Memetic search for the max-bisection problem

Qinghua Wu and Jin-Kao Hao *

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

Computers and Operations Research 40(1): 166-179, 2013
DOI: http://dx.doi.org/10.1016/j.cor.2012.06.001

Abstract

Given an undirected graph $G = (V, E)$ with weights on the edges, the max-bisection problem (MBP) is to find a partition of the vertex set V into two subsets V_1 and V_2 of equal cardinality such that the sum of the weights of the edges crossing V_1 and V_2 is maximized. Relaxing the equal cardinality constraint leads to the max-cut problem (MCP). In this work, we present a memetic algorithm for MBP which integrates a grouping crossover operator and a tabu search optimization procedure. The proposed crossover operator preserves the largest common vertex groupings with respect to the parent solutions while controlling the distance between the offspring solution and its parents. Extensive experimental studies on 71 well-known G-set benchmark instances demonstrate that our memetic algorithm improves, in many cases, the current best known solutions for both MBP and MCP.

Keywords: Max-bisection, max-cut, memetic search, evolutionary computing, tabu search, optimization.

1 Introduction

Let $G = (V, E)$ be an undirected graph with vertex set $V = \{1, ..., n\}$ and edge set $E \subset V \times V$, each edge $\{i, j\} \in E$ being associated with a weight $w_{ij} \in Z$. The well-known *max-cut* problem (MCP) is to partition the vertex set V into two disjoint subsets $V_1 \subset V$ and $V_2 = V \setminus V_1$ such that the sum of the weights

* Corresponding author.
Email addresses: wu@info.univ-angers.fr (Qinghua Wu), hao@info.univ-angers.fr (Jin-Kao Hao).

of the edges from E that have one endpoint in each subset is maximized, i.e., $max \sum_{i \in V_1, j \in V_2} w_{ij}$. MCP is one of the first 21 NP-complete problems studied in [21]. When the two subsets V_1 and V_2 are required to have the same cardinality (assuming that n is even), the max-cut problem becomes the *max-bisection* problem (MBP) which remains NP-complete in the general case [13]. Both the max-bisection and max-cut problems has many applications such as statistical physics, classification, social network analysis, and VLSI design [6].

In this work, we are basically interested in the max-bisection problem. Given max-cut is a relaxed max-bisection problem, advances in solving max-bisection can benefit directly the solving of the max-cut problem.

There are two related 'dual' partition problems known as *minimum cut* and *minimum bisection* that aim to determine a two-way partition of a graph while minimizing the sum of the weights of the cutting edges (equal cardinality constraint is required for minimum bisection). Notice that in the general case, these minimization problems are different from the max-bisection and max-cut problems considered in this work which concern the two-way partition problems with the *maximization* criterion. Finally, graph partition problems with the minimization criterion have received much attention in the literature, leading to several well-known public-domain software packages like Chaco [18], Jostle [37], and Metis [22] (see [4] for a recent review).

The computational challenge of the general max-cut and max-bisection problems has motivated a variety of solution approaches including exact methods, approximation algorithms and metaheuristic methods. Examples of approximation algorithms based on semidefinite programming are described in [10,15,16,20,39]. These approaches provide a performance guarantee, but do not compete well with other methods in computational testing. Two recent examples of exact methods are described in [24,34] which are based on the cut and price approach and the branch and bound approach respectively. While these methods have the theoretical advantage of finding optimal solutions to a given problem, their applications are generally limited to problems with no more than a few hundred vertices.

For larger problem instances, a number of heuristics and metaheuristics are often used to find approximate solutions of good quality with a reasonable computing time. This includes for the max-bisection problem deterministic annealing [7], Lagrangian net [38] and variable neighborhood search [25]. There are many heuristics and metaheuristics for the max-cut problem including simulated annealing [1], rank-2 relaxation heuristic [5], GRASP [9], diversification driven tabu search [23], advanced scatter search [28], global equilibrium search [35], and probabilistic tabu search [36].

In this paper, we present a memetic algorithm for the max-bisection problem

(denoted by MAMBP). The proposed algorithm integrates three complementary key components which jointly ensure the high efficiency of the search process. First, to generate promising new solutions, we introduce a dedicated crossover operator which tries to preserve groups of vertices that are shared by parent solutions. The design of this crossover operator relies on the observation that, given a set of high quality bisections of a graph, there is always a large number of vertices grouped together throughout these bisections. Second, we devise a tabu search optimization procedure for the purpose of intensified search around a given solution. The tabu search procedure uses a vertex move neighborhood and incremental evaluation techniques for a fast neighborhood examination. Finally, to maintain a healthy diversity of the population, we employ a pool updating strategy which takes into account both the solution quality and the distance between solutions.

We show extensive experimental results on 71 well-known G-set benchmark graphs (with 800 to 20000 vertices) in the literature, showing that the proposed algorithm achieves highly competitive results with respect to the existing max-bisection heuristics. Moreover, when considering the relaxed max-cut problem, the results produced by our MAMBP algorithm remain highly competitive even when they are compared to those obtained by dedicated max-cut algorithms; for 31 max-cut instances, MAMBP improves the previous best known max-cut solutions of the literature.

In the next section, the components of our memetic algorithm are described, including the tabu search procedure, the crossover operator and the pool replacement strategy. Section 3 is dedicated to computational results and detailed comparisons with other state-of-the-art algorithms in the literature. Section 4 investigates several essential parts of the proposed memetic algorithm, followed by concluding remarks given in Section 5.

2 Memetic Algorithm

Memetic algorithms are known to be an effective approach in solving a number of hard combinatorial optimization problems [29,30,17]. Typically, a memetic approach repeatedly alternates between a recombination (or crossover) operator to generate solutions located in promising regions in the search space and a local optimization procedure to search around the newly generated solutions. It is commonly admitted that the success of this approach depends critically on the recombination operator. In order to be effective, the recombination operator must be adapted to the problem being solved and should be able to transmit meaningful features from parents to offspring.

The general scheme of our memetic approach for MBP is summarized in Algo-

rithm 1. Basically, our memetic algorithm begins with an initial population of solutions which are first improved by the local optimization procedure based on tabu search [14] (lines 1–5, Sections 2.2 and 2.3) and then repeats an iterative process for a fixed number of times (generations) (lines 6–13). At each generation, two solutions are selected to serve as parents (Section 2.4). The crossover operator is applied to the parents to generate a new offspring solution (Section 2.5) which is further improved by the tabu search optimization procedure (Section 2.3). Finally, we apply a quality-and-diversity based rule to decide whether the improved offspring solution can be inserted into the population (Section 2.6). In the following subsections, we give more details on the components of our memetic algorithm.

2.1 Search space and cost function

Recall that MBP consists in partitioning the vertex set V into two subsets of equal cardinality such that the weights on the edges between the two subsets is maximized. As such, we define the search space explored by our memetic algorithm as the set of all possible partitions of V into 2 disjoint subsets of equal cardinality (also called bisections), i.e., $\Omega = \{\{V_1, V_2\} : |V_1| = |V_2| = \frac{|V|}{2}, V_1 \bigcap V_2 = \varnothing\}$. Clearly, the size of Ω is given by $C(|V|, |V|/2)$.

Given a bisection $I = \{V_1, V_2\} \in \Omega$, the cost function (also called the fitness function) $f(I)$ sums up the weights of the edges between the two subsets V_1 and V_2 such that:

$$f(I) = \sum_{i \in V_1, j \in V_2} w_{ij} \tag{1}$$

Algorithm 1 Memetic algorithm for the max-bisection problem

Require: A weighted graph $G = (V, E, \omega)$, population size p
Ensure: The best solution I^* found
1: $Pop = \{I_1, ..., I_p\} \leftarrow Initial\_Population()$
2: $I^* \leftarrow Best(Pop)$
3: **for** $i = 1$ to p **do**
4: $I_i \leftarrow Tabu\_Search(I_i)$ /* Section 2.3 */
5: **end for**
6: **while** the stop criterion is not met **do**
7: Select randomly two solutions (parents) I_i and I_j from Pop /* Section 2.4 */
8: $I_0 = Cross\_Over(I_i, I_j)$ /* Section 2.5 */
9: $I_0 \leftarrow Tabu\_Search(I_0)$ /* Section 2.3 */
10: **if** $f(I_0) > f(I^*)$ **then**
11: $I^* \leftarrow I_0$ /* Update the best solution found so far */
12: **end if**
13: $Pop \leftarrow Pool\_Updating(I_0, Pop)$ /* Section 2.6 */
14: **end while**

Then, for two bisections $I_A \in \Omega$ and $I_B \in \Omega$, I_A is better than I_B if and only if $f(I_A) > f(I_B)$. The goal of the max-bisection problem is to find:

$$\arg\max_{I \in \Omega} f(I)$$

Given the size of the search space Ω, it is particularly challenging to find an exact solution or an approximate solution of high quality.

2.2 Initial population

The solutions (individuals) of the initial population are created as follows. For each individual, an equal sized partition is first created at random and then improved by the tabu search procedure (see Section 2.3). The improved solution is added into the population if this solution is not already present in it. Otherwise, this solution is discarded and a new random (equal sized) partition is created. This procedure is iterated until the population is filled with p solutions (p is the population size). This simple procedure provides an initial population of diverse solutions of good quality.

2.3 The Perturbation-based Tabu Search Procedure

Our tabu search (TS) procedure aims to improve a given solution I and plays the key role of local optimization within our memetic algorithm. Basically, our tabu search procedure repeatedly alternates between an intensification phase ensured by the basic tabu search and a diversification phase controlled by a perturbation mechanism [14]. Algorithm 2 describes this perturbation-based tabu search procedure, whose components are detailed in the following subsections.

2.3.1 Neighborhood and move

Given a bisection $I = \{V_1, V_2\}$ with $|V_1| = |V_2|$, the basic idea of the neighborhood explored by our tabu search consists in moving first a vertex from subset V_1 to V_2, accompanied by moving another vertex from V_2 to V_1. Note that in such a way, the balance between the two parts of the bisection is always maintained (i.e., $|V_1| = |V_2|$).

The key concept related to our neighborhood is the move gain, which indicates how much a solution (bisection) is improved according to the optimization objective f (Eq. 1, Section 2.1) if a vertex is moved from its subset to the

other subset. For large problem instances, it is imperative to be able to rapidly determine the move gain of a move. In our implementation, we use a fast incremental evaluation technique which is based on a streamlined calculation for updating the move gain after each move. More formally, let Δ_v be the move gain of moving vertex v to the other subset. Then initially, each move value can be calculated in linear time using the formula :

$$\Delta_v = \begin{cases} \sum\limits_{x \in V_1, x \neq v} w_{vx} - \sum\limits_{y \in V_2} w_{vy}, & \text{if } v \in V_1 \\ \sum\limits_{y \in V_2, y \neq v} w_{vy} - \sum\limits_{x \in V_1} w_{vx}, & \text{otherwise.} \end{cases} \quad (2)$$

Once a move is performed, one just needs to update a subset of move gains

Algorithm 2 Perturbation-based tabu search for the max-bisection problem

Require: A weighted graph $G = (V, E, \omega)$, initial bisection $I = \{V_1, V_2\}$, number lr of tabu search iterations, number cr of consecutive iterations before triggering a perturbation

Ensure: The best bisection I^*

1: $Iter \leftarrow 0$ /* Iteration counter */
2: $I^* \leftarrow I$ /* I^* records the best bisection found so far */
3: Initiate the tabu list and tabu tenure /* See Section 2.3.2 */
4: **for** each vertex $v \in V$ **do**
5: Compute the move gain Δ_v according to Eq. 2.
6: **end for**
7: **while** $Iter < lr$ **do**
8: Choose a best allowed (i.e., not forbidden by the tabu list) vertex $v_1 \in V_1$ according to max-move gain criterion (ties are broken randomly) /* See this section */
9: Generate intermediate solution I by moving v_1 from V_1 to V_2 (i.e., $V_1 = V_1 \setminus \{v_1\}$ and $V_2 = V_2 \cup \{v_1\}$)
10: Add v_1 in the tabu list and update the move gain Δ_v for each $v \in V$
11: Choose a best allowed vertex $v_2 \in V_2$ (i.e., v_2 is not forbidden by the tabu list or v_2 leads to a new solution better than solution I^*) according to max-move gain criterion (ties are broken randomly)
12: Generate new solution I by moving v_2 from V_2 to V_1, (i.e., $V_1 = V_1 \cup \{v_2\}$ and $V_2 = V_2 \setminus \{v_2\}$)
13: Add v_2 in the tabu list and update the move gain Δ_v for each $v \in V$
14: **if** $f(I) > f(I^*)$ **then**
15: $I^* \leftarrow I$ /* Update the best solution found so far */
16: **end if**
17: $Iter \leftarrow Iter + 1$
18: **if** I^* not improved after cr iterations **then**
19: $I \leftarrow Perturb(I)$ /* Apply perturbations to I, Section 2.3.3 */
20: **end if**
21: **end while**

affected by this move. Specifically, the following abbreviated calculation can be performed to update the move gains upon moving v from its set to the other set:

- $\Delta_v = -\Delta_v$
- for each $u \in V - \{v\}$,

$$\Delta_u = \begin{cases} \Delta_u - 2 \times w_{uv}, & \text{if } u \text{ is in the same set as } v \text{ before moving } v \\ \Delta_u + 2 \times w_{uv}, & \text{otherwise.} \end{cases}$$

An illustration of the proposed approach for initializing and updating the move gains is provided in Figure 1.

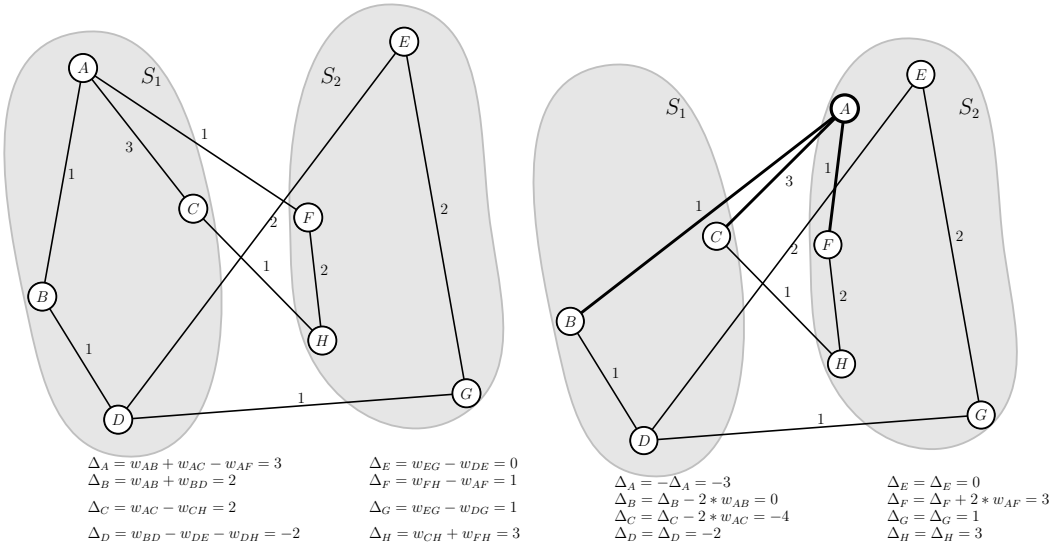

Fig. 1. An example of the initialization (left) and update (right) of the move gain

Our tabu search procedure then restricts consideration to those vertices which are not forbidden by the tabu list (see Section 2.3.2) and have the largest Δ value. At each iteration of our tabu search, we first select a non tabu vertex in V_1 with the largest Δ value and move it from V_1 to V_2. Then, we choose another non tabu vertex in V_2 with the best move gain and move it from V_2 to V_1. In the case that two or more non tabu moves have the same largest Δ value, one of them is chosen at random. Once a vertex v is changed from its original set to the other set, it is classified tabu for the next tl iterations, during which v is forbidden to be moved back to its original set. A simple aspiration criterion is applied that permits a vertex to be selected in spite of being tabu if it leads to a solution better than the current best solution. Notice that aspiration is only applicable to the selection of the second vertex from V_2 (see Algorithm 2, line 11) since the selection of the first vertex from V_1 leads to an intermediate imbalanced bisection which is not a feasible solution. The TS process stops when a maximum allowed number (lr) of iterations is reached.

2.3.2 Tabu list and tabu tenure management

Within TS, a tabu list is introduced to forbid the previously visited solutions to be revisited. In our TS algorithm, each time a vertex v goes from its original subset to the opposite set, it is forbidden to bring back v to its original set for a certain number of iterations (called the tabu tenure). Inspired by the tabu mechanism proposed in [11], we employ a particular technique whose goal is to vary the tabu tenure throughout the search.

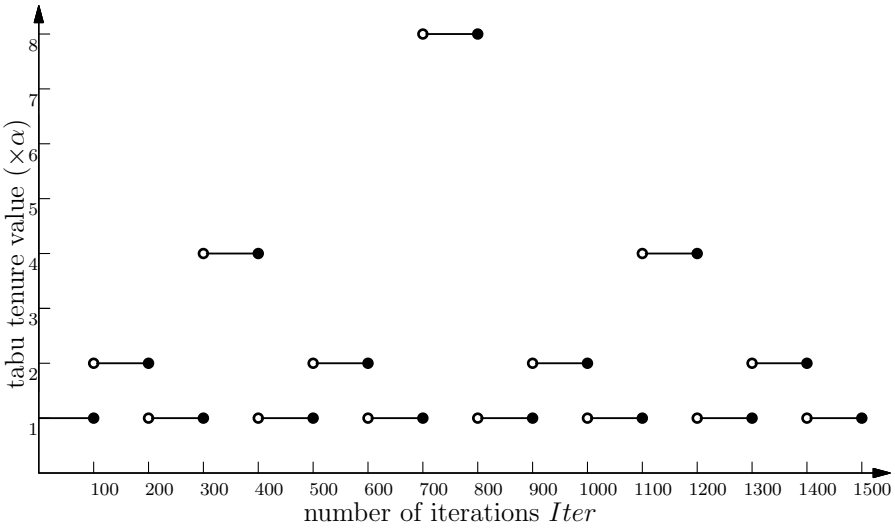

Fig. 2. An illustration of the step function (one period) used for tuning the tabu tenure.

The tabu tenure is dynamically tuned by a periodic step function T defined over the number of iterations $Iter$ (see Figure 2 for an example). Each period of the step function is composed of 1500 iterations divided into 15 steps or intervals $[x_i..x_{i+1}-1]_{i=1,2,...,15}$ with $x_1 = 1, x_{i+1} = x_i+100$. The tabu tenure changes dynamically during the search and according to the current iteration number, it takes one of four possible values $(\alpha, 2 \times \alpha, 4 \times \alpha, 8 \times \alpha)$ where α is a parameter. Precisely, for an iteration $Iter \in [x_i..x_{i+1} - 1]$, the tabu tenure $T(Iter)$ is given by $(y_i)_{i=1,2,...,15} = \alpha \times (1, 2, 1, 4, 1, 2, 1, 8, 1, 2, 1, 4, 1, 2, 1)$. Therefore, the tabu tenure is equal to α for the first 100 iterations [1..100], then $2 \times \alpha$ for iterations from [101..200], followed by α again for iterations [201..300] and $4 \times \alpha$ for iterations [401..500] etc. After reaching the largest value $8 \times \alpha$ for iterations [701..800], the tabu tenure drops again to α for the next 100 iterations and so on. This function repeats periodically this variation scheme after every 1500 iterations.

We have also experimented the four values $(\alpha, 2 \times \alpha, 4 \times \alpha, 8 \times \alpha)$ in a static way, i.e., applying each of them during the whole search. Experimental results showed that the dynamic tabu tenure method performs better than the static method and allows the search to effectively escape from local optima.

2.3.3 Perturbation mechanism

The above basic tabu search is generally able to attain solutions of good quality. However, it may happen that it gets stuck in deep local optima. In our case, the search is judged stagnating each time the best bisection I^* found so far is not further improved after a number cr of consecutive iterations.

To help the search to escape from such deep local optima, we apply a simple perturbation mechanism to the current solution to bring diversification into the search. The perturbation consists in swapping a number of pairs of vertices in the following way. For each swap, we randomly choose one vertex u from V_1 and another vertex v from V_2, and then swap u and v (i.e., $V_1 = V_1 \bigcup \{v\} \backslash \{u\}$ and $V_2 = V_2 \bigcup \{u\} \backslash \{v\}$). This process is repeated γ times where γ is a parameter which indicates the strength of the perturbation.

2.4 Parent selection

Our memetic algorithm is a steady-state algorithm such that at each generation one new solution is generated by combining two parent solutions chosen from the population. To choose the parents, several selection methods can be applied. In our case, we have experimented three well-known strategies: roulette-wheel, tournament and random [2]. With roulette-wheel selection, an individual is selected with a probability which is proportional to its fitness (objective) value. For tournament selection (the standard 2-way deterministic tournament), two individuals are taken at random and the best of the two is selected. Random selection chooses each individual with equal probability. Experimental results with these selection methods have shown that within the context of this work, no significant difference was observed. This can be explained intuitively as follows. In fact, since the individuals are improved by the powerful tabu search procedure, they are generally of good quality, implying that their fitnesses are not so different in the objective space (though the bi-sections they represent may be quite different due to the pool updating strategy used in Section 2.6). Under such a circumstance, it is clear that roulette-wheel, tournament and random would lead to a similar selection effect. For this reason, we have decided to retain the simple random selection in our algorithm.

2.5 Crossover operator

Crossover is another key component of our memetic algorithm. In practice, instead of applying blind crossovers, it is often preferable to devise a dedicated recombination operator that have strong "semantics" with respect to the prob-

lem being solved [17]. In the case of MBP, a bisection of a graph $G = (V, E)$ corresponds to a partition of V into two disjoint groups of vertices. With this point of view, MBP can be considered as a grouping problem [8]. For grouping problems, the crossover operators should manipulate groups of objects rather than individual objects. Such an approach for designing crossover operators has been successfully applied to solve a number of grouping problems such as graph coloring [12,27,32], bin packing [8] and graph partitioning [3,11].

In this work, we propose a dedicated grouping crossover operator for MBP, the proposed crossover operator is constrained to conserve subsets (or grouping vertices) of the vertex partitions of parents in offspring solutions. The rationale behind this approach is based on a careful analysis of local optimal solutions which discloses that high quality local optima share many grouping vertices (see Section 4). Therefore, if a set of vertices are always grouped together throughout a set of high quality solutions, these vertices are considered to have a high chance to be part of a globally optimal solution.

The proposed crossover operator follows the general principle presented above and operates in two sequential steps. We first create a partial bisection of maximal size by conserving the largest common vertex groupings with respect to the two selected parents. In order to obtain a full bisection, we complete this partial bisection by redistributing the remaining unassigned vertices based on a greedy construction strategy.

More precisely, given two parents $I^a = \{V_1^a, V_2^a\}$ and $I^b = \{V_1^b, V_2^b\}$, the partial offspring $I^c = \{V_1^c, V_2^c\}$ is constructed such that $V_1^c = V_1^a \bigcap V_1^b$ and $V_2^c = V_2^a \bigcap V_2^b$. It should be noted that if $|V_1^a \bigcap V_1^b| < |V_2^a \bigcap V_2^b|$, we first exchange the two parts of I^b before constructing the offspring I^c for the purpose of preserving the largest common grouping vertices (see Figure 2 for an example). For the constructed partial offspring $I^c = \{V_1^c, V_2^c\}$, it can be shown that $|V_1^c| = |V_2^c|$.

When the partial offspring $I^c = \{V_1^c, V_2^c\}$ is constructed, the vertices in $V_1^a \setminus V_1^c$ and $V_2^a \setminus V_2^c$ are left unassigned. For the vertices in $V_1^a \setminus V_1^c$, they are grouped together in both parent solutions I^a and I^b (in fact, $V_1^a \setminus V_1^c \subset V_1^a$ and $V_1^a \setminus V_1^c \subset V_2^b$, see Figure 3 for an example). Similarly, the vertices in $V_2^a \setminus V_2^c$ are also grouped together in both I^a and I^b. Thus, these left vertices should be dealt with care in order to generate a diversified offspring solution. In the following, we use a diversification-guided strategy to redistribute these unassigned vertices by taking into account both the solution quality of the offspring and its distance to its parent solutions.

Given the partial solution $I^c = \{V_1^c, V_2^c\}$, we use two greedy functions that take into account the contribution of the left vertices to the objective function of I^c. Specifically, for each $v \in V_1^a \setminus V_1^c$, we define $\sigma(v) = \sum_{u \in V_1^c} w_{uv}$ and $\sigma'(v) =$

$\sum_{u \in V_2^c} w_{uv}$. Then the vertices in $V_1^a \setminus V_1^c$ are assigned to the two parts of I^c one by one. At the first step, we examine all the vertices in $V_1^a \setminus V_1^c$ to identify the vertex with the largest σ value and move it from $V_1^a \setminus V_1^c$ to V_1^c. Afterward, V_2^c is considered, we identify the vertex in $V_1^a \setminus V_1^c$ with the largest σ' value and assign it to V_2^c. Then at each step of our procedure, V_1^c and V_2^c are considered in turn to distribute the remaining vertices in $V_1^a \setminus V_1^c$ (see Figure 2). Once all vertices in $V_1^a \setminus V_1^c$ are assigned to the two subsets of I^c, the same procedure is applied to handle the vertices in $V_2^a \setminus V_2^c$. Finally, one observes that the offspring I^c generated by our crossover operator has essentially the same distance to its two parents and is of relatively high quality.

2.6 Pool updating strategy

When an offspring solution is created by the crossover operator presented in the last section and improved by the tabu search algorithm described in Section 2.3, we decide whether the improved offspring should be inserted into the population and which existing solution of the population should be replaced. Basically, our decisions are made based on both solution quality and distance between solutions in the population [26,27,32]. Therefore, we first define the distance between two solutions before presenting the used pool updating strategy.

Definition 1. Distance between two bisections: Given two bisections $I^a = \{V_1^a, V_2^a\}$ and $I^b = \{V_1^b, V_2^b\}$, the distance between I^a and I^b (call it d) can

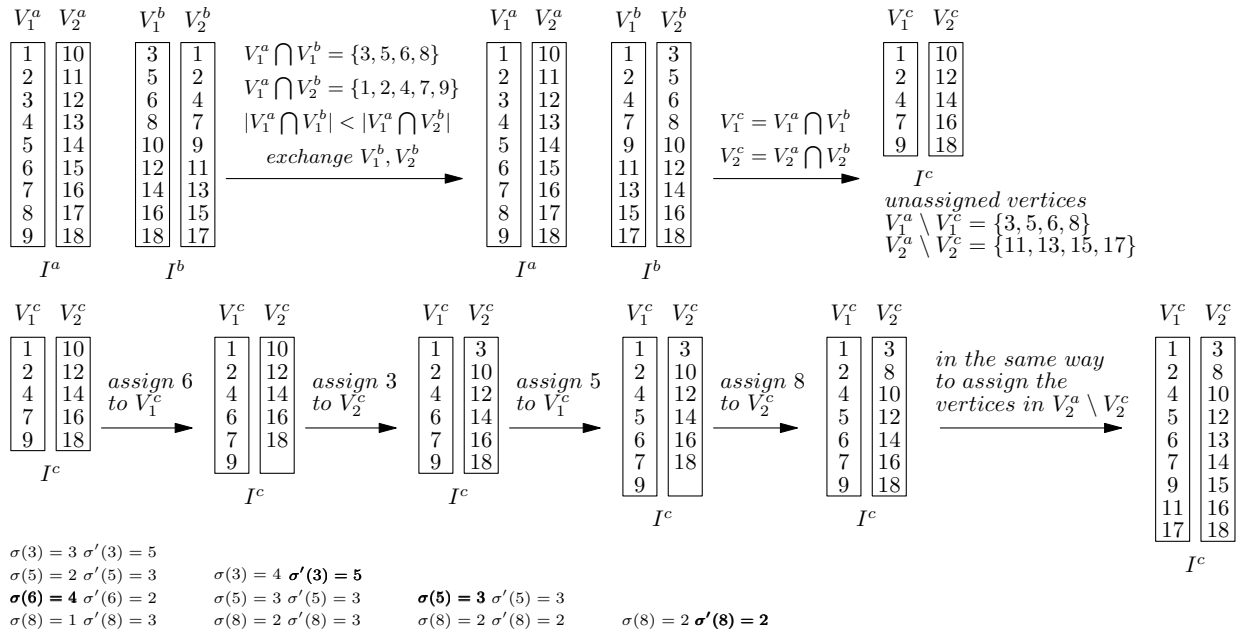

Fig. 3. An illustration of the crossover operator.

be defined as the minimum number of one-move steps necessary to transform I^b to I^a. A move step corresponds to the operation of moving one vertex from one side of the bisection to the other side. More formally, the distance $d(I^a, I^b)$ is determined using the formula $d(I^a, I^b) = |V| - s(I^a, I^b)$ where $s(I^a, I^b)$ is a similarity function which corresponds to the number of vertices that do not need to be moved to transform I^b to I^a [33]. This similarity function can be defined as follows. If $|V_1^a \bigcap V_1^b| < |V_1^a \bigcap V_2^b|$, we first exchange the two parts of I_b, then $s(I^a, I^b)$ is defined as $s(I^a, I^b) = |(V_1^a \bigcap V_1^b) \cup (V_2^a \bigcap V_2^b)|$.

Definition 2. Distance between one bisection and a population: Given a population $Pop = \{I^1, ..., I^p\}$ and the distance d_{ij} between any two bisections I^i and I^j $(i, j = 1, ..., p, i \neq j)$, the distance between a bisection I^i $(i = 1, ..., p)$ and the population Pop is defined as the minimum distance between I^i and any other bisection in the population:

$$D_{i,Pop} = min\{d_{ij} | I^j \in Pop, j \neq i\} \tag{3}$$

The general scheme of our pool updating strategy is described in Algorithm 3. In order to update the population, the new generated offspring I^0 is tentatively inserted into the population, i.e., $Pop' = Pop \bigcup \{I^0\}$. Then, the worst solution (denoted by I^w) in the population $Pop' = \{I^0, ..., I^p\}$ is identified. If I^w is not the offspring I^0, then I^0 is inserted into the population and replaces the worst solution I^w, i.e., $Pop = Pop \cup \{I^0\} \backslash \{I^w\}$. To determine the worst solution in the population, we use a quality-and-distance scoring function which was originally proposed in [26] and is defined as follows:

$$g(I^i) = \beta \widetilde{A}(f(I^i)) + (1 - \beta)\widetilde{A}(D_{i,Pop}) \tag{4}$$

where $f(I^i)$ is the objective function value defined in section 2.1, β is a param-

Algorithm 3 Pool Updating Rule

Require: Population $Pop = \{I^1, ..., I^p\}$ and offspring solution I^0
Ensure: Updated population $Pop = \{I^1, ..., I^p\}$
1: Temporarily insert I^0 into the population: $Pop' = Pop \cup \{I^0\}$
2: **for** $i = 0, ..., p$ **do**
3: Calculate the distance between I^i and Pop' according to Eq. (3)
4: Calculate the goodness score $g(I^i)$ of I^i according to Eq. (4)
5: **end for**
6: Identify the worst solution I^w with the smallest goodness score in Pop': $I^w = arg\ min\{g(I^i) | i = 0, ..., p\}$
7: **if** I^w is not the offspring I^0 **then**
8: Replace I^w with I^0: $Pop = Pop \cup \{I^0\} \backslash \{I^w\}$
9: **end if**

Table 1
Settings of important parameters

Parameters	Section	Description	Values
p	2.2	size of population	10
α	2.3	tabu tenure management factor	15
cr	2.3	non-improvement TS iterations before perturbation	3,000
γ	2.3	perturbation strength	200
lr	2.3	number of TS iterations after recombination	10^6
β	2.6	goodness score coefficient	0.6 [26]

eter set to 0.6 according to [26], and $\widetilde{A}(.)$ represents the normalized function:

$$\widetilde{A}(y) = \frac{y - y_{min}}{y_{max} - y_{min} + 1} \tag{5}$$

where y_{min} and y_{max} are respectively the minimum and maximum of y in the population Pop. "+1" is used to avoid the possibility of a 0 denominator.

3 Experimental results

3.1 Test Instances

To evaluate the efficiency of the proposed memetic approach, extensive experiments were carried out on 71 well-known G-set benchmark graphs that are frequently used to assess max-bisection and max-cut algorithms. The G-set contains graphs ranging in size from 800 vertices to 20000 vertices [1]. These instances are generated by a machine-independent graph generator, comprising of toroidal, planar and random weighted graphs. Many authors including [38,5,9,23,28,31,35,36] employ these instances to test their algorithms for the max-bisection problem as well as the max-cut problem.

3.2 Parameter settings

The parameter settings used in our experiments are given in Table 1. These parameter values have been determined by performing a preliminary experiment on a selection of 11 (hard) instances (G_{32}, G_{33}, G_{35}, G_{40}, G_{41}, G_{42}, G_{55}, G_{57}, G_{62}, G_{65} and G_{66}). To determine the parameter values, we first began with the parameters required by the TS algorithm (α, cr, γ, see Section 2.3) and then determined the parameters used by the memetic algorithm (p, lr, β). For the TS algorithm we tested values for α in the range [10..30], cr in the range [2000..6000] and γ in the range [100..300]. For each value of

[1] Available at http://www.stanford.edu/∼yyye/yyye/Gset/

Table 2
Parameter tuning: α

Instance	TS with different α values									
	$\alpha = 10$		$\alpha = 15$		$\alpha = 20$		$\alpha = 25$		$\alpha = 30$	
	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
G_{32}	1396	1391.20	1400	1392.80	1400	1393.10	1400	1392.70	1398	1393.90
G_{33}	1374	1371.20	1376	1371.80	1374	1372.40	1376	1373.60	1374	1372.30
G_{35}	7669	7659.30	7673	7663.30	7673	7664.05	7668	7664.45	7673	7666.25
G_{40}	2383	2366.45	2388	2369.70	2392	2373.05	2389	2368.95	2386	2364.80
G_{41}	2390	2368.55	2396	2369.20	2388	2372.25	2393	2368.60	2384	2369.70
G_{42}	2474	2459.45	2476	2465.95	2472	2463.65	2477	2465.75	2474	2465.40
G_{55}	10257	10229.40	10263	10231.20	10265	10229.80	10263	10228.90	10259	10232.80
G_{57}	3458	3445.90	3460	3446.20	3456	3446.00	3456	3446.50	3454	3445.60
G_{62}	4812	4794.00	4812	4792.00	4810	4791.80	4810	4796.90	4810	4795.10
G_{65}	5504	5474.50	5504	5476.50	5498	5475.70	5498	5477.00	5492	5477.80
G_{66}	6276	6254.00	6282	6259.50	6284	6255.00	6276	6255.90	6276	6257.40

Table 3
Parameter tuning: cr

Instance	TS with different cr values									
	cr=2000		$cr = 3000$		$cr = 4000$		$cr = 5000$		$cr = 6000$	
	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
G_{32}	1396	1390.20	1400	1392.80	1400	1393.10	1400	1393.50	1402	1394.20
G_{33}	1376	1371.40	1376	1371.80	1376	1372.70	1376	1371.90	1376	1372.20
G_{35}	7672	7663.65	7673	7663.30	7671	7663.25	7673	7662.20	7668	7660.00
G_{40}	2394	2368.80	2388	2369.70	2388	2366.20	2385	2366.10	2385	2358.80
G_{41}	2387	2372.10	2396	2369.20	2395	2367.90	2393	2369.95	2395	2369.00
G_{42}	2471	2463.85	2476	2465.95	2473	2456.50	2468	2456.05	2478	2456.85
G_{55}	10265	10220.60	10263	10231.20	10263	10229.60	10261	10226.20	10257	10225.90
G_{57}	3458	3448.00	3460	3446.20	3454	3441.90	3462	3446.00	3452	3438.40
G_{62}	4812	4798.20	4812	4792.00	4808	4792.10	4808	4792.10	4802	4788.90
G_{65}	5500	5476.00	5504	5476.50	5494	5473.40	5492	5478.90	5484	5465.90
G_{66}	6280	6262.00	6282	6259.50	6272	6252.10	6286	6258.80	6262	6238.70

these parameters, 20 independent runs of TS were performed on each problem instance. Table 2 shows the results for different α values (with $cr = 3000$ and $\gamma = 200$) while Table 3 indicates the results for different cr values (with $\alpha = 15$ and $\gamma = 200$) and Table 4 shows the results for different γ values (with $\alpha = 15$ and $cr = 3000$). The tables show the best and averaged objective values found over the 20 runs. From these tables, we observe that the combination $(\alpha = 15, cr = 3000, \gamma = 200)$ globally leads to the best results. For the parameters related to the memetic algorithm, we fixed $p = 10, lr = 10^6$ in a similar way while $\beta = 0.6$ is chosen according to [26].

These parameter values were used to report our computational results, even

Table 4
Parameter tuning: γ

Instance	TS with different γ values									
	$\gamma = 100$		$\gamma = 150$		$\gamma = 200$		$\gamma = 250$		$\gamma = 300$	
	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
G_{32}	1402	1394.50	1402	1393.60	1400	1392.80	1398	1387.60	1392	1381.90
G_{33}	1376	1371.00	1378	1372.00	1376	1371.80	1374	1370.70	1372	1368.10
G_{35}	7671	7654.75	7672	7660.85	7673	7663.30	7670	7661.00	7672	7650.35
G_{40}	2382	2352.35	2386	2360.85	2388	2369.70	2391	2370.20	2392	2375.50
G_{41}	2382	2354.40	2401	2364.00	2396	2369.20	2394	2375.95	2399	2376.10
G_{42}	2466	2449.60	2475	2457.75	2476	2465.95	2470	2457.55	2473	2456.70
G_{55}	10235	10211.80	10249	10229.40	10263	10231.20	10265	10233.70	10267	10238.20
G_{57}	3448	3433.00	3452	3438.80	3460	3446.20	3458	3447.30	3458	3448.90
G_{62}	4796	4775.50	4806	4790.50	4812	4792.00	4810	4795.80	4812	4791.00
G_{65}	5482	5455.70	5486	5467.70	5504	5476.50	5498	5479.90	5498	5477.50
G_{66}	6258	6230.90	6266	6245.90	6282	6259.50	6284	6250.90	6286	6258.00

though improved results would be possible by further fine turning the parameters. As we see below, the adopted parameter settings are good and robust enough to allow the algorithm to attain very competitive results for the set of the tested instances compared with those reported in the literature.

3.3 Experimental protocol

Our MAMBP algorithm is programmed in C and compiled using GNU GCC on a PC running Windows XP with an Intel Xeon E5440 processor (2.83 GHz and 8G RAM).

Given the stochastic nature of MAMBP, each instance is independently solved 20 times with different random seeds. Each run is stopped when the processing time reaches its timeout limit. The timeout limit is set to be 30 minutes for graphs with $|V| < 5000$ while for graphs with $|V| \geq 5000$ a time limit of 120 minutes is allowed. Note that our timeout limit condition is comparable with the stop condition reported in [28,36].

Two comparisons are performed in order to assess the performance of the proposed memetic algorithm. The first comparison focuses on the max-bisection problem while the second one contrasts our (balanced) bisections with partitions obtained by several state of the art max-cut algorithms [5,9,28,31,35,36]. Notice that for the case of max-cut, the two subsets of a partition obtained by a max-cut algorithm are not required to be of equal cardinality. In this sense, a max-cut algorithm would have more opportunities to find partitions of larger weights. The best bisections obtained by our memetic algorithm are

available from: `http://www.info.univ-angers.fr/pub/hao/MAMBP.html`

3.4 Comparison with state-of-art max-bisection algorithm

In this section, we present the equal sized bisections obtained by our memetic approach for the max-bisection problem. To assess the relative quality of our results, we compare MAMBP with the new Lagrangian net algorithm (NLNA) which was published very recently [38]. NLNA was run on a Lenovo PC with an Intel Core E5300 processor (2.36 GHZ CPU and 1.96 GB RAM). According to the Standard Performance Evaluation Cooperation (www.spec.org), this computer is 1.2 time slower than the computer we used for our experiments. Note that in the literature there are at least two other heuristic bisection algorithms based on deterministic annealing [7] and variable neighborhood search [25]. Unfortunately, the results reported in these references cannot be reproduced since the graphs used are not available to the public domain. As a consequence, it is impossible to compare with their results.

Table 5 presents the detailed computational results of our MAMBP algorithm in comparison with those of the reference algorithm (NLNA) reported in [38]. The first two columns in the table indicate the name of and the number of vertices of the graph. Columns 3 to 7 show MAMBP's results including the best objective value (f_{best}), the success rate (hit) for reaching f_{best}, the averaged objective value ($f_{avg}(std)$) over the 20 runs with the standard deviation between parentheses for a hit value smaller than 100%, the average iterations ($Iter$) and the average CPU time in seconds ($time$) over the 20 runs on which the f_{best} value is reached. Columns 8 to 9 present NLNA's results. Given the speed ratio between our computer and the computer used for NLNA, we normalize the computing times of NLNA by dividing them with a factor of 1.2. The last column (Δ) indicates the differences in the best objective values between MAMBP and NLNA.

From Table 5, one observes that, for the 56 instances where the NLNA algorithm reports its results, our MAMBP algorithm dominates NLNA in terms of solution quality. For each of these instances, our MAMBP algorithm is able to attain a much larger objective value (f_{best}) compared with NLNA. Even our average results (f_{avg}) are better than the best results reported by NLNA. For the other 15 instances, no results are reported by NLNA ("-" in Table 5 indicates that a result for that particular instance was not available).

Finally, concerning the computing times, the times indicated are those that are needed for each algorithm to attain its f_{best} values. To verify that our MAMBP algorithm can easily attain the best objective values of NLNA, we rerun MAMBP and use NLNA's best objective values as our stop conditions.

Although we do not show the detailed results, this second experiment show that for the instances with less than 5000 vertices, MAMBP attains NLNA's best objective value in less than one second (against 3.17 to 58.18 seconds for NLNA). For larger graphs with at least 5000 vertices, MAMBP discovers NLNA's best objective value in no more than 10 seconds (against 260.91 to 512.62 seconds for NLNA). It is clear that MAMBP needs much less computing time than NLNA to discover solutions of comparable quality.

We can conclude that our MAMBP algorithm dominates the reference NLNA algorithm both in terms of solution quality and computing time.

Table 5: Comparative results for the max-bisection problem between MAMBP and NLNA [38] on the G-set benchmark instances. The symbol "-" means that the related statistics are not available.

Instance	$\|V\|$	$MAMBP$					$NLNA$[38]		Δ
		f_{best}	$f_{avg}(std)$	hit	$Iter$	$time$	f_{best}	$time$	
G_1	800	11624	11624	20/20	186225	2.40	11490	22.22	134
G_2	800	11617	11617	20/20	457273	5.20	11505	21.95	112
G_3	800	11621	11621	20/20	54756	1.32	11511	21.95	110
G_4	800	11646	11646	20/20	183623	1.77	11554	22.04	92
G_5	800	11631	11631	20/20	35178	0.88	11521	21.80	110
G_6	800	2177	2177	20/20	83819	1.16	2037	22.08	140
G_7	800	2002	2002	20/20	91160	0.82	1889	22.00	113
G_8	800	2004	2004	20/20	312230	4.26	1873	21.94	131
G_9	800	2052	2052	20/20	202926	1.19	1907	21.86	145
G_{10}	800	1998	1998	20/20	555555	5.59	1875	21.96	123
G_{11}	800	564	564	20/20	847410	12.10	560	3.18	4
G_{12}	800	556	556	20/20	810531	11.54	546	3.17	10
G_{13}	800	582	582	20/20	14481952	32.52	572	3.17	10
G_{14}	800	3062	3062	20/20	165369788	799.00	3023	7.02	39
G_{15}	800	3050	3050	20/20	125879080	692.96	2996	7.01	54
G_{16}	800	3052	3052	20/20	28611328	82.82	2994	7.02	58
G_{17}	800	3047	3047	20/20	257435761	778.67	2997	6.99	53
G_{18}	800	992	992	20/20	3210382	16.36	909	7.03	83
G_{19}	800	905	905	20/20	13295276	40.31	823	7.00	82
G_{20}	800	941	941	20/20	275771	2.48	865	6.98	76
G_{21}	800	930	930	20/20	6733404	34.71	849	6.98	81
G_{22}	2000	13359	13359	20/20	49429658	303.20	13105	57.48	254
G_{23}	2000	13344	13344	20/20	33121221	132.13	13120	57.36	224
G_{24}	2000	13336	13336	20/20	25855349	102.75	13115	57.34	221
G_{25}	2000	13340	13340	20/20	76664567	308.51	13125	57.41	215
G_{26}	2000	13328	13328	20/20	56585007	366.09	13160	57.25	168
G_{27}	2000	3341	3341	20/20	17155489	109.49	3109	57.16	232
G_{28}	2000	3298	3298	20/20	34382069	217.84	3063	58.13	235
G_{29}	2000	3403	3403	20/20	3121	1.36	3179	58.06	224
G_{30}	2000	3412	3412	20/20	11178762	44.82	3139	58.18	273
G_{31}	2000	3309	3309	20/20	41271756	263.21	3092	58.13	217
G_{32}	2000	1410	1410	20/20	139405200	887.50	1382	16.88	28
G_{33}	2000	1382	1382	20/20	132970476	856.80	1344	17.01	38
G_{34}	2000	1384	1384	20/20	82098799	536.12	1350	16.88	34
G_{35}	2000	7686	7684.70(1.93)	11/20	168054486	1312.42	7548	39.22	138
G_{36}	2000	7678	7676.20(2.29)	12/20	158749652	1259.10	7530	39.08	148
G_{37}	2000	7689	7687.80(1.81)	8/20	200196553	1543.36	7541	39.21	148
G_{38}	2000	7688	7688	20/20	132667831	922.66	7537	39.23	151
G_{39}	2000	2408	2408	20/20	212777429	976.95	2255	40.11	153
G_{40}	2000	2400	2398.20(5.41)	18/20	227544366	1198.28	2189	40.00	211
G_{41}	2000	2405	2405	20/20	111406876	546.57	2234	40.03	171

Table 5 – continued from previous page

Instance	$\|V\|$	$MAMBP$					$NLNA$[38]		Δ
		f_{best}	$f_{avg}(std)$	hit	$Iter$	$time(s)$	f_{best}	$time(s)$	
G_{42}	2000	2481	2477.50(6.19)	8/20	325913745	1513.96	2290	40.11	191
G_{43}	1000	6659	6659	20/20	135951	1.25	6580	15.34	79
G_{44}	1000	6650	6650	20/20	1797956	1.18	6548	15.33	102
G_{45}	1000	6654	6654	20/20	1089575	4.23	6513	15.33	141
G_{46}	1000	6649	6649	20/20	2886106	10.48	6538	15.33	111
G_{47}	1000	6657	6657	20/20	1650153	5.97	6529	15.34	128
G_{48}	3000	6000	6000	20/20	13128	1.42	-	-	-
G_{49}	3000	6000	6000	20/20	25179	1.28	-	-	-
G_{50}	3000	5880	5880	20/20	1509358	33.89	-	-	-
G_{51}	1000	3847	3847	20/20	90999846	292.60	3773	10.58	114
G_{52}	1000	3851	3851	20/20	245428916	814.96	3788	10.61	63
G_{53}	1000	3850	3850	20/20	133669193	516.28	3784	10.60	66
G_{54}	1000	3851	3851	20/20	150196875	551.51	3789	10.63	62
G_{55}	5000	10299	10291.50(8.36)	2/20	454657489	2396.84	-	-	-
G_{56}	5000	4016	4007.30(9.14)	3/20	354003885	1886.98	-	-	-
G_{57}	5000	3488	3475.60(12.98)	2/20	343206720	4883.34	-	-	-
G_{58}	5000	19276	19265.10(11.28)	1/20	466659576	4276.67	18931	268.71	345
G_{59}	5000	6085	6070.80(15.27)	1/20	476554320	4446.16	5578	260.91	507
G_{60}	7000	14186	14172.90(10.15)	1/20	484216617	5508.45	-	-	-
G_{61}	7000	5796	5782.30(15.82)	2/20	515858117	3755.71	-	-	-
G_{62}	7000	4866	4848.20(16.00)	1/20	399958710	4652.00	-	-	-
G_{63}	7000	26754	26725.20(24.91)	2/20	253009370	5670.30	-	-	-
G_{64}	7000	8731	8707.60(17.21)	1/20	495005836	5793.56	-	-	-
G_{65}	8000	5556	5538.60(17.45)	1/20	389509251	5385.86	5418	290.72	132
G_{66}	9000	6352	6332.50(21.18)	2/20	260212500	6267.15	6194	391.03	154
G_{67}	10000	6934	6923.60(9.97)	1/20	374288575	6203.44	6782	512.62	152
G_{70}	10000	9580	9565.00(17.36)	2/20	247649652	7032.70	-	-	-
G_{72}	10000	6990	6972.20(17.34)	2/20	373117581	7046.03	-	-	-
G_{77}	14000	9900	9881.10(17.41)	1/20	251667626	6752.26	-	-	-
G_{81}	20000	13978	13961.00(15.03)	1/20	76080159	7023.49	-	-	-

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

In this section, we further assess the performance of our MAMBP approach by comparing our results for MBP (i.e., equal sized partitions) with those obtained with different max-cut algorithms (i.e., not necessarily equal sized partitions). Recall that the max-bisection problem is constrained by the equal cardinality requirement for the two parts of its partitions while the max-cut problem accepts any two-part partitions. Consequently, a solution for the MBP is always a solution for the MCP, but the reverse is not true. In this section, we show that even with this equal cardinality constraint, our max-bisection algorithm MAMBP is able to find solutions that are better than the state of the art max-cut algorithms.

For this comparison, we are mainly interested in solution quality, i.e., the largest sum of the weights of the cutting edges of a partition. Due to the differences among the programming languages, data structures, compiler options and computers, we don't focus on computing time. We just mention that

the timeout limits we used are quite similar to those adopted by two recent reference algorithms [28,36].

The comparative results are summarized in Table 6 on the 69 G-set benchmark instances excluding the two largest graphs $G77$ and $G81$ for which no results are available for the reference algorithms. Columns 1 and 2 respectively give the instance name and the previous best solution value $f^*_{max\_cut}$ from references [1,5,9,23,28,31,35,36] (most of them are dedicated max-cut algorithms). Column 3 recalls the best results of our MAMBP approach extracted from Table 5. Columns 4–9 present the best results obtained by 6 reference algorithms: SS [28], TS-UBQP [23], DSA [1], VNSPR [9], CirCut [5] and GRASP-TS/PM [36]. The last three rows summarize the comparison between these algorithms and ours. The rows 'Better', 'Equal' and 'Worse' respectively denote the number of instances for which our MAMBP algorithm attains a result of better, equal and worse quality than each reference algorithm. We mark in bold those results that are the updated best known values obtained by the MAMBP approach.

From Table 6, we observe that our MAMBP algorithm is able to improve the current best known results in the literature for 31 instances while equaling the best known results for 25 other instances. This is a remarkable performance given that these best known results are the joint results of 2 or more max-cut algorithms. Indeed, when one compares our MAMBP algorithm with each of the 6 reference algorithms (SS, TS-UBQP, DSA, VNSPR, CirCut amd GRASP-TS/PM), we observe that MAMBP attains worse results for at most 10 instances while better results for at least 25 instances. These comparisons provide further evidence of the quality of the partitions achieved by our MAMBP algorithm.

On the other hand, on 13 instances, MAMBP's results are slightly inferior to the best known results. It is interesting to notice that most of these instances are of small size (with no more than 800 vertices). One possible explanation could be that for these graphs, the best max-cut solution is such that the two parts of the partition are of unequal size and thus not included in the search space of our MAMBP algorithm.

To sum, even if our MAMBP algorithm operates within a constrained search space composed solely of bisections (i.e., partitions of equal parts), its performance is remarkable for the max-cut problem when MAMBP is compared with dedicated max-cut algorithms.

Table 6: Comparative results of our MAMBP algorithm with the results of 6 state of the art max-cut algorithms on the G-set benchmark instances. Notice that max-cut algorithms lead to a partition whose two sets V_1 and V_2 may be of different sizes contrary to a partition of MAMBP whose two sets are always of equal size.

Instance	$f^*_{max\_cut}$	f_{MAMBP}	6 reference max-cut algorithms					
			SS[28]	TS-UBQP[23]	DSA[1]	VNSPR[9]	CirCut[5]	GRASP-TS/PM[36]
G_1	11624	11624	11624	11624	-	11621	11624	11624
G_2	11620	11617	11620	11620	-	11615	11617	11620
G_3	11622	11621	11622	11620	-	11622	11622	11620
G_4	11646	11646	11646	11646	-	11600	11641	11646
G_5	11631	11631	11631	11631	-	11598	11627	11631
G_6	2178	2177	2165	2178	-	2102	2178	2178
G_7	2006	2002	1982	2006	-	1906	2003	2006
G_8	2005	2004	1986	2005	-	1908	2003	2005
G_9	2054	2052	2040	2054	-	1998	2048	2054
G_{10}	2000	1998	1993	2000	-	1910	1994	2000
G_{11}	564	564	562	564	542	564	560	564
G_{12}	556	556	552	556	540	556	552	556
G_{13}	582	582	578	580	564	580	574	582
G_{14}	3064	3062	3060	3061	2982	3055	3058	3063
G_{15}	3050	3050	3049	3050	2975	3043	3049	3050
G_{16}	3052	3052	3045	3052	-	3043	3045	3052
G_{17}	3047	3047	3043	3046	-	3030	3037	3047
G_{18}	992	992	988	991	-	916	978	992
G_{19}	906	905	903	904	-	838	888	906
G_{20}	941	941	941	941	876	900	941	941
G_{21}	931	930	930	930	855	902	931	931
G_{22}	13359	13359	13346	13359	12989	13295	13346	13349
G_{23}	13342	**13344**	13317	13342	13006	13290	13317	13332
G_{24}	13337	13336	13303	13337	12985	13276	13314	13324
G_{25}	13332	**13340**	13320	13332	-	12298	13326	13326
G_{26}	13328	13328	13294	13328	-	12290	13314	13313
G_{27}	3336	**3341**	3318	3336	-	3296	3306	3325
G_{28}	3295	**3298**	3285	3295	-	3220	3260	3287
G_{29}	3394	**3403**	3389	3391	-	3303	3376	3394
G_{30}	3403	**3412**	3403	3403	3080	3320	3385	3402
G_{31}	3302	**3309**	3288	3288	2936	3302	3285	3299
G_{32}	1410	1410	1398	1406	1338	1396	1390	1406
G_{33}	1382	1382	1362	1378	1330	1376	1360	1374
G_{34}	1384	1384	1364	1378	1334	1372	1368	1376
G_{35}	7685	**7686**	7668	7678	-	7635	7670	7661
G_{36}	7677	**7678**	7660	7670	-	7632	7660	7660
G_{37}	7689	7689	7664	7682	-	7643	7666	7670
G_{38}	7683	**7688**	7681	7683	-	7602	7646	7670
G_{39}	2397	**2408**	2393	2397	-	2303	2395	2397
G_{40}	2392	**2400**	2374	2390	-	2302	2387	2392
G_{41}	2400	**2405**	2386	2400	-	2298	2398	2398
G_{42}	2474	**2481**	2457	2469	-	2390	2469	2474
G_{43}	6660	6659	6656	6660	-	6659	6656	6660
G_{44}	6650	6650	6648	6639	-	6642	6643	6649
G_{45}	6654	6654	6642	6652	-	6646	6652	6654
G_{46}	6649	6649	6634	6649	-	6630	6645	6649
G_{47}	6656	**6657**	6649	6656	-	6640	6656	6656
G_{48}	6000	6000	6000	6000	6000	6000	6000	6000
G_{49}	6000	6000	6000	6000	6000	6000	6000	6000
G_{50}	5880	5880	5880	5880	5880	5880	5880	5880
G_{51}	3847	3847	3846	3847	-	3808	3837	3847
G_{52}	3850	**3851**	3849	3849	-	3816	3833	3850
G_{53}	3848	**3850**	3846	3848	-	3802	3842	3848

Table 6 – continued from previous page

Instance	$f^*_{max\_cut}$	f_{MAMBP}	6 reference max cut algorithms					
			SS[28]	TS-UBQP[23]	DSA[1]	VNSPR[9]	CirCut[5]	GRASP-TS/PM[36]
G_{54}	3851	3851	3846	3851	-	3820	3842	3850
G_{55}	10236	**10299**	-	10236	9960	-	-	-
G_{56}	3934	**4016**	-	3934	3649	-	-	-
G_{57}	3460	**3488**	-	3460	3220	-	-	-
G_{58}	19248	**19276**	-	19248	-	-	-	-
G_{59}	6019	**6085**	-	6019	-	-	-	-
G_{60}	14057	**14186**	-	14057	13658	-	-	-
G_{61}	5680	**5796**	-	5680	5273	-	-	-
G_{62}	4822	**4866**	-	4822	4612	-	-	-
G_{63}	26963	26754	-	26963	8059	-	-	-
G_{64}	8610	**8731**	-	8610	7861	-	-	-
G_{65}	5518	**5556**	-	5518	-	-	-	-
G_{66}	6304	**6352**	-	6304	-	-	-	-
G_{67}	6894	**6934**	-	6894	-	-	-	-
G_{70}	9499	**9580**	-	9458	9499	-	-	-
G_{72}	6922	**6990**	-	6922	6644	-	-	-
Better			44	42	25	47	44	27
Equal			8	17	3	6	6	18
Worse			2	10	0	1	4	9

4 Analysis of MAMBP

4.1 Influence of crossover, pool replacement strategy and perturbation mechanism

In this section, we investigate the influence of some important ingredients of the proposed memetic algorithm: crossover (Section 2.5), population updating strategy (Section 2.6) and perturbation mechanism (Section 2.3.3). These studies are based on the same selection of the 11 hard problem instances used for setting the parameters (Section 3.2).

4.1.1 Influence of crossover: comparing MAMBP with TS

MAMBP uses TS as one of its main optimization procedures. It is therefore interesting to know if MAMBP, by integrating a crossover operator, is able to improve on the results of the TS algorithm alone. To verify this, we carry out additional experiments on the selection of the 11 G-set graphs and show a comparison between MAMBP and TS. For this experiment, we use the TS algorithm described in Algorithm 2 (see Section 2.3) with the same parameter settings given in Table 1. To make the comparison as fair as possible, we run TS with the same timeout limits as for MAMBP: 30 minutes for graphs with $|V| < 5000$ and 120 minutes if $|V| \geq 5000$ (see Section 3.3). In order not to penalize the tabu search algorithm which usually stops after lr iterations

Table 7
Comparative results of MAMBP and TS on 11 G-set instances for the max-bisection problem

Instance	$\|V\|$	MAMBP				TS			
		f_{best}	f_{avg}	hit	$time(s)$	f_{best}	f_{avg}	hit	$time(s)$
G_{32}	2000	1410	1410	20/20	887.50	1402	1400.80	4/20	1519.17
G_{33}	2000	1382	1382	20/20	856.80	1380	1378.60	6/20	802.67
G_{35}	2000	7686	7684.70	11/20	1312.42	7683	7680.35	1/20	1521.15
G_{40}	2000	2400	2398.20	18/20	1198.28	2400	2397.80	2/20	1285.30
G_{41}	2000	2405	2405	20/20	546.57	2405	2403.40	5/20	982.45
G_{42}	2000	2481	2479.50	8/20	1513.96	2481	2477.50	2/20	807.34
G_{55}	5000	10299	10291.50	2/20	2396.84	10291	10275.70	1/20	1971.73
G_{57}	5000	3488	3475.60	2/20	4883.34	3472	3466.20	1/20	2833.71
G_{62}	7000	4866	4848.20	1/20	4652.00	4830	4827.20	3/20	5089.40
G_{65}	8000	5556	5538.60	1/20	5385.86	5516	5511.00	2/20	4408.32
G_{66}	9000	6352	6332.50	2/20	6267.15	6298	6293.20	2/20	5959.70

($lr = 10^6$ in our case as indicated in Table 1), TS is restarted after $lr = 10^6$ iterations whenever the timeout limit is not reached. Like for MAMBP, we run the tabu search algorithm 20 times on each of the 11 selected G-set graphs.

Table 7 summarizes the computational results of the TS algorithm in comparison with those of MAMBP taken from Table 5. The following information for each instance is shown: the best objective value (f_{best}), the averaged objective value over the 20 runs (f_{avg}), the success rate (hit) for reaching its f_{best} value and the average CPU time ($time$) over the runs on which the f_{best} value is reached. From Table 7, we notice that for all the 11 selected instances, MAMBP performs far better than TS. Especially for each of the instances with at least 5000 vertices, MAMBP is able to reach a larger objective value f_{best} than TS. In addition, improvements are more pronounced for larger-size graphs. These comparative results provide clear evidence that the recombination operator plays an important role in the overall performance of our MAMBP algorithm and is more efficient when it comes to handling large graphs.

4.1.2 Influence of population updating strategy

In order to evaluate the distance-and-quality based population updating strategy (denoted by $DisQual$), we compare it with a popular updating strategy widely used in the literature (denoted by $PoolWorst$), which uses the new offspring solution to replace the worst solution (in terms of the fitness) of the population.

We have experimented both strategies within our memetic algorithm. As an illustration, Fig. 4 shows the detailed comparison on a large and difficult graph (G_{55}). Similar results have been observed on other instances. The stopping

criterion is the number of generations (i.e., the number of applications of the recombination operation) which is set to 300.

We keep other ingredients unchanged in the MAMBP algorithm and observe two evolution profiles of each population updating strategy: the best objective value (over 20 runs) vs. the number of generations (Fig. 4, left) and the population diversity vs. the number of generations (Fig. 4, right). The diversity of the population is defined as the average distance between each pair of solutions in the population (see Section 2.6) [12,27,32].

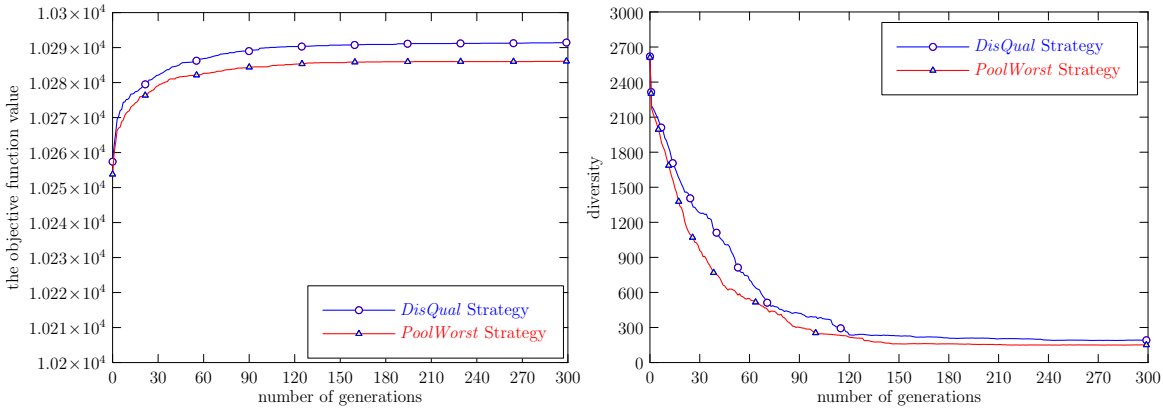

Fig. 4. Comparison between two population updating strategies

Fig. 4 (left) shows that the population converges more quickly toward high quality solutions with the distance-and-quality based population updating strategy than with the classical $PoolWorst$ strategy. In addition, the population diversity is better preserved during the evolution process with $DisQual$ than with $PoolWorst$, which is directly correlated to the evolution of the solution quality. We can conclude that the distance-and-quality updating strategy introduced in our approach allows the algorithm to better explore the search space and prevents the population from stagnating in poor local optima.

4.2 Influence of the perturbation mechanism

As indicated in Section 2.3.3, our tabu search procedure employs a perturbation mechanism which tries to bring controlled diversifications into the search. In order to be sure this perturbation mechanism makes a meaningful contribution, we report in Table 8 a comparison between the two versions of our tabu search algorithm with and without the perturbation mechanism (denoted by TS+PM and TS-PM respectively). We run both algorithms 20 times on the set of the 11 selected instances, each run being limited to $lr = 10^6$ iterations. Table 8 reports the results of this experiment. For each instance, the following information is provided: the best objective value (f_{best}), the averaged objective value over the 20 executions (f_{avg}), and the number of executions (hit)

Table 8
The influence of the perturbation mechanism on 11 G-set bi-section instances

Instance	$\|V\|$	TS-PM			TS+PM		
		f_{best}	f_{avg}	hit	f_{best}	f_{avg}	hit
G_{32}	2000	1398	1382.00	1/20	1400	1392.80	2/20
G_{33}	2000	1372	1355.20	1/20	1376	1371.80	4/20
G_{35}	2000	7665	7650.25	1/20	7673	7663.30	1/20
G_{40}	2000	2366	2343.85	2/20	2388	2369.70	1/20
G_{41}	2000	2379	2346.85	2/20	2396	2369.20	2/20
G_{42}	2000	2459	2430.40	1/20	2476	2465.95	1/20
G_{55}	5000	10004	9923.05	1/20	10263	10231.20	1/20
G_{57}	5000	3420	3399.60	1/20	3460	3446.20	1/20
G_{62}	7000	4778	4755.20	1/20	4812	4792.00	1/20
G_{65}	8000	5450	5423.80	1/20	5504	5476.50	2/20
G_{66}	9000	6226	6205.60	1/20	6282	6259.50	1/20

reaching its f_{best} value. Table 8 discloses that in most cases, our TS algorithm performs better with the perturbation mechanism (Column "TS+PM") than without it (Column "TS-PM"), confirming the usefulness of the perturbation mechanism within the TS algorithm.

4.3 Landscape analysis and motivation for the proposed crossover operator

In the last section, we demonstrated the importance of the proposed crossover operator to the overall performance of the memetic algorithm by comparing the performance of the proposed memetic algorithm with its underlying tabu search algorithm. In this section, we try to explain why our grouping crossover operator performs well and provide motivations for the proposed crossover operator. For this purpose, we analyze structural similarity between local optima of different quality in terms of the size of the commonly shared vertex groupings. Additionally, we investigate the correlation between quality (fitness) of local optima and their distances (according to Definition 1, Section 2.6) to the optimum by utilizing fitness landscape analysis techniques, which have been shown to be useful for understanding the behavior of an optimization algorithm.

To analyze structural similarity between local optima, we use the same set of the 11 selected G-set graphs as before. For each instance, we produce 8000 local optima of different quality using our memetic algorithm as well as its underlying tabu search with different time limits. Among these 8000 local optima, we select the top 10% (800) with the largest objective function values and call them 'high-quality solutions'. Moreover, since the global optimal solutions for the selected instances are unknown, we use instead the best solutions found by our memetic algorithm as our supposed global optima.

Table 9
Analysis of structural similarity between high-quality solutions for 11 hard max-bisection instances

Instance	$n_{distinct}$	ρ_{fdc}	S_{hq}	S_{lo}	D_{hq}	D_{lo}
G_{32}	200	-0.71	0.64	0.55	34.10	64.70
G_{33}	79	-0.69	0.63	0.57	49.60	67.70
G_{35}	167	-0.69	0.69	0.57	32.80	66.50
G_{40}	177	-0.31	0.73	0.59	44.60	81.40
G_{41}	613	-0.80	0.93	0.63	11.40	67.40
G_{42}	152	-0.22	0.70	0.59	60.70	82.50
G_{55}	5	-0.19	0.66	0.56	63.40	84.00
G_{57}	30	-0.61	0.60	0.55	53.20	67.60
G_{62}	26	-0.62	0.59	0.54	61.00	75.10
G_{65}	37	-0.63	0.58	0.54	60.70	75.40
G_{66}	44	-0.62	0.58	0.53	60.80	76.90

Table 9 contains the data related to the similarity between our 8000 local optima. Column $n_{distinct}$ indicates the total number of distinct global optima found for each instance. Columns S_{hq} and S_{lo} report respectively the average degree of similarity between the 800 high-quality solutions and the average degree of similarity between all the 8000 sampled local optima, the degree of similarity between two solutions I^a and I^b is expressed as $\frac{s(I^a,I^b)}{|V|}$ where $s(I^a, I^b)$ is a similarity function as described in Section 2.6. Columns D_{hq} and D_{lo} present respectively the average distance of high-quality solutions to their nearest global optimum and the average distance of all local optima to their nearest global optimum, expressed as a percentage of $\frac{|V|}{2}$ given that $\frac{|V|}{2}$ is the maximum distance between any two solutions.

From Table 9, we observe that in most cases, the degree of similarity between high-quality solutions is generally very large, which indicates high quality solutions share many groupings of vertices. We also notice that for all the cases, the degree of similarity between high-quality solutions is larger than that between other local optima. In addition, the values reported in 'D_{hq}' are generally smaller than the ones in 'D_{lo}', which indicates that the structure of the set of high quality solutions is very similar to the structure of the supposed global optimum. This suggests that if a significant number of vertices is grouped together throughout each of the high quality solutions, there is a strong chance that they are also grouped together in the global optimum. This observation constitutes a motivation and justification for our crossover which tries to preserve the largest common vertex groupings with respect to the two selected parent solutions.

In order to give further evidence that the G-set instances are well suited for our memetic algorithm, we carry out a landscape analysis and employ the fitness distance correlation (FDC) [19] to study the difficulty of the G-set instances for our memetic algorithm. FDC estimates how closely the fitness and distance to

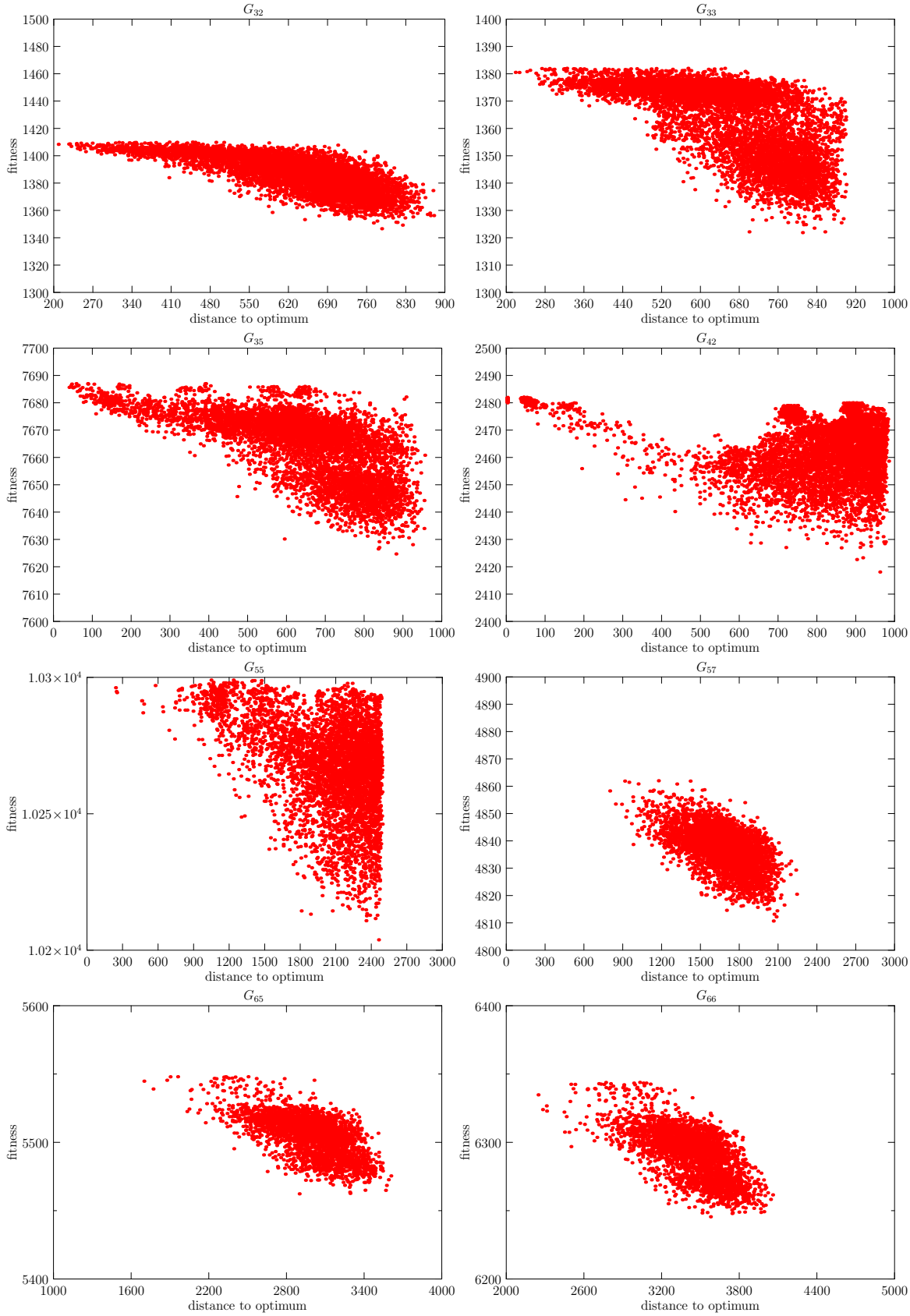

Fig. 5. Fitness-distance correlation plots with respect to the solution fitness and distance to the optimum for 8 graphs G32, G33, G35, G42, G55, G57, G65 and G66.

the nearest global optimum are related. If fitness improves when the distance to the optimum becomes smaller, then the search space is expected to be easy for the search algorithm, since there is a "path" to the optimum via solutions with increasing fitness. A value of the fitness distance correlation coefficient $\rho_{fdc} = -1$ for a maximization problem indicates that fitness and distance to the optimum are perfectly related and that search promises to be easy. A value of $\rho_{fdc} = 1$ means that the fitness function is completely misleading. FDC can also be visualized with the FD plot, where the same data used for estimating ρ_{fdc} is displayed graphically. For this study, we use the same solutions as above and the best solution as our supposed global optima. FD plots of 8 of the 11 instances are given in Fig. 5.

From these plots, one observes that G42 and G55 have the weakest FD correlations with a value of ρ_{fdc} equaling -0.261 and -0.193 respectively (Table 9). It is interesting to relate these values to the results of the memetic algorithm shown in Table 5. Indeed, for these graphs, MAMBP reaches its best solutions with a relatively low probability of 8/20 and 2/20 respectively. On the contrary, for the two graphs where the FC correlations are the highest (G42 and G41 with $\rho_{fdc} = -0.706$ and -0.799), MAMBP attains its best solutions easily with a 100% success rate. However, for the remaining graphs, the relation between the FDC and MAMBP's performance is more difficult to establish. This shows that a simple measure like FDC cannot be used to fully characterize the search difficulty of a problem instance.

5 Conclusion

In this paper, we have presented for the first time a memetic algorithm for the max-bisection problem (MAMBP). The proposed algorithm relies on a diversification-guided grouping crossover operator, a tabu search optimization procedure and a quality-and-diversity based population updating strategy.

The computational results obtained on the set of 71 well-known G-set graphs demonstrate that the proposed algorithm outperforms a recently published Lagrangian net algorithm for MBP by delivering improved solutions for each graph. Moreover, when considering the max-bisection problem as the "balanced" max-cut problem, the MAMBP algorithm remain highly competitive even compared to 6 state of the art max-cut algorithms. For 31 instances, MAMBP improves on the previous best known results reported in the literature.

Furthermore, we have carried out experiments to show the importance of the proposed crossover operator to the overall performance of the memetic algorithm. We have also shown an analysis of the similarity between high quality

solutions and of the search landscapes to get insights into the search space and provide motivations for the proposed grouping crossover operator. We have shown experimentally the usefulness of other important ingredients like the dynamic tabu tenure management, the perturbation mechanism and the pool updating strategy.

Acknowledgment

We are grateful to the anonymous referees for valuable suggestions and comments which have helped us to improve the paper. The work is partially supported by the RaDaPop (2009-2013) and LigeRo projects (2009-2013) from Pays de la Loire Region, France.

References

[1] J.E. Beasley, Heuristic algorithms for the unconstrained binary quadratic programming problem. Working Paper, The Management School, Imperial College, London, England, 1998.

[2] T. Bäck, D. B. Fogel, Z. Michalewicz (Eds), Handbook of evolutionary computation. Springer, 1997.

[3] U. Benlic, J.K. Hao, A multilevel memetic approach for improving graph k-partitions. IEEE Transactions on Evolutionary Computation, 15(5):624–472, 2011.

[4] U. Benlic, J.K. Hao, Hybrid metaheuristics for the graph partitioning problem. In EG Talbi (Ed.) Hybrid Metaheuristics. Studies in Computational Intelligence, Chapter 9, 2012.

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

[6] K.C. Chang, DH-C. Du, Efficient algorithms for layer assignment problems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 6(1):67-78, 1987.

[7] C. Dang, L. He, I. Hui, A deterministic annealing algorithm for approximating a solution of the max-bisection problem, Neural Networks, 15(3):441–458, 2002.

[8] E. Falkenauer, Genetic algorithms and grouping problems. New York: Wiley, 1998.

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

[10] A. Frieze, M. Jerrum. Improved approximation algorithm for max k-cut and max-bisection, Algorithmica, 18(1):67-81, 1997.

[11] P. Galinier, Z. Boujbel, M.C. Fernandes, An efficient memetic algorithm for the graph partitioning problem, Annals of Operations Research, 191(1):1–22, 2011.

[12] P. Galinier, J.K. Hao, Hybrid evolutionary algorithms for graph coloring, Journal of Combinatorial Optimization, 3(4):379–397, 1999.

[13] M. Garey, D. Johnson, Computers and Intractability. Freeman. N.Y., 1979.

[14] F. Glover, M. Laguna, Tabu Search, Kluwer Academic, Boston, 1997.

[15] M.X. Goemans, D.P. Williamson, Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, Journal of the Association for Computing Machinery, 42(6):1115-1145, 1995.

[16] E. Halperin, U. Zwick, A unified framework for obtaining improved approximation algorithms for maximum graph bisection problem, Random Struct. Algorithms, 20(3):382-402, 2002.

[17] J.K. Hao, Memetic algorithms in discrete optimization. In F. Neri, C. Cotta, P. Moscato (Eds.) Handbook of Memetic Algorithms. Studies in Computational Intelligence 379, Chapter 6, pages 73–94, 2012.

[18] B. Hendrickson, R. Leland, The Chaco User's Guide Version 2.0. Technical Report, Sandia National Laboratories, 1995.

[19] T. Jones, S. Forrest, Fitness distance correlation as a measure of problem difficulty for genetic algorithms. Proceedings of the 6th International Conference on Genetic Algorithms, Morgan Kaufmann, 184-192, 1995.

[20] S. Karish, F. Rendl, J. Clausen, Solving graph bisection problems with semidefinite programming, SIAM Journal on Computing 12(3):177-191, 2000.

[21] R.M. Karp, Reducibility among combinatorial problems, in R. E. Miller, J. W. Thacher (Ed.), Complexity of Computer Computation, Plenum Press, pages 85–103, 1972.

[22] G. Karypis, V. Kumar, Multilevel k-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed Computing, 48(1): 96–129, 1998.

[23] G. A. Kochenberger, J.K. Hao, Z. Lü, H. Wang, F. Glover, Solving large scale max cut problems via tabu search. Accepted to Journal of Heuristics to appear in 2012. DOI:10.1007/s10732-011-9189-8

[24] K. Krishnan, J. Mitchell, A semidefinite programming based polyhedral cut and price approach for the Max-Cut problem. Computational Optimization and Applications, 33(1): 51-71, 2006.

[25] A. Ling, C. Xu, L. Tang, A modified VNS metaheuristic for max-bisection problems, Journal of Computational and Applied Mathematics, 220(1–2):413–421, 2008.

[26] Z. Lü, F. Glover, J.K. Hao, A hybrid metaheuristic approach to solving the UBQP problem. European Journal of Operational Research, 207(3):1254–1262, 2010.

[27] Z. Lü, J.K. Hao, A memetic algorithm for graph coloring. European Journal of Operational Research, 203(1):241–250, 2010.

[28] R. Marti, A. Duarte, M. Laguna, Advanced scatter search for the max-cut problem. INFORMS Journal on Computing, 21(1):26-38, 2009.

[29] P. Moscato, C. Cotta, A Gentle Introduction to Memetic Algorithms. In F. Glover and G. A. Kochenberger (Eds.), Handbook of Metaheuristics. Kluwer, Norwell, Massachusetts, USA, 2003.

[30] F. Neri, C. Cotta, P. Moscato (Eds.) Handbook of Memetic Algorithms. Studies in Computational Intelligence 379, Springer, 2011.

[31] G. Palubeckis, Application of multistart tabu search to the MaxCut problem. Information Technology and Control, 2(31):29-35, 2004.

[32] D.C. Porumbel, J.K. Hao, P. Kuntz, An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. Computers and Operations Research, 37(10):1822–1832, 2010.

[33] D.C. Porumbel, J.K. Hao, P. Kuntz, An efficient algorithm for computing the distance between close partitions. Discrete Applied Mathematics, 159(1): 53–59, 2011.

[34] F. Rendl, G. Rinaldi, A. Wiegele, Solving Max-Cut to optimality by intersecting semidefinite and polyhedral relaxations, Mathematical Programming 121(2): 307-335 2008.

[35] V.P. Shylo, O.V. Shylo, Solving the maxcut problem by the global equilibrium search. Cybernetics and Systems Analysis, 46(5):744-754, 2010.

[36] Y. Wang, Z. Lü, F. Glover, J.K. Hao, Probabilistic GRASP-tabu search algorithms for the UBQP problem. Accepted to Computers and Operations Research DOI:10.1016/j.cor.2011.12.006

[37] C. Walshaw, M. Cross, Mesh partitioning: A multilevel balancing and refinement algorithm. SIAM Journal on Scientific Computing, 22(1): 63–80, 2000.

[38] F. Xu, X. Ma, B. Chen, A new Lagrangian net algorithm for solving max-bisection problems, Journal of Computational and Applied Mathematics, 235(13):3718–3723, 2011.

[39] Y. Ye, A 0.699-approximation algorithm for max-bisection, Mathematical Programming, 90(1):101-111, 2001.