinatorial optimization problems (for some recent application examples, see [4, 10, 33, 34, 37]). The proposed iterated tabu search (ITS) algorithm relies on two distinct local search operators for solution transformations. The algorithm is composed of three complementary search phases (descent-based improvement, diversifying improvement and perturbation) to ensure an effective examination of the search space. The basic idea of the proposed approach can be summarized as follows. Using the fast *1-move* operator (Section 2.3), the descent-based improvement procedure aims to locate a local optimum from an initial solution

(Section 2.6). Then the diversifying improvement phase applies a tabu search procedure (with the *1-move* and *constrained swap* operators) to examine nearby search areas around the obtained local optimum to discover improved solutions (Section 2.7). Each time that an improved solution is found, the search switches back to the descent-based improvement phase to make an intensive exploitation of the area. If the search stagnates, the perturbation phase applies a random search operator to definitively move the search process to a distant region from which a new round of the search procedure starts. This process is iterated until a stopping condition is met. To ensure the computational efficiency of the search operators, we employ streamlining techniques based on dedicated data structures and tie-breaking techniques (Sections 2.4 and 2.5)

The proposed ITS algorithm includes the following original features. First, ITS relies on a joint use of two complementary search operators to conduct an extensive exploitation of the search space. The *1-move* operator is used to first discover a local optimal solution from which improved solutions are further sought by employing the *constrained swap* operator. Second, in addition to an improvement phase and a perturbation phase used in conventional ILS algorithms, the proposed ITS algorithm uses a fast descent procedure to quickly attain promising search areas which are intensively explored with the powerful tabu search procedure. This combination prevents the search procedure from running the more expensive tabu search procedure in unpromising areas and thus helps to increase the search efficiency of the whole algorithm.

We assess the performance of the proposed algorithm based on 71 well-known benchmark graphs in the literature which were commonly used to test max-cut and max-bisection algorithms. Computational results show that ITS competes favorably with respect to the existing best performing max-bisection heuristics, by obtaining improved best-known results (new lower bounds) for 8 instances.

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

2. Iterated tabu search for max-bisection

This section presents the proposed ITS algorithm for max-bisection. We first introduce its general working scheme and then present in detail its components (search space, move operators, descent procedure, tabu search procedure and perturbation procedure).

2.1. General working scheme

The general procedure of the proposed ITS algorithm is described in Algorithm 1 whose components are explained in the following subsections. The algorithm explores the search space of bisections (Section 2.2) by alternately applying two distinct and complementary move operators (*1-move* and constrained

Algorithm 1 General ITS procedure for the max-bisection problem.

1: **Require**: Graph $G = (V, E)$, max number ω of consecutive non-improvement iterations in diversified phase, probability ρ for selecting *1-move* and $c - swap()$.
2: **Ensure**: the best solution I_{best} found
3: $I \leftarrow$ Random_Initial_solution() ▷ A random bisection from the search space Ω, see Section 2.2
4: $I_{best} \leftarrow I$ ▷ I_{best} records the best solution found so far
5: $iter \leftarrow 0$ ▷ Iteration counter
6: **while** stopping condition not satisfied **do**
7: /* lines 8 to 20: Descent local search phase, see Section 2.6 */
8: **repeat**
9: $I^* \leftarrow I$
10: $I \leftarrow I \oplus$ *1-move*(u, S_1) ▷ Select a vertex with the best move gain and perform the *1-move*
11: Update move gains ▷ Move gains recorded in a bucket data structure, see Section 2.4
12: $iter \leftarrow iter + 1$
13: $I \leftarrow I \oplus$ *1-move*(v, S_2)
14: Update move gains; $iter \leftarrow iter + 1$
15: **until** $f(I) < f(I^*)$
16: /* lines 17 to 20: Roll back to recover the search status when the local optimum I^* is reached */
17: $I \leftarrow I \oplus$ *1-move*(v, S_1)
18: Update move gains; $iter \leftarrow iter + 1$
19: $I \leftarrow I \oplus$ *1-move*(u, S_2)
20: Update move gains; $iter \leftarrow iter + 1$
21: **if** $f(I^*) > f(I_{best})$ **then**
22: $I_{best} \leftarrow I^*$ ▷ Update the best solution found so far
23: **end if**
24: /* lines 25 to 44: Diversifying improvement phase, see Section 2.7 */
25: $c \leftarrow 0$ ▷ Counter of non-improvement iterations
26: **while** $c < \omega$ **do**
27: **if** $Random(0, 1) < \rho$ **then** ▷ $Random(0, 1)$ returns a random real number between 0 to 1
28: $I \leftarrow I \oplus$ *c-swap*(u, v) ▷ Perform the best *c-swap* considering tabu status
29: Add $\{u, v\}$ to tabu list
30: Update move gains; $iter \leftarrow iter + 1$
31: **else**
32: $I \leftarrow I \oplus$ *1-move*(u, S_1) ▷ Perform the best *1-move* considering tabu status
33: Add u to tabu list
34: Update move gains; $iter \leftarrow iter + 1$
35: $I \leftarrow I \oplus$ *1-move*(v, S_2)
36: Add v to tabu list
37: Update move gains; $iter \leftarrow iter + 1$
38: **end if**
39: **if** $f(I) > f(I_{best})$ **then**
40: $I_{best} \leftarrow I$; $c \leftarrow 0$ ▷ Update the best solution found so far
41: **else**
42: $c \leftarrow c + 1$
43: **end if**
44: **end while**
45: /* Perturbation phase, see Section 2.8 */
46: $I \leftarrow Perturb(I)$
47: **end while**

swap) to make transitions from the current solution to a neighboring solution (Section 2.3). Basically, from an initial solution (i.e., a bisection) which is randomly sampled from the search space, the algorithm first applies, with operator *1-move*, a descent local search (DLS) to attain a local optimum I (Algorithm 1, lines 8-20, descent local search phase, Section 2.6). We note that, since the last solution I from the descent local search phase is obtained from the reached local optimum I^* after two additional *1-move* operations, we need to apply two reverse *1-move* operations (rollback) to I to recover I^* (Algorithm 1, lines 17-20). An alternative method to achieve the same purpose would be to copy I^* to I which however requires to re-initialize the bucket data structure (Section 2.4) and thus is more expensive than the adopted rollback method.

From the attained local optimum I, the algorithm continues the search with the diversifying improvement phase (Algorithm 1, lines 25-44, Section 2.7) that uses a tabu search procedure to explore new solutions around I. This search phase relies on both the *1-move* and constrained swap (denoted by *c-swap*) operators, which are applied in a probabilistic way. This tabu search phase ends when the best solution found I_{best} cannot be further improved during ω consecutive iterations. In this case, it is considered that an exhaustive exploration of the search area has been performed. To continue the search, the algorithm applies a perturbation phase (Algorithm 1, line 46), which deeply transforms the current solution by randomly swapping γ vertices (Section 2.8). The perturbed solution serves then as a new starting solution for the next round of the descent local search phase. The overall process is iterated until a stopping criterion (e.g., a given cutoff time) is met and the best solution found during the search is returned as the outcome of the algorithm.

2.2. Search space and evaluation solution

Given the purpose of max-bisection (i.e., to partition the vertex set V into two equal-sized subsets such that the weight sum of the edges crossing the two subsets is maximized), we define the search space Ω to be composed of all possible *bisections* (i.e., balanced two-way partitions) $\{S_1, S_2\}$ of vertex set V as follows.

$$\Omega = \{\{S_1, S_2\} : S_1, S_2 \subset V, S_1 \cup S_2 = V, S_1 \cap S_2 = \emptyset, |S_1| = |S_2|\}. \quad (2)$$

The objective value $f(I)$ of a given bisection $I = \{S_1, S_2\}$ of Ω is the weight sum of the edges crossing S_1 and S_2:

$$f(I) = \sum_{i \in S_1, j \in S_2} w_{ij}. \quad (3)$$

For two candidate bisections $I' \in \Omega$ 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. Our algorithm only samples feasible solutions within the above search space.

2.3. Move operators and neighborhood

From the current solution which is necessarily a feasible solution (i.e., a bisection), the ITS algorithm explores its neighboring solutions by applying the *1-move* and *c-swap* operators. Formally, we let $I = \{S_1, S_2\}$ be the current solution and mv be a move operator, we use $I' \leftarrow I \oplus mv$ to denote the neighboring solution I' obtained by applying mv to I.

For a given move operator mv, we define the *move gain* Δ_{mv} as the variation in the objective value when mv is applied to transform the current solution I into a neighboring solution I', i.e.,

$$\Delta_{mv} = f(I') - f(I) \tag{4}$$

where f is the optimization objective defined in Eq. (3).

The *1-move* and *c-swap* operators are defined as follows.

- *1-move*: Given a bisection $I = \{S_1, S_2\}$, *1-move*(v, S_i) displaces a vertex v from its current subset S_i $(i = 1, 2)$ to the other subset. We note that one application of *1-move* always leads to an unbalanced partition (thus an infeasible bisection). To maintain the balance of the partition, two consecutive applications of *1-move* are always jointly performed by moving first a vertex u from subset S_1 to S_2 (denoted by *1-move*(u, S_1)), and then by moving another vertex v from S_2 to S_1 (denoted by *1-move*(v, S_2)). Such a combined application of *1-move* ensures a balanced partition (thus a feasible bisection).

- *c-swap*: Given a bisection $I = \{S_1, S_2\}$, *c-swap*(v_1, v_2) exchanges two vertices $v_1 \in S_1$ and $v_2 \in S_2$ belonging to two subsets subject to the constraint that v_1 and v_2 are linked by an edge $\{v_1, v_2\} \in E$. Thus, our *c-swap* operator only considers pairs of vertices such that they not only belong to the two subsets of the bisection, but also are linked by an edge crossing the subsets.

Based on these two move operators (*1-move* and *c-swap*), two neighborhoods $N1$ and $N2$ are defined as follows:

$$N1 = \{I \oplus \textit{1-move}(v, S_i) : v \in S_i\}$$

$$N2 = \{I \oplus \textit{c-swap}(v_1, v_2) : v_1 \in S_1, v_2 \in S_2, \{v_1, v_2\} \in E\}$$

where $I = \{S_1, S_2\}$ is a feasible solution. Clearly, $N1$ and $N2$ are bounded in size by $O(|V|)$ and $O(|E|)$ respectively.

As stated above, since the neighboring solutions of I in $N1$ are infeasible, two consecutive applications of *1-move* are performed to maintain the feasibility of the new neighboring solution. We also note that the *1-move* operator was commonly used in previous studies [17, 25, 40].

On the contrary, few studies investigate the *swap* operator. When it was employed, it was usually applied in an *unconstrained* way in the sense that each possible pair of vertices (v_1, v_2) such that $v_1 \in S_1$ and $v_2 \in S_2$ was considered

[23]. Note that the unconstrained swap operator leads to a neighborhood of size $O(|V|^2)$ which is typically much larger than our $N2$ neighborhood induced by the constrained *c-swap* operator (bounded by $O(|E|)$ in size). This is particularly true for sparse graphs. We will study the merit of *c-swap* compared to the unconstrained swap operator in Section 4.2.

After an application of either one of the two move operators, the gain obtained with regard to the objective is updated according to the dedicated streamlining techniques explained below.

2.4. Bucket sorting for fast move gain evaluation and updating

As we show in Sections 2.6 and 2.7, our algorithm iteratively makes transitions from the current solution to a particular neighboring solution by applying a selected move operation. Typically, to make the right choice, the algorithm needs to identify the most favorable move operation with an increased move gain among many candidates. To ensure a high search efficiency, it is crucial for the algorithm to be able to rapidly evaluate all the candidate moves at each iteration of its search process. In this section, we describe fast incremental evaluation techniques based on bucket sorting [11] to streamline the calculations. These specific techniques allow the algorithm to efficiently keep and update the move gains after each move application.

1-move: For each *1-move(v,S)* application, let Δ_v be the move gain of moving vertex $v \in S$ to the other subset $V \setminus S$ (We use the notation $\Delta_{v \to S}$ if the destination subset S needs to be emphasized). Then initially, each move gain can be determined by the following Formula:

$$\Delta_v = \sum_{i \in S, i \neq v} \omega_{vi} - \sum_{j \in V \setminus S} \omega_{vj} \tag{5}$$

where ω_{vi} and ω_{vj} are respectively the weights of edges $\{v, i\}$ and $\{v, j\}$.

Then, once a *1-move(v,S)* is performed, the move gain of each vertex can be updated as follows:

1. $\Delta_v = -\Delta_v$
2. for each $u \in V \setminus \{v\}$,

$$\Delta_u = \begin{cases} \Delta_u - 2\omega_{uv}, & \text{if } u \in S \\ \Delta_u + 2\omega_{uv}, & \text{if } u \in V \setminus S \end{cases} \tag{6}$$

Now we explain where the factor 2 in Eq. (6) comes from. Let us first consider the gain of moving a vertex $u \in S$ ($u \neq v$), which is $\Delta_u = \sum_{i \in S, i \neq u} w_{ui} - \sum_{j \in V \setminus S} w_{uj}$ according to the definition of the objective function. After the vertex v is moved from S to $V \setminus S$, the gain of moving vertex u is updated as $\Delta_u = \sum_{i \in S \setminus \{v\}, i \neq u} w_{ui} - \sum_{j \in V \setminus S \cup \{v\}} w_{uj} = (\sum_{i \in S, i \neq u} w_{ui} - w_{uv}) - (\sum_{j \in V \setminus S} w_{uj} + w_{uv}) = \Delta_u - 2w_{uv}$. Similarly, the gain of moving a vertex $u \in V \setminus S$ ($u \neq v$) is given by

$\Delta_u = \sum_{j \in V \backslash S, j \neq u} w_{uj} - \sum_{i \in S} w_{ui}$. After the vertex v is moved from S to $V \backslash S$, the gain of moving $u \in V \backslash S$ is updated as $\Delta_u = \sum_{j \in V \backslash S \cup \{v\}, j \neq u} w_{uj} - \sum_{i \in S \backslash \{v\}} w_{ui} = (\sum_{j \in V \backslash S, j \neq u} w_{uj} + w_{uv}) - (\sum_{i \in S} w_{ui} - w_{uv}) = \Delta_u + 2w_{uv}$.

We note that if there is no edge between the vertices u and v, the edge weight ω_{uv} equals 0, in which case the associated Δ_u value will not change. We observe that only the move gains of vertices affected by this move (i.e., the displaced vertex and its adjacent vertices) will be updated, which reduces the computation time significantly.

Usually the move gains can be stored in an array, so that the time for finding the best move (maximum gain) grows linearly with the number of vertices ($O(n)$). For large problem instances (very large n), the required time can still be quite high. To avoid unnecessary search for the best move, we adopt a bucket structure which is inspired by the bucket sorting proposed in [17] for the circuit partitioning problem (More details about bucket sorting can be found in [11]). With this technique, we always keep the vertices ordered by their move gains in decreasing order, so that the most favorable vertex can be identified quickly as we explain below.

Our bucket sorting for *1-move* relies on two arrays of buckets, one for each partition subset $S_i \in \{S_1, S_2\}$. In each bucket array i, $i \in \{1, 2\}$, the j^{th} entry stores the vertices with the move gain $\Delta_{v \rightarrow S_i}$ currently being equal to j, where the vertices are maintained by a circular double linked list (CDLL). To ensure a direct access to the vertices in the CDLLs, as suggested in [17], the algorithm also maintains another array for all vertices, where each element points to its corresponding vertex in the CDLL. The use of a CDLL instead of a double linked list (DLL) like in [17] aims to ease the implementation of our tie-breaking scheme which is needed to select the vertex when several candidates exist (see Section 2.5 for more details on this issue). An illustrative example of the bucket structure for max-bisection is given in Appendix A.

After each *1-move*, the bucket structure is updated by recomputing the move gains (see Formula (6)) of the affected vertices (the moved vertex and its adjacent vertices) and by shifting them to appropriate buckets.

We note that the bucket data structure was originally proposed and used in a local search method developed for the circuit partitioning partitioning problem [17]. Very recently this technique has been used to speed up a swap-based Lin-Kernighan local search method for max-bisection within an improved memetic algorithm [42].

2.5. Selection of the best vertex with a tie breaking scheme

For each bucket array, finding the best vertex with maximum move gain is equivalent to finding the first non-empty bucket from the top of the array and then selecting a vertex in its circular double linked list. If there are more than one vertex with maximum move gain in the CDLL (see Figure A.3 in Appendix A), a tie occurs. We observed experimentally that many ties may occur during

the runs of our ITS algorithm, which reveals the importance of a suitable tie-breaking scheme. Three tie-breaking schemes, namely random selection, FIFO (first-in-first-out) selection and LIFO (last-in-first-out) selection are often used. The work of [19] showed that the LIFO selection of gain buckets was superior to the FIFO selection and random selection. A possible explanation given by the authors was that clustered vertices tend to move together.

In our algorithm, we use the LIFO selection scheme to break ties. However, given that our algorithm employs a tabu mechanism to forbid a vertex to move back to its original subset (see Section 2.7), it is inappropriate to insert the forbidden vertices at the head of the list, since doing this will cause useless computations when searching for a proper vertex for a move operation. To adapt the LIFO selection scheme to tabu search, we make the following improvements.

To update the move gain of an impacted vertex after a move, ITS first checks the tabu status of the vertex. If the vertex is in the tabu list, ITS inserts the vertex at the *tail* of the corresponding gain bucket, otherwise, ITS inserts the vertex at the *head* of the gain bucket. To choose the vertex for a *1-move* operation, ITS always selects the first vertex which is not in the tabu list from the head of the gain bucket. This strategy reduces the computing time for checking those forbidden vertices, as we show in Section 4.1.

***c-swap*:** For each *c-swap*(u, v) operation, let $\Delta_{u,v}$ be the move gain of exchanging vertices u and v between the two subsets of the bisection. Then $\Delta_{u,v}$ can be calculated by a combination of the move gains of its two underlying *1-move* operations (Δ_u and Δ_v) as follows:

$$\Delta_{u,v} = \Delta_u + \Delta_v + 2\omega_{uv} \tag{7}$$

According to the definition of the neighborhood $N2$, only the endpoints (vertices) of the edges crossing the two subsets S_1 and S_2 are considered. Thus for a given solution, there are at most $|E|$ candidate *c-swap* moves to evaluate. Still, directly searching for the best move among all candidate moves may be too computationally expensive. In order to mitigate this problem, we maintain another bucket structure for *c-swap* moves to accelerate the move evaluation process. The bucket structure for *c-swap* is similar to that for *1-move*. This is achieved by keeping an array of buckets and in each bucket, the i^{th} entry stores the edge $\{u, v\}$ with the move gain $\Delta_{u,v}$ currently being equal to i, where the edges are maintained by a circular double linked list. To ensure a direct access to the edges in the circular double linked lists, as described above, the algorithm also maintains another array for all edges, where each entry points to its corresponding edge in the circular double linked lists.

Similarly, after each move, the bucket structure is updated by recomputing the move gains (see Formula (6)) of the affected vertices (i.e., each swapped vertex and its adjacent vertices), by shifting them to appropriate buckets.

Complexity: Each move involves searching for the vertex or a pair of vertices with maximum move gain, recomputing the move gain for the affected vertices and updating the bucket structure. The vertex with maximum move gain can be found in constant time ($O(1)$). Recomputing move gains requires

linear time relative to the number of affected vertices ($O(n)$). The time of updating the bucket structure is also only related to the number of affected vertices bounded by ($O(d_{max})$) where d_{max} is the maximum degree of the graph.

2.6. Descent local search phase to locate local optima

The descent local search phase is used to obtain a local optimum from a given starting solution [32] (see Algorithm 1, lines 8 - 20). DLS employs the *1-move* operator defined in Section 2.3 to iteratively improve the current solution until a local optimum is reached. At each of its iterations, a best *1-move* (i.e., with maximum gain) is selected by using the bucket structure described in Section 2.4 to move the associated vertex from its current subset to the other subset. If two or more vertices have the same largest move gain, the LIFO tie-breaking strategy described in Section 2.5 is used to choose the vertex. As explained in Section 2.3, DLS always performs two consecutive *1-move* operations to maintain the balance of the two subsets of the bisection.

After each combined application of two consecutive *1-move* operations, if the new objective value is better (larger) than the objective value of the former solution, DLS continues its descent process with the newly attained solution as its new current solution. Otherwise, DLS stops after rolling back to the solution obtained before the application of the last two consecutive *1-move* operations (see Algorithm 1, lines 17 - 20). This solution corresponds to a local optimum with respect to the $N1$ neighborhood and serves as the input solution to the diversifying improvement search presented in the next section.

2.7. Diversifying improvement phase to discover promising region

The descent local phase alone (see Section 2.6) cannot search beyond the first local optimum it encounters. The diversifying improvement search phase, which is based on the tabu search method [18], aims to intensify the search around this local optimum with the purpose of discovering solutions which are better than the input local optimum.

The diversifying improvement search procedure jointly uses the *1-move* and *c-swap* operators defined in Section 2.3. To apply these two operators, we employ a probabilistic combination technique which extends the existing combination schemes described in [29]. The application of *1-move* or *c-swap* is determined probabilistically at each iteration: with probability ρ (a parameter), *c-swap* is applied; with probability $1-\rho$, *1-move* is applied (see Algorithm 1, lines 25-44).

When *1-move* is selected, the algorithm performs the combined *1-move* operations in a way similar to that described in Section 2.6 except that here a tabu list H is considered [18]. The tabu list is a memory which keeps track of displaced vertices to prevent them from being moved back to their initial subsets. Precisely, the algorithm first selects an *eligible* vertex (see below) with maximum move gain and transfers it from its current subset (say S_1) to the other subset, then it updates the bucket structure of move gains according to the technique described in Section 2.4. Then, it selects another eligible vertex

in the other subset (say S_2) with maximum move gain and moves it from S_2 to S_1. The bucket structure is updated accordingly to account for the modified move gains.

After the transfer of a vertex v, the vertex is added to the tabu list H and forbidden from joining again its original subset during the next H_v iterations. Parameter H_v (called the tabu tenure) is determined dynamically as follows:

$$H_v = 3 + rand(|V|/10) \tag{8}$$

where $rand(k)$ is a random number from 0 to k.

A move leading to a solution better than all solutions is always performed even if the underlying vertex is forbidden by the tabu list (This is called the aspiration criterion in the terminology of tabu search). A vertex is said to be *eligible* if it is not forbidden by the tabu list or if the aspiration criterion is satisfied.

Similarly, when *c-swap* is selected, two vertices $v_1 \in S_1$ and $v_2 \in S_2$ with maximum move gain are selected subject to $\{v_1, v_2\} \in E$. Another tabu list H^c is maintained for *c-swap*. After each *c-swap* move, the edge $\{v_1, v_2\}$ is added to the tabu list H^c and it is forbidden to swap v_1 and v_2 back to their original subsets during the next H^c iterations, which, like for the *1-move*, is dynamically determined by formula (8). The same aspiration criterion than the one used by *1-move* is also applied. After each *c-swap* move, the bucket structure is updated to account for the modified move gains. When multiple best *c-swap* moves are available, the LIFO selection strategy is used to choose the applied *c-swap* move (see Section 2.4).

The tabu search procedure iteratively applies *1-move* and *c-swap* to improve the current solution. If the best solution found so far (f_{best}) cannot be improved during a maximum number ω of consecutive iterations, the search is judged to be trapped in a deep local optimum. In this case, the perturbation phase (Section 2.8) is invoked to move the search to a distant region.

2.8. Perturbation phase for strong diversification

The diversifying improvement phase allows the search to escape from some local optima. However, the algorithm may still get stuck in a non-promising search area. This is the case when the best-found solution f_{best} cannot be improved during ω consecutive iterations. To help the search to move to new search regions, we apply a simple perturbation mechanism to deeply transform the current solution. The perturbation swaps a number of pairs of vertices in the following way. For each swap, we randomly choose one vertex v from S_1 and another vertex u from S_2, and then swap v and u. This process is repeated γ times where γ is a parameter which indicates the strength of the perturbation. After the perturbation phase, the search returns to the descent local search phase with the perturbed solution as its new starting solution.

3. Experimental results and comparisons

3.1. Benchmark instances

To assess the performance of the proposed ITS approach, we carried out intensive computational experiments on 71 well-known benchmark graphs used in previous studies. These graphs have 800 to 20000 vertices and an edge density from 0.02% to 6%. They were 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` or from the authors of this paper. These well-known benchmark graphs were frequently used to evaluate max-bisection and max-cut algorithms [4, 16, 25, 36, 35, 38, 39, 40, 41, 42].

3.2. Experimental protocol

Our ITS algorithm was programmed in C++ and compiled with GNU g++ (optimization flag "-O2"). Our computer is equipped with a Xeon E5440 (2.83GHz, 2GB RAM). When running the DIMACS machine benchmarking program[1] on graphs r300.5, r400.5, and r500.5, our machine required 0.43, 2.62 and 9.85 seconds of CPU time respectively.

3.3. Parameters

The proposed algorithm requires three parameters: maximum allowed number ω of non-improvement iterations, probability ρ for move operator selection, and number γ of perturbation moves. To achieve a reasonable calibration of the parameters, we adopted the irace package [27] which implements the Iterated F-race (IFR) method [3] and allows an automatic parameter configuration. We used the following parameter value ranges for this tuning: $\omega = \{1500, 2500, 3500, 4500, 5500\}, \rho = [0.1, 0.5], \gamma = \{50, 200, 400, 600\}$. We performed the parameter tuning experiment on a selection of 5 representative instances from the 71 benchmark graphs: G22, G23, G37, G55, G62. This calibration experiment led to the parameter values $\omega = 3500$, $\rho = 0.3$ and $\gamma = 200$, which were used in all our experiments. Considering the stochastic nature of our ITS algorithm, each of the 71 benchmark instance was independently solved 20 times with different random seeds. To ensure fair comparisons with other state-of-the-art methods in Sections 3.4 and 3.5, we followed the main reference algorithm (MA-WH) [40] and used a timeout limit as the stopping criterion of our ITS algorithm. The timeout limit was set to 30 minutes for graphs with $|V| < 5000$ and 120 minutes for graphs with $|V| \geq 5000$.

To fully evaluate the performance of the proposed algorithm, we performed a comparison with the four recent and best performing state-of-the-art max-bisection algorithms [25, 40, 41, 42], while the best results in the literature have been reported in [25, 40, 42] from 2013 to 2015.

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

3.4. Comparison with the current best-known solutions

Table 1 shows the computational results of our ITS algorithm for the 71 benchmark graphs[2] in comparison with the previous best-known results f_{pre} reported in [25, 40, 42]. The first two columns of the table indicate the name and number of vertices of the graphs. Columns 4 to 7 present the computational statistics of our algorithm, where f_{best} and f_{avg} show the best and average objective values over 20 runs, *std* gives the standard deviation and $time(s)$ indicates the average CPU time in seconds to reach f_{best}.

From Table 1, we observe that ITS, evaluated under the same cutoff time limit as the best performing reference algorithm MA-WH, is able to improve the previous best-known results for 8 large benchmark graphs (indicated in bold) and match the best-known results for 61 graphs. ITS obtains solution of worse quality for only 2 instances (G57 and G70, indicated in italic). This performance is remarkable given that the current best results were reported recently. Moreover, the results of the proposed algorithm show small standard deviations across different runs and different graphs, indicating a good robustness of the algorithm.

Table 1: Computational results of the proposed ITS algorithm on the set of 71 benchmark graphs in comparison with the current best-known results f_{pre} reported in three recent studies [25, 40, 42].

Instance	$\|V\|$	f_{pre}	f_{best}	f_{avg}	std	$time(s)$
G1	800	11624	11624	11624	0.00	1.50
G2	800	11617	11617	11617	0.00	3.24
G3	800	11621	11621	11621	0.00	1.02
G4	800	11646	11646	11646	0.00	1.77
G5	800	11631	11631	11631	0.00	0.76
G6	800	2177	2177	2177	0.00	1.50
G7	800	2002	2002	2002	0.00	0.53
G8	800	2004	2004	2004	0.00	3.50
G9	800	2052	2052	2052	0.00	1.88
G10	800	1998	1998	1998	0.00	4.99
G11	800	564	564	564	0.00	0.12
G12	800	556	556	556	0.00	0.56
G13	800	582	582	582	0.00	4.52
G14	800	3062	3062	3062	0.00	90.68
G15	800	3050	3050	3050	0.00	55.84
G16	800	3052	3052	3052	0.00	32.82
G17	800	3047	3047	3047	0.00	200.67
G18	800	992	992	992	0.00	14.50
G19	800	905	905	905	0.00	3.51
G20	800	941	941	941	0.00	1.52
G21	800	930	930	930	0.00	50.41
G22	2000	13359	13359	13355.5	5.47	432.10
G23	2000	13344	13344	13342.1	2.09	168.24
G24	2000	13336	13336	13335.0	1.67	300.75
G25	2000	13340	13340	13338.2	1.98	149.21
G26	2000	13328	13328	13327.4	1.54	433.68
G27	2000	3341	3341	3340.65	1.75	140.64
G28	2000	3298	3298	3298	0.00	198.23
G29	2000	3403	3403	3403	0.00	3.26
G30	2000	3412	3412	3412	0.00	54.22
G31	2000	3309	3309	3309	0.00	242.19
G32	2000	1410	1410	1410	0.00	425.70
G33	2000	1382	1382	1382	0.00	485.83

[2]Our best results are available at: `http://www.info.univ-angers.fr/pub/hao/maxbisection/ITSresults.zip`.

Table 1 – continued from previous page

Instance	$\|V\|$	f_{pre}	f_{best}	f_{avg}	std	$time(s)$
G34	2000	1384	1384	1384	0.00	189.27
G35	2000	7686	7686	7684.1	2.04	448.35
G36	2000	7678	7678	7676.45	2.16	634.11
G37	2000	7689	7689	7687.7	2.09	627.86
G38	2000	7688	7688	7686.5	3.04	688.32
G39	2000	2408	2408	2406.8	2.56	242.60
G40	2000	2400	2400	2398.8	3.02	354.50
G41	2000	2405	2405	2404.2	0.99	82.55
G42	2000	2481	2481	2476.8	5.85	286.18
G43	1000	6659	6659	6659	0.00	5.25
G44	1000	6650	6650	6650	0.00	2.09
G45	1000	6654	6654	6654	0.00	3.99
G46	1000	6649	6649	6649	0.00	30.12
G47	1000	6657	6657	6657	0.00	4.88
G48	3000	6000	6000	6000	0.00	0.97
G49	3000	6000	6000	6000	0.00	1.57
G50	3000	5880	5880	5880	0.00	50.64
G51	1000	3847	3847	3847	0.00	101.43
G52	1000	3851	3851	3851	0.00	98.43
G53	1000	3850	3850	3850	0.00	109.50
G54	1000	3851	3851	3851	0.00	177.89
G55	5000	10299	10299	10290.8	4.54	2596.84
G56	5000	4016	4016	4013.1	2.28	1926.45
G57	5000	3492	*3490*	3487.8	1.88	610.16
G58	5000	19276	19276	19265.9	3.18	5102.34
G59	5000	6085	6085	6074.3	2.35	4902.13
G60	7000	14186	**14187**	14176.5	4.01	5678.63
G61	7000	5796	5796	5780.2	5.08	4072.54
G62	7000	4866	4866	4860.1	2.69	1472.10
G63	7000	26977	**26988**	26985.3	1.18	2256.66
G64	7000	8731	**8737**	8712.1	6.28	6032.55
G65	8000	5556	5556	5550.9	2.42	2350.98
G66	9000	6352	**6356**	6352.0	1.93	1323.15
G67	10000	6936	**6938**	6935.5	1.34	1023.40
G70	10000	9582	*9581*	9576.3	0.98	1154.32
G72	10000	6990	**6994**	6992.5	0.84	1201.97
G77	14000	9910	**9918**	9915.1	1.02	2013.44
G81	20000	14008	**14030**	14025.45	1.36	1953.23

3.5. Comparison with state-of-the-art max-bisection algorithms

In this section, we further evaluate the performance of the proposed algorithm by comparing it with the four best performing algorithms that achieved state-of-art performances.

1. A Lagrangian net algorithm (LNA) (2011) [41] integrating the discrete Hopfield neural network and the penalty function method (relaxing the bisection constraint in the objective function). The reported results of LNA were obtained on a PC with a 2.36GHz CPU and 1.96GB RAM. The algorithm was programmed in Matlab 7.4. The exact stopping conditions were not explicitly provided.
2. A memetic algorithm (MA-WH) (2013) [40] combining a grouping crossover operator with a tabu search procedure. The results reported in the paper were obtained on a PC with a 2.83GHz Intel Xeon E5440 CPU and 2.0GB RAM (the same platform was used in our study). The program was coded in C. MA-WH used, as its stopping condition, a cutoff limit of 30 minutes for graphs with $|V| < 5000$ and 120 minutes for graphs with $|V| \geq 5000$. We also used the same stopping criterion in our ITS algorithm.
3. Another memetic algorithm (MA-LZ) (2014) [25] integrating a grouping crossover operator and an improved local search procedure based on the

FM heuristic proposed in [17]. The reported results of MA-LZ were obtained on a PC with a 2.11GHz AMD CPU and 1.0GB RAM. The algorithm was programmed in C++. MA-LZ used the following stopping conditions: when the current best solution was not improved after 500 consecutive generations or when MA-LZ reached the maximum number of 5000 generations.

4. A recent memetic algorithm (MA-ZLL) (2015) [42] which is an improvement of the MA-LZ algorithm [25]. In particular, MA-ZLL is based on a fast modified Lin-Kernighan local search using bucket-sort and a new population updating function. The reported results of MA-ZLL were obtained on a PC with an Intel(R) Core (TM)2 Duo CPU E7500 (2.93GHz) and 2.0G RAM under Windows XP. The algorithm was programmed in C. MA-ZLL used the following stopping conditions: when the current best solution was not improved after 3000 consecutive generations or when the algorithm reached the maximum number of 10,000 generations.

Both the MA-WH algorithm and our ITS algorithm used the same computing platform while LNA, MA-LZ and MA-ZLL were run on different computing platforms. In order to compare the computing time, we measured the differences among the four computing platforms according to the Standard Performance Evaluation Cooperation (SPEC) (`www.spec.org`), which indicated that the computers used by LNA, MA-LZ and MA-ZLL are respectively 1.2, 1.4 and 0.966 times slower than the computer used in our experiments.

Table 2 shows the comparative results of ITS on the whole set of 71 benchmark graphs with respect to the four reference algorithms LNA, MA-WH, MA-LZ and MA-ZLL. For each reference algorithm, we report the best objective values (f_{best}), the CPU times ($time$) in seconds to attain the best objective values (f_{best}), and the differences (gap) between each reference algorithm and our ITS algorithm (a negative value thus indicates a worse result). As mentioned above, to harmonize the computing times, we divided the times of LNA, MA-LZ and MA-ZLL by the factor provided by SPEC, i.e., 1.2, 1.4 and 0.966 respectively. The last two columns reporting the results of our ITS algorithm are extracted from Table 1. The entries marked as "-" in the table indicate that the results are not available for the algorithm.

From Table 2, we first observe that the ITS algorithm performs the best in terms of the best objective values. Specifically, ITS dominates LNA for all the tested instances. Compared to MA-LZ, ITS performs better for 61 instances and obtains equal results for the remaining 10 instances. Compared to the most recent algorithm MA-ZLL, ITS obtains better, equal and worse solutions for 31, 38 and 2 instances respectively. Compared to the main reference algorithm MA-WH, ITS reaches a larger f_{best} objective value for 10 instances and an equal objective value for the other 61 instances. In terms of the computational time, it is not obvious to make a fair comparison given that the competing algorithms obtains solutions of quite different quality. This is particularly true for LNA and MA-LZ which are the worst and second worst in terms of solution quality. With regard to the main reference MA-WH (which was run on the same

computing platform than ITS), ITS produces solutions of equal or better quality for similar computation times on large instances. Moreover, Table 3 reports the computation times required by ITS to produce solutions of the same quality than MA-WH on the 12 largest instances with 7000 to 20000 vertices. The table shows that ITS is much faster for these large instances (except G61). For 6 instances, ITS is even 10 to 20 times faster. Finally, we observe that the MA-ZLL algorithm scales better than ITS to obtain solutions of equal quality.

ITS has a similar computing performance to obtain solutions of equal or better quality for large instances. Moreover, in Table 3 we show time information of ITS to obtain solutions of the same quality as MA-WH for the 12 largest instances with 7000 to 20000 vertices. The table shows that ITS is much faster for these large instances (except G61). For 6 instances, ITS is even 10 to 20 times faster. Finally, we observe the latest MA-ZLL algorithm scales better than ITS to obtain solutions of equal quality.

3.6. Comparison with a state-of-the-art algorithm for minimum bisection

Given an unweighted graph $G = (V, E)$, where $|V|$ is even, the *minimum bisection* problem (or graph bisection) involves finding a partition of the vertices of V into two disjoint subsets S_1 and S_2 of equal cardinality while minimizing the number of cutting edges $\{u, v\} \in E$ such that $u \in S_1$ and $v \in S_2$. The minimum bisection problem can be solved by the ITS algorithm proposed in this paper. In fact, for the given unweighted graph $G = (V, E)$, we can create a weighted graph $G' = (V, E)$ where each edge has a weight value of -1. Then the objective value of the maximum bisection problem of G' multiplied by -1 corresponds to the objective value of the minimum bisection problem of G. Consequently, to solve the minimum bisection problem, we can run our ITS algorithm on the weighted graph G' and return the resulting objective value multiplied by -1.

To test the performance of our ITS algorithm on minimum bisection, we carried out a comparative study with a very recent and powerful exact algorithm specifically designed for this problem [13]. This study was based on 3 sets of benchmarks with a total of 20 graphs used in the reference paper, including CG-Mesh graphs (meshes representing various objects), SteinLib graphs (sparse benchmark instances for the Steiner problem in graphs) and Walshaw graphs (mostly finite-element meshes). We did not test all the graphs used in [13] given that the current implementation of the ITS algorithm does not allow us to solve very large graphs with more than 70,000 vertices.

Table 2: Comparative results of ITS with four state-of-the-art algorithms: LNA [41], MA-LZ [25], MA-WH [40] and MA-ZLL [42].

Instance	$\|V\|$	LNA [41]			MA-LZ [25]			MA-WH[40]			MA-ZLL [42]			ITS	
		f_{best}	$time(s)$	gap	f_{best}	$time(s)$	gap	f_{best}	$time(s)$	gap	f_{best}	$time(s)$	gap	f_{best}	$time(s)$
G1	800	11490	22.22	-134	11624	13.38	0	11624	2.4	0	11624	2.79	0	11624	1.5
G2	800	11505	21.95	-112	11617	11.66	0	11617	5.2	0	11617	6.45	0	11617	3.24
G3	800	11511	21.95	-110	11621	14.77	0	11621	1.32	0	11621	3.11	0	11621	1.02
G4	800	11554	22.04	-92	11641	16.29	-5	11646	1.77	0	11646	5.46	0	11646	1.77
G5	800	11521	21.8	-110	11630	14.3	-1	11631	0.88	0	11631	7.69	0	11631	0.76
G6	800	2037	22.08	-140	2177	10.35	0	2177	1.16	0	2177	3.11	0	2177	1.5
G7	800	1889	22	-113	2000	14.72	-2	2002	0.82	0	2002	12.56	0	2002	0.53
G8	800	1873	21.94	-131	2001	16.66	-3	2004	4.26	0	2004	12.56	0	2004	3.5
G9	800	1907	21.86	-145	2046	11.94	-6	2052	1.19	0	2052	7.30	0	2052	1.88
G10	800	1875	21.96	-123	1998	14.99	0	1998	5.59	0	1998	14.04	0	1998	4.99
G11	800	560	3.18	-4	564	11.67	0	564	12.1	0	564	1.70	0	564	0.12
G12	800	546	3.17	-10	554	11.29	-2	556	11.54	0	556	1.08	0	556	0.56
G13	800	572	3.17	-10	578	11.12	-4	582	32.52	0	580	1.12	-2	582	4.52
G14	800	3023	7.02	-39	3058	17.76	-4	3062	799	0	3062	9.75	0	3062	90.68
G15	800	2996	7.01	-54	3049	15.2	-1	3050	692.96	0	3050	8.35	0	3050	55.84
G16	800	2994	7.02	-58	3047	15.83	-5	3052	82.82	0	3052	4.59	0	3052	32.82
G17	800	2997	6.99	-50	3043	17.16	-4	3047	778.67	0	3047	4.98	0	3047	200.67
G18	800	909	7.03	-83	991	10.82	-1	992	16.36	0	992	2.71	0	992	14.5
G19	800	823	7	-82	905	8.59	0	905	40.31	0	904	2.00	-1	905	3.51
G20	800	865	6.98	-76	941	6.09	0	941	2.48	0	941	0.79	0	941	1.52
G21	800	849	6.98	-81	930	9.97	0	930	34.71	0	930	2.43	0	930	50.41
G22	2000	13105	57.48	-254	13346	25.97	-13	13359	303.2	0	13359	14.83	0	13359	432.1
G23	2000	13120	57.36	-224	13319	27.67	-25	13344	132.13	0	13342	17.22	-2	13344	168.24
G24	2000	13115	57.34	-221	13322	25.87	-14	13336	102.75	0	13334	19.26	-2	13336	300.75
G25	2000	13125	57.41	-215	13314	26.36	-26	13340	308.51	0	13338	16.98	-2	13340	149.21
G26	2000	13160	57.25	-168	13300	27.64	-28	13328	366.09	0	13328	22.56	0	13328	433.68
G27	2000	3109	57.16	-232	3317	26.74	-24	3341	109.49	0	3335	13.95	-6	3341	140.64
G28	2000	3063	58.13	-235	3289	26.96	-9	3298	217.84	0	3297	10.85	-1	3298	198.23
G29	2000	3179	58.06	-224	3376	26.54	-27	3403	1.36	0	3402	8.86	-1	3403	3.26
G30	2000	3139	58.18	-273	3397	26.11	-15	3412	44.82	0	3412	15.21	0	3412	54.22
G31	2000	3092	58.13	-217	3296	25.43	-13	3309	263.21	0	3308	16.54	-1	3309	242.19
G32	2000	1382	16.88	-28	1410	61.07	0	1410	887.5	0	1410	5.69	0	1410	425.7
G33	2000	1344	17.01	-38	1378	59.8	-4	1382	856.8	0	1380	6.45	-2	1382	485.83
G34	2000	1350	16.88	-34	1382	52.09	-2	1384	536.12	0	1384	5.89	0	1384	189.27
G35	2000	7548	39.22	-138	7659	34.26	-27	7686	1312.42	0	7684	24.62	-2	7686	448.35
G36	2000	7530	39.08	-148	7655	33.79	-23	7678	1259.1	0	7674	33.69	-4	7678	634.11
G37	2000	7541	39.21	-148	7669	33.86	-20	7689	1543.36	0	7683	41.48	-6	7689	627.86
G38	2000	7537	39.23	-151	7662	34.63	-26	7688	922.66	0	7688	29.44	0	7688	688.32
G39	2000	2255	40.11	-153	2382	23.11	-26	2408	976.95	0	2408	21.64	0	2408	242.6
G40	2000	2189	40	-211	2386	24.82	-14	2400	1198.28	0	2399	13.34	-1	2400	354.5

Table 2 – continued from previous page

Instance	$\|V\|$	LNA [41]			MA-LZ [25]			MA-WH [40]			MA-ZLL [42]			ITS	
		f_{best}	$time(s)$	gap	f_{best}	$time(s)$	gap	f_{best}	$time(s)$	gap	f_{best}	$time(s)$	gap	f_{best}	$time(s)$
G41	2000	2234	40.03	-171	2383	25.78	-22	2405	546.57	0	2405	16.64	0	2405	82.55
G42	2000	2290	40.11	-191	2456	26.74	-25	2481	1513.96	0	2481	15.49	0	2481	286.18
G43	1000	6580	15.34	-79	-	-	-	6659	1.25	0	6659	5.15	0	6659	5.25
G44	1000	6548	15.33	-102	-	-	-	6650	1.18	0	6650	7.75	0	6650	2.09
G45	1000	6513	15.33	-141	-	-	-	6654	4.23	0	6654	4.91	0	6654	3.99
G46	1000	6538	15.33	-111	-	-	-	6649	10.48	0	6649	7.74	0	6649	30.12
G47	1000	6529	15.34	-128	-	-	-	6657	5.97	0	6657	4.72	0	6657	4.88
G48	3000	-	-	-	-	-	-	6000	1.42	0	6000	0.02	0	6000	0.97
G49	3000	-	-	-	-	-	-	6000	1.28	0	6000	0.02	0	6000	1.57
G50	3000	-	-	-	-	-	-	5880	33.89	0	5880	0.03	0	5880	50.64
G51	1000	3773	10.58	-74	-	-	-	3847	292.6	0	3847	6.37	0	3847	101.43
G52	1000	3788	10.61	-63	-	-	-	3851	814.96	0	3849	9.76	-2	3851	98.43
G53	1000	3784	10.6	-66	-	-	-	3850	516.28	0	3848	8.60	-2	3850	109.5
G54	1000	3789	10.63	-62	-	-	-	3851	551.51	0	3849	7.91	-2	3851	177.89
G55	5000	-	-	-	-	-	-	10299	2396.84	0	10281	37.77	-18	10299	2596.84
G56	5000	-	-	-	-	-	-	4016	1886.98	0	3994	37.10	-22	4016	1926.45
G57	5000	-	-	-	-	-	-	3488	4883.34	-2	3492	51.52	2	3490	610.16
G58	5000	18931	268.71	-345	19213	120.67	-63	19276	4276.67	0	19243	72.85	-33	19276	5102.34
G59	5000	5578	260.91	-507	5978	88.69	-107	6085	4446.16	0	6053	65.39	-32	6085	4902.13
G60	7000	-	-	-	-	-	-	14186	5508.45	0	14156	90.92	-31	14187	5678.63
G61	7000	-	-	-	-	-	-	5796	3755.71	0	5752	71.20	-44	5796	4072.54
G62	7000	-	-	-	-	-	-	4866	4652	0	4866	86.29	0	4866	1472.1
G63	7000	-	-	-	-	-	-	26754	5670.3	-234	26977	129.57	-11	26988	2256.66
G64	7000	-	-	-	-	-	-	8731	5793.56	-6	8697	131.66	-40	8737	6032.55
G65	8000	5418	290.72	-138	5534	463.44	-22	5556	5385.86	0	5554	86.70	-2	5556	2350.98
G66	9000	6194	391.03	-162	6324	850.69	-32	6352	6267.15	-4	6352	120.44	-4	6356	1323.15
G67	10000	6782	512.62	-156	6912	797.09	-26	6934	6203.44	-4	6936	74.49	-2	6938	1023.4
G70	10000	-	-	-	-	-	-	9580	7032.7	-1	9582	53.99	1	9581	1154.32
G72	10000	-	-	-	-	-	-	6990	7046.03	-4	6990	126.36	-4	6994	1201.97
G77	14000	-	-	-	-	-	-	9900	6752.26	-18	9910	186.29	-8	9918	2013.44
G81	20000	-	-	-	-	-	-	13978	7023.49	-52	14008	321.50	-22	14030	1953.23

Table 3: ITS needs much less time to attain the best objectives of the current best performing MA-WH algorithm [40] on the 12 largest instances with 7000 to 20000 vertices.

Instance	MA-WH [40]		ITS
	f_{best}	$time(s)$	$time(s)$
G60	14186	5508.45	5678.63
G61	5796	3755.71	4072.54
G62	4866	4652.00	1472.1
G63	26754	5670.30	238.16
G64	8731	5793.56	5532.55
G65	5556	5385.86	2350.98
G66	6352	6267.15	930.15
G67	6934	6203.44	1223.4
G70	9580	7032.70	1154.32
G72	6990	7046.03	970.92
G77	9900	6752.26	530.71
G81	13978	7023.49	486.70

We performed 10 independent runs of our ITS algorithm to solve each tested graph within a cutoff time limit of 3600 seconds and terminated each run once the best-known result was found. We used the same parameter settings as in the previous sections. For brevity purposes, we only summarize the main findings obtained from this experiment in the following. Our ITS algorithm was able to obtain the optimal solutions for the Walshaw and CG-Mesh graphs. In particular, for the CG-Mesh graphs, ITS reached the optimal solutions with computing times 5 to 10 times shorter than the time needed by the exact algorithm to complete its search. On the other hand, ITS failed to reach the optimal solutions for large SteinLib graphs. An interesting observation is that ITS performs well for graphs with a large minimum bisection value while the exact algorithm performs well for graphs with a small minimum bisection value (the latter was put forward in [13]). In this sense, we can consider that both algorithms may complement each other and can address graphs with different characteristics. The inferiority of ITS on graphs with small minimum bisection values is partly attributed to the ineffectiveness of the *c-swap* operator for this type of special graphs. Essentially, the *c-swap* operator only concentrates on swapping cutting edges, which proved to be effective for the graphs used to benchmark max-bisection algorithms, but becomes inefficient when the cutting edges are very limited as it is the case for the SteinLib graphs. Finally, we note that even if minimum bisection and maximum bisection are two different problems, both problems can be solved in the same way by the ITS algorithm.

4. Discussion

In this section, we investigate the roles of two important components of the ITS algorithm: the Last In First Out (LIFO) tie breaking strategy based on bucket sorting and the combined neighborhood using *1-move* and *c-swap*. The experiments of this section were based on a selection of 17 challenging instances while the tested ITS variants used the same stopping conditions as in the previous experiments.

4.1. Impact of the bucket-sorting based tie breaking strategies

The adopted bucket sorting is a crucial data structure to the effectiveness of the proposed algorithm, in particular to the LIFO tie breaking strategy. Recall that each bucket in the bucket array generally includes multiple vertices (organized into a circular double linked list) which lead to the same move gain. Apparently, no difference occurs among vertices in the same bucket. However, we assume that potential connections among vertices exist and the order of vertices being inserted into a bucket is worthy of a careful consideration. Based on this assumption, we proposed an improved LIFO insertion strategy (see Section 2.4), where a vertex is inserted at the head of the circular double linked list whenever its move gain is changed (i.e., its inserted position in the bucket array is changed accordingly), to ensure that this vertex will be selected when a tie break is required. The reason lies in the fact that if the move gain of a vertex u is changed because of moving a vertex v, then u has a higher opportunity to be moved during the following iterations. An exception is the insertion of tabu vertices at the tail of the circular double linked lists in order to penalize the recently moved vertices.

To assess the impact of bucket sorting on the performance of the ITS algorithm, we tested an ITS variant without the bucket sorting structure. That is, only a vector is kept to record the gains resulting from the 1-moves. In this case, to identify the vertex with maximum gain, the ITS variant has to scan the whole vector instead of looking at the top of the bucket array. When there are more than one vertex with maximum gain, ties are broken randomly. Table 4 (upper part) compares the standard ITS algorithm (with the bucket sorting structure and the LIFO tie breaking strategy, named ITS_{LIFO}) and the ITS variant without the bucket sorting structure (named $\text{ITS}_{No-bucket}$). From the results, we observe that removing the bucket sorting structure considerably degrades the performance of the ITS algorithm both in terms of the best and average solutions. This is confirmed by a small p-value of 3.738e-05 from the Friedman test in both cases. Moreover, compared to ITS_{LIFO}, $\text{ITS}_{No-bucket}$ generally requires more computing time to reach its best results (which are worse than those of ITS_{LIFO}). In conclusion, the experiment demonstrates the usefulness of the bucket sorting technique in the proposed ITS algorithm.

To further test the adopted LIFO tie breaking strategy, we compared LIFO with the Random Strategy (Random) and the First In First Out strategy (FIFO). The random strategy scans vertices of the same bucket in random order, no matter if a new vertex is inserted at the head or the tail of a circular double linked list. The FIFO strategy uses a queue structure, with the vertices in a bucket being scanned from the head to the tail like the LIFO strategy but with every vertex being inserted at the tail of the circular double linked list. For this experiment, we kept all the other components of the proposed ITS algorithm unchanged except for the tie breaking strategy.

Table 4 (lower part) reports the best objective value f_{best} and average objective value f_{avg} over 20 runs as well as the average time $time$ to reach f_{best}. From this table, we observe that the LIFO tie breaking strategy dominates the

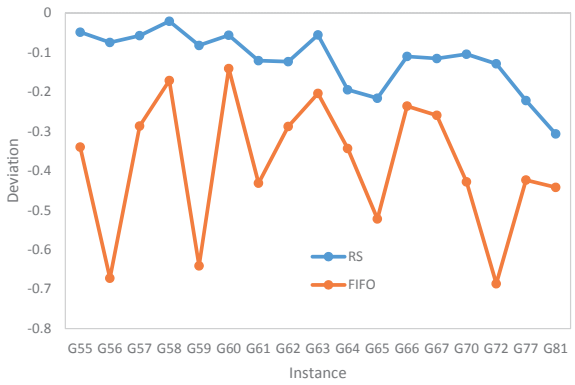
Table 4: Assessment of the bucket sorting structure and comparisons of the different tie-breaking strategies

Instance	ITS_{LIFO}			$ITS_{No-bucket}$		
	f_{best}	f_{avg}	$time(s)$	f_{best}	f_{avg}	$time(s)$
G55	10299	10290.8	2596.84	10296	10285.2	3755.78
G56	4016	4013.1	1926.45	4012	4007.9	4237.44
G57	3490	3487.8	610.16	3488	3478.2	5451.93
G58	19276	19265.9	5102.34	19272	19264.1	4759.58
G59	6085	6074.3	4902.13	6078	6063.6	4192.53
G60	14186	14176.5	5678.63	14170	14162.9	6012.47
G61	5796	5780.2	4072.54	5786	5770.4	3699.49
G62	4866	4860.1	1472.1	4860	4848.7	4275.62
G63	26988	26985.3	2256.66	26976	26967.0	5071.38
G64	8737	8712.1	6032.55	8725	8707.2	3975.71
G65	5556	5550.9	2350.98	5542	5535.6	4217.56
G66	6356	6352.0	1323.15	6345	6334.45	5274.54
G67	6938	6935.5	1023.4	6927	6920.5	4057.85
G70	9581	9576.3	1154.32	9564	9540.3	4538.75
G72	6994	6992.5	1201.97	6980	6975.2	5638.47
G77	9918	9915.1	2013.44	9890	9880.1	6972.68
G81	14030	14025.45	1953.23	13978	13950.45	7001.35
Instance	ITS_{FIFO}			ITS_{Random}		
	f_{best}	f_{avg}	$time(s)$	f_{best}	f_{avg}	$time(s)$
G55	10264	10255.6	5389.34	10294	10284.1	5560.1
G56	3989	3981.85	6883.49	4013	4009.6	5895.28
G57	3480	3473.2	5573.11	3488	3483.5	4560.34
G58	19243	19240.3	5991.60	19272	19261.9	6832.53
G59	6046	6041.8	7137.13	6080	6064.2	6102.8
G60	14166	14155.5	5365.37	14178	14170.4	6016.74
G61	5771	5758.3	5966.63	5789	5768.2	5319.93
G62	4852	4845.8	6084.48	4860	4857.5	6087.27
G63	26933	26914.0	5274.82	26973	26960.2	5752.03
G64	8707	8697.7	6462.01	8720	8711.5	5143.31
G65	5527	5520.4	6587.86	5544	5541.9	6136.65
G66	6341	6336.8	6728.68	6349	6340.9	7056.77
G67	6920	6914.6	5612.06	6930	6925.2	6835.17
G70	9540	9532.55	6177.37	9571	9561.1	6326.62
G72	6946	6941.7	6567.88	6985	6981.35	6964.13
G77	9876	9867.6	7139.18	9896	9888.7	6587.06
G81	13968	13955.5	5581.10	13987	13980.9	7019.52

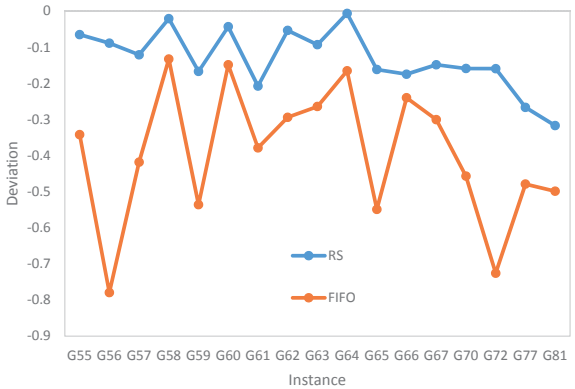
Random and FIFO strategies both in terms of solution quality and computing time. The superiority of the LIFO strategy can be observed in Figures 1(a) and 1(b) where we plot the deviation of the best and average objective values obtained by Random and FIFO from that of LIFO for each tested instance. If the absolute value of the deviation is smaller, then the corresponding objective value is better. From Figures 1(a) and 1(b), we observe that the deviation values are all negative, meaning that both Random and FIFO are inferior to LIFO in terms of the best and average objective values. In conclusion, this experiment demonstrates the benefits of the adopted LIFO tie breaking strategy based on bucket sorting.

4.2. Impact of the combined use of 1-move and c-swap operators

As described in Section 2.3, the ITS algorithm jointly employs the *1-move* and *c-swap* operators in a probabilistic way. To confirm the effectiveness of the



(a) The best objective deviation of the tie breaking strategies Random and FIFO from LIFO



(b) The average objective deviation of the tie breaking strategies Random and FIFO from LIFO

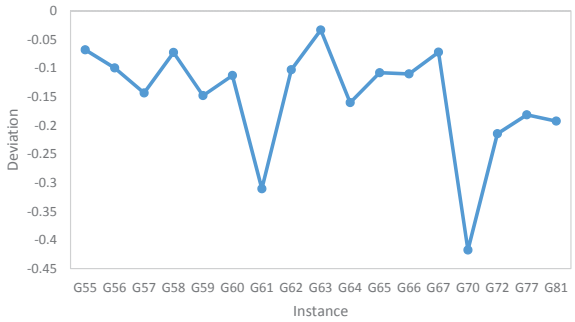
Figure 1: Analysis of the tie breaking strategies

combined use of these operators, we tested two ITS variants. The first variant (denoted by ITS$_{1\text{-}move}$) disables *c-swap* and only uses *1-move* (i.e., by removing lines 27-31 in Algorithm 1). The second variant (denoted by ITS$_{1\text{-}move+s\text{-}swap}$) just replaces *c-swap* by the conventional unconstrained swap operator (*s-swap*) (see Section 2.3). For both variants, we kept the other ITS components unchanged. We ran ITS (denoted by ITS$_{1\text{-}move+c\text{-}swap}$) as well as the two ITS variants under the same experimental conditions than before on the 17 selected instances. The results in terms of f_{best}, f_{avg} and $time$ are reported in Table 5.

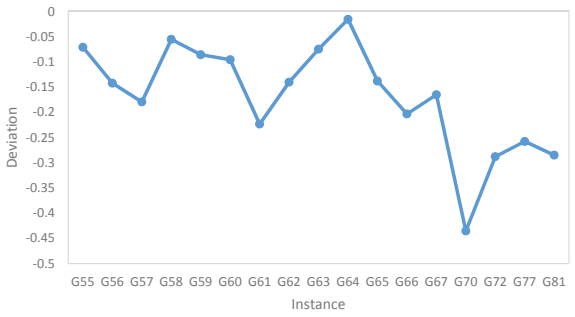
Table 5 indicates that the ITS algorithm with *1-move* and *c-swap* obtains better f_{best} and f_{avg} values for each tested instance. In addition, the joint use of *1-move* and *c-swap* requires the shortest computation time while producing results of much better quality. We also observe that the variant using *1-move* alone performs better than the variant using *1-move* and the conventional swap. This indicates that contrary to our fast *c-swap* operator, the expensive *s-swap* operator is not suitable here due to the high time complexity needed to explore the induced neighborhood of quadratic size $O(|V|^2)$. Furthermore, Figures 2(a) and 2(b) show respectively the best and average deviations with *1-move* from the corresponding objective values with *1-move + c-swap*, which clearly confirm the merit of the joint use of *1-move* and *c-swap*. We do not provide additional figures for the *1-move + s-swap* variant, but the observations made for the *1-move* variant hold as well. Moreover, Friedman statistical tests confirm that ITS with *1-move + c-swap* performs significantly better than the two other ITS variants in terms of best and average solution values. This experiment demonstrates the contribution of the constrained *c-swap* operator to the performance of the proposed ITS algorithm.

Table 5: Computational comparisons of the ITS algorithm using the *1-move* operator and the constrained swap operator (*c-swap*) with an ITS variant using *1-move* alone and another ITS variant using *1-move* and the conventional unconstrained swap operator (*s-swap*)

Instance	ITS$_{1\text{-}move+c\text{-}swap}$			ITS$_{1\text{-}move}$			ITS$_{1\text{-}move+s\text{-}swap}$		
	f_{best}	f_{avg}	$time(s)$	f_{best}	f_{avg}	$time(s)$	f_{best}	f_{avg}	$time(s)$
G55	10299	10290.8	2596.84	10292	10283.5	6192.26	10254	10231.45	6587.15
G56	4016	4013.1	1926.45	4012	4007.4	5595.33	4008	3988.9	5697.57
G57	3490	3487.8	610.16	3485	3481.5	5013.16	3466	3460.4	4989.16
G58	19276	19265.9	5102.34	19262	19255.25	6382.23	19190	19175.55	7014.74
G59	6085	6074.3	4902.13	6076	6069.1	4637.32	6043	6030.7	6910.92
G60	14186	14176.5	5678.63	14170	14163.0	5435.35	14101	14079.3	6514.35
G61	5796	5780.2	4072.54	5778	5767.3	6816.18	5709	5684.35	5638.13
G62	4866	4860.1	1472.1	4861	4853.3	4267.49	4821	4810.1	4968.75
G63	26988	26985.3	2256.66	26979	26965.1	4758.43	26910	26803.3	5017.68
G64	8737	8712.1	6032.55	8723	8710.75	6026.28	8705	8692.1	6987.14
G65	5556	5550.9	2350.98	5550	5543.2	5472.34	5318	5301.8	6541.25
G66	6356	6352.0	1323.15	6349	6339.1	5262.37	6036	6012.2	5746.28
G67	6938	6935.5	1023.4	6933	6924.0	6465.22	6714	6683.4	6357.17
G70	9581	9576.3	1154.32	9541	9534.6	4785.42	9013	8981.3	7104.38
G72	6994	6992.5	1201.97	6979	6972.7	6679.44	6034	5986.45	6879.32
G77	9918	9915.1	2013.44	9900	9889.6	6944.30	9062	9013.4	6245.84
G81	14030	14025.45	1953.23	14003	13985.5	7004.45	12002	11946.45	7008.46



(a) The best objective deviation of the ITS variant using the *1-move* operator alone from the ITS algorithm using both the *1-move* and *c-swap* operators



(b) The average objective deviation of the ITS variant using the *1-move* operator alone from the ITS algorithm using both the *1-move* and *c-swap* operators

Figure 2: Analysis of the combined use of the *1-move* operator and the constrained swap (*c-swap*) operator

5. Conclusion and perspectives

The iterated tabu search algorithm designed for the maximum bisection problem achieved a high performance level by including two distinct search operators which are applied in three search phases. The descent-based improvement phase uses the vertex move operator (*1-move*) to discover a first local optimum from a starting solution. The diversifying improvement phase jointly employs the *1-move* operator and the constrained swap operator in a probabilistic way (under the tabu search framework) to discover better solutions. The perturbation phase is applied as a means to achieve strong diversification to get out of deep local optimum traps. To obtain an efficient implementation of the proposed algorithm, we developed streamlining techniques and a LIFO tie-breaking strategy based on dedicated bucket data structures.

Experimental assessments on the set of 71 well-known benchmark instances with up to 20,000 vertices indicated that the proposed algorithm was able to obtain improved best results (new lower bounds) for 8 large instances and match

the best-known results for 61 instances. Comparisons with state-of-the-art algorithms showed that the ITS algorithm is competitive and dominates the reference algorithms in terms of solution quality. We also showed the interest of ITS to solve the related minimum bisection problem. Finally, we observe that the proposed ITS framework can be adapted conveniently to the maximum cut problem for which the combined application of two consecutive *1-move* operations, which is necessary to ensure balanced bisections for max-bisection is no longer required.

From the work presented in this paper, several perspectives can be contemplated for future studies. First, from the point of view of solution methods, it is known that hybrid approaches can lead to improved results compared with local search based single trajectory approaches. Thus, the proposed ITS algorithm can be advantageously integrated into a population-based memetic algorithm as its key local optimization procedure. For this, it would be necessary to seek a dedicated recombination operator able to generate promising offspring solutions from existing solutions. Second, the existing exact algorithms for the maximum bisection problem can only be applied to solve problems of moderate sizes (i.e., with up to a few hundreds of vertices). Thus there is a need to develop more powerful exact algorithms able to solve larger problems. For this purpose, the ITS algorithm can be used to compute high-quality lower bounds that can be used to better prune the search tree. In addition, most ideas of the ITS algorithm are general. As a result, they could be adapted to design effective heuristics for other graph partitioning problems. Finally, designing effective strategies for handling large sparse graphs is another interesting research direction.

Acknowledgment

We are grateful to the reviewers and the Area Editor in charge of the paper for their constructive comments which have helped us to improve the quality of the paper. This work was supported by 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

[1] 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.

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

[3] Bartz-Beielstein, T., Chiarandini, M., Paquete, L., & Preuss, M. (2010). *Experimental Methods for the Analysis of Optimization Algorithms*. Springer.

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

[5] Brunetta, L., Conforti, M., & Rinaldi, G. (1997). A branch-and-cut algorithm for the equicut problem. *Mathematical Programming*, *78*, 243–263.

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

[7] 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*, 503–521.

[8] 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*, 67–78.

[9] 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*, 1253–1266.

[10] Cordeau, J.F., & Maischberger M. (2012). A parallel iterated tabu search heuristic for vehicle routing problems. *Computers and Operations Research*, 39(9), 2033–2050.

[11] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to Algorithms, 2nd edition.* McGraw-Hill Higher Education.

[12] Dang, C., He, L., & Hui, I. K. (2002). A deterministic annealing algorithm for approximating a solution of the max-bisection problem. *Neural Networks*, *15*, 441–458.

[13] Delling, D., Fleischman, D., Goldberg, A.V., Razenshteyn, I., & Werneck, R.F. (2015). An exact combinatorial algorithm for minimum graph bisection. *Mathematical Programming*, *153*, 417–458.

[14] 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 of 2001 IEEE International Conference on Data Mining*, pp. 107–114, IEEE.

[15] Elf, M., Jünger, M., & Rinaldi, G. (2003). Minimizing breaks by maximizing cuts. *Operations Research Letters*, *31*, 343–349.

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

[17] 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.

[18] Glover, F., & Laguna, M. (1999). *Tabu Search*. Springer.

[19] Hagen, L. W., Huang, D. J., & Kahng, A. B. (1997). On implementation choices for iterative improvement partitioning algorithms. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, *16*, 1199–1205.

[20] 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*, 211–218.

[21] Karisch, S. E., Rendl, F., & Clausen, J. (2000). Solving graph bisection problems with semidefinite programming. *INFORMS Journal on Computing*, *12*, 177–191.

[22] Karp, R. M. (1972). *Reducibility among combinatorial problems*. Springer.

[23] Kernighan, B. W., & Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, *49*, 291–307.

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

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

[26] Ling, A.-F., Xu, C.-X., & Tang, L. (2008). A modified VNS metaheuristic for max-bisection problems. *Journal of Computational and Applied Mathematics*, *220*, 413–421.

[27] López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., & Birattari, M. (2011). *The irace package, iterated race for automatic algorithm configuration*. TR.

[28] Lourenço, H.R., Martin O., & Stützle T. (2010). Iterated local search: framework and applications. Handbook of Metaheuristics. Kluwer, International Series in Operations Research & Management Science 146: 363–397.

[29] Lü, Z.P., Hao, J.K., & Glover F. (2011). Neighborhood analysis: a case study on curriculum-based course timetabling. *Journal of Heuristics, 17(2)*, 97–118.

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

[31] Murty, K. G., & Kabadi, S. N. (1987). Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, *39*, 117–129.

[32] Papadimitriou, C.H., & Papadimitriou, K. (1982). Combinatorial optimization: algorithms and complexity. Prentice-Hall.

[33] Palubeckis, G., Ostreika, A., & Rubliauskas, D. (2014). Maximally diverse grouping: an iterated tabu search approach. *Journal of the Operational Research Society*, *66*, 579–592.

[34] Qin, T, Peng, B., Benlic, U., Cheng, T.C.E., Wang, Y., & Lü Z.P. (2015). Iterated local search based on multi-type perturbation for single-machine earliness/tardiness scheduling. *Computers and Operations Research, 61*, 81–88.

[35] 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*, 16–24.

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

[37] Silva, M.M., Subramanian, A., & Ochi L.S. (2015). An iterated local search heuristic for the split delivery vehicle routing problem. *Computers and Operations Research, 53*, 234–249.

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

[39] Wu, Q., & Hao, J.K. (2012). A memetic approach for the max-cut problem. In *Parallel Problem Solving from Nature-PPSN XII, Lecture Notes in Computer Science, 7492,* pp. 297–306. Springer.

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

[41] Xu, F., Ma, X., & Chen, B. (2011). A new Lagrangian net algorithm for solving max-bisection problems. *Journal of Computational and Applied Mathematics*, *235*, 3718–3723.

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

Appendix A. An example of the bucket structure

Fig. A.3 shows an illustrative example of the bucket structure for max-bisection. The graph (Fig. A.3, left) has 8 vertices belonging to the two subsets S_1 and S_2 (edge weights are supposed to be equal to 1). The bucket structure for this graph is shown in Fig. A.3 (right). One observes that the gain of moving vertex c or h to subset S_1 equals 0, then those two vertices are stored in the entry of B_1 with index 0. Notice that vertices c and h are managed as a circular double linked list. The array AI shown at the bottom of Fig. A.3 manages position indexes for all vertices. For simplicity, we do not show all the links in the figure.

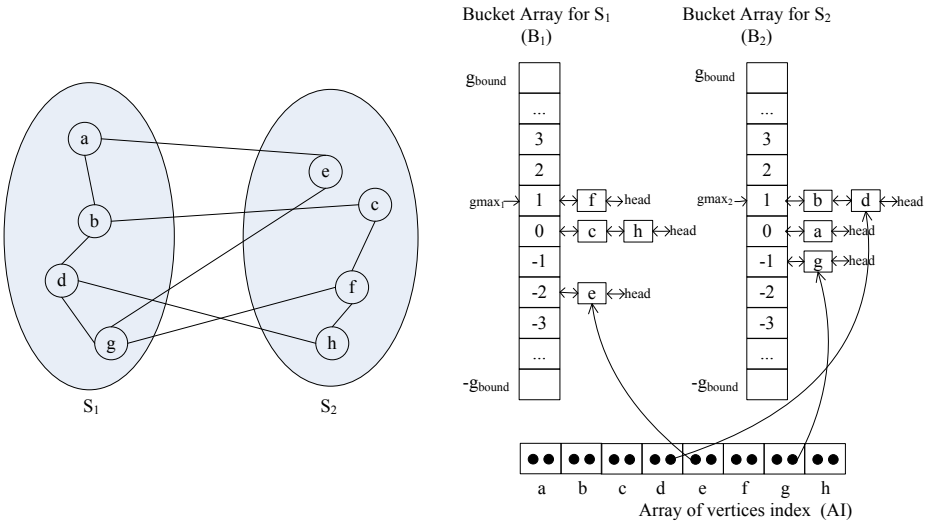


Figure A.3: An example of bucket structure for max-bisection