A Memetic Approach for the Max-Cut Problem

Qinghua Wu and Jin-Kao Hao*

LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
{wu,hao}@info.univ-angers.fr

Abstract. The max-cut problem is to partition the vertices of a weighted graph $G = (V, E)$ into two subsets such that the weight sum of the edges crossing the two subsets is maximized. This paper presents a memetic max-cut algorithm (MACUT) that relies on a dedicated multi-parent crossover operator and a perturbation-based tabu search procedure. Experiments on 30 G-set benchmark instances show that MACUT competes favorably with 6 state-of-the-art max-cut algorithms, and for 10 instances improves on the best known results ever reported in the literature.

Keywords: Multi-parent crossover, memetic algorithm, local search, graph partitioning.

1 Introduction

Consider an undirected graph $G = (V, E)$ with vertex set $V = \{1, ..., n\}$ and edge set $E \subset V \times V$. Let $w_{ij} \in Z$ be the weight associated with edge $\{i, j\} \in E$. The well-known max-cut problem is to seek a partition of the vertex set V into two disjoint subsets $S_1 \subset V$ and $S_2 = V \setminus S_1$, such that the weight of the cut, defined as the sum of the weights on the edges connecting the two subsets, is maximized, i.e., $max \sum_{u \in S_1, v \in S_2} w_{uv}$. The max-cut problem, more precisely its weighted version, is one of Karp's 21 NP-complete problems [10].

The computational challenge of the max-cut problem has motivated a large number of solution procedures including approximation algorithms, exact methods and metaheuristics. The approximation approach (see for example [5,9,11]) provides a guaranteed performance, but is generally outperformed by other methods in computational testing. Recent examples on exact methods include the cut and price approach [14] and the branch and bound approach [18]. For large instances, various metaheuristic algorithms have been extensively used to find high-quality solutions in an acceptable time. Some representative examples include GRASP [4], ant colony [8], hybrid genetic algorithm [12], tabu search [16,13,21], scatter search [15], global equilibrium search [19] and maximum neural network [20].

In this paper, we present a memetic algorithm for the max-cut problem which is inspired by a very recent algorithm initially designed for the balanced max-bisection problem [22]. Experiments on a set of 30 well-known benchmark instances show that our memetic approach performs very well compared with state of the art algorithms.

* Corresponding author.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part II, LNCS 7492, pp. 297–306, 2012.

2 A Memetic Algorithm for the Max-Cut Problem

2.1 Outline of the Memetic Algorithm

Memetic algorithms are hybrid search methods that typically blend population-based search and neighborhood-based local search framework. The basic idea behind memetic approaches is to combine advantages of the crossover that discovers unexplored promising regions of the search space, and local optimization that finds good solutions by concentrating the search around these regions. The general architecture of our memetic algorithm for the max-cut problem is summarized in Algorithm 1. From an initial population of solutions which are first improved by a tabu search procedure, the algorithm carries out a number of evolution cycles. At each cycle, which is also called a generation, m ($m \geq 2$) parents are randomly selected to serve as parents and the crossover operator is applied to create an offspring solution, which is further optimized by tabu search. Subsequently, the population updating rule decides whether the improved offspring should be inserted into the population and which existing individual should be replaced. We describe below the main components of our memetic algorithm.

Algorithm 1. Memetic algorithm for the max-cut problem

Require: A weighted graph $G = (V, E, \omega)$, population size p
Ensure: The best solution I^* found
1: $Pop = \{I_1, ..., I_p\} \leftarrow GeneratePopulation(p)$ /* Section 2.3 */
2: $I^* \leftarrow Best(Pop)$
3: **while** Stop condition is not verified **do**
4: $(I^1, ..., I^m) \leftarrow ChooseParents(Pop)$ /*Randomly select $m \geq 2$ parents */
5: $I_0 = Recombination(I^1, ..., I^m)$ /* Section 2.5 */
6: $I_0 \leftarrow Tabu\_Search(I_0)$ /* Section 2.4 */
7: **if** $f(I_0) > f(I^*)$ **then**
8: $I^* \leftarrow I_0$ /* Update the best solution found so far */
9: **end if**
10: $Pop \leftarrow Pool\_Updating(I_0, Pop)$ /* Section 2.6 */
11: **end while**

2.2 Search Space and Fitness Function

Given a graph $G = (V, E)$ where each edge $\{i, j\} \in E$ is assigned a weight w_{ij}, the search space explored by our memetic algorithm is defined as the set of all the partitions of V into 2 disjoint subsets, i.e., $\Omega = \{\{S_1, S_2\} : S_1 \cap S_2 = \emptyset, S_1 \cup S_2 = V\}$. For a given partition or cut $I = \{S_1, S_2\}$, its fitness $f(I)$ is the weight of the cutting edges crossing S_1 and S_2, i.e.,

$$f(I) = \sum_{i \in S_1, j \in S_2} w_{ij} \tag{1}$$

2.3 Initial Population

The initial population of size p is constructed as follows. For each individual, we first assign randomly the vertices of the graph to the two vertex subsets S_1 and S_2 to produce a starting solution, and then apply the tabu search improvement procedure (see section 2.4) to obtain a local optimum. The resulting solution is added to the population if the solution does not duplicate any solution in the population. This procedure is repeated until $2 \times p$ solutions are obtained from which we retain the p best ones to form the initial population.

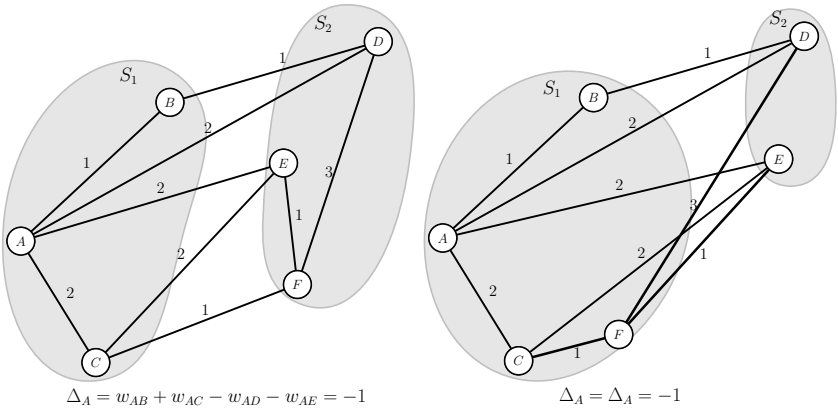
2.4 Perturbation-Based Tabu Search Improvement

To improve the newly generated offspring created by the crossover, we apply a perturbation-based tabu search procedure which integrates a periodic perturbation mechanism to bring diversification into the search. The general procedure of our tabu search method is described in Algorithm 2. Starting from an given solution, the tabu search procedure is first used to optimize the solution as far as possible until the best solution found so far cannot be improved within a certain number of iterations (lines 6–12), then the perturbation mechanism is applied to the current solution to generate a new starting solution (line 13–15), whereupon a new round of tabu search is launched. This process is repeated until a maximum allowed number ($MaxIter$) of iterations is reached.

Algorithm 2. Perturbation-based tabu search for the max-cut problem

Require: A weighted graph $G = (V, E, \omega)$, initial solution $I = \{S_1, S_2\}$, number $Piter$ of consecutive iterations eclipsed before triggering a perturbation, number $MaxIter$ of tabu search iterations
Ensure: The best solution I^* found and $f(I^*)$
1: $I^* \leftarrow I$ /* Records the best solution found so far */
2: $Iter \leftarrow 0$ /* Iteration counter */
3: Compute the move gain Δ_v according to Eq. 2 for each vertex $v \in V$.
4: Initiate the tabu list and tabu tenure
5: **while** $Iter < MaxIter$ **do**
6: Select an overall best allowed vertex $v \in V$ with the maximal move gain (ties are broken randomly)
7: Move v from its original subset to the opposite set
8: Update the tabu list and the move gain Δ_v for each $v \in V$
9: **if** $f(I) > f(I^*)$ **then**
10: $I^* \leftarrow I$ /* Update the best solution found so far */
11: **end if**
12: $Iter \leftarrow Iter + 1$
13: **if** I^* not improved after $Piter$ iterations **then**
14: $I \leftarrow Perturb(I)$ /* Apply perturbations to I */
15: **end if**
16: **end while**

Our tabu search procedure employs a neighborhood defined by the simple one-flip move, which consists of moving a vertex $v \in V$ from its original subset to the opposite set. Notice that this neighborhood is larger than the neighborhood used in [22] where the move operator displaces consecutively two vertices between the two subsets of the current solution to keep the partition balance.



$\Delta_A = w_{AB} + w_{AC} - w_{AD} - w_{AE} = -1$
$\Delta_B = w_{AB} - w_{BD} = 0$
$\Delta_C = w_{AC} - w_{CE} - w_{CF} = -1$
$\Delta_D = w_{DF} - w_{AD} - w_{BD} = 0$
$\Delta_E = w_{EF} - w_{AE} - w_{CE} = -3$
$\Delta_F = w_{DF} + w_{EF} - w_{CF} = 3$

$\Delta_A = \Delta_A = -1$
$\Delta_B = \Delta_B = 0$
$\Delta_C = \Delta_C + 2 \times w_{CF} = 1$
$\Delta_D = \Delta_D - 2 \times w_{DF} = -6$
$\Delta_E = \Delta_E - 2 \times w_{EF} = -5$
$\Delta_F = -\Delta_F = -3$

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

The concept of move gain is used to represent the change in the fitness function f (Eq. 1, Section 2.2). It expresses how much a cut could be improved if a vertex v is moved from its subset to the other subset. In our implementation, we employ a streamlined incremental technique for fast evaluation of move gains. Specifically, 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 following formula (see Figure 1 (left)).

$$\Delta_v = \begin{cases} \sum\limits_{x \in S_1, x \neq v} w_{vx} - \sum\limits_{y \in S_2} w_{vy}, & \text{if } v \in S_1 \\ \sum\limits_{y \in S_2, y \neq v} w_{vy} - \sum\limits_{x \in S_1} w_{vx}, & \text{otherwise.} \end{cases} \tag{2}$$

Each time one displaces a vertex v from its set to the other set, one just needs to update a subset of move gains affected by this move by applying the following abbreviated calculation (see Figure 1 (right)):

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

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

Then each iteration of our tabu search procedure selects a move with the largest Δ value (breaking ties randomly) whcih is not forbidden by the tabu list. Each time a vertex v is moved from its original subset to the opposite subset, v is forbidden to go back to its original set for a certain number tt of iterations (tt is called the tabu tenure). The tabu tenure is tuned dynamically according to the mechanism described in [6]. Finally, a simple aspiration criterion is applied which allows a move to be performed in spite of being tabu if it leads to a solution better than the current best solution.

When the best solution cannot be further improved by tabu search, a perturbation operator is triggered to vary the local optimum solution from which a new round of tabu search is launched. The perturbation consists in randomly moving γ vertices from their original subsets to the opposite subsets where γ is a parameter which indicates the strength of the perturbation.

2.5 The Multi-parent Crossover

It is commonly admitted that, in order to be efficient, a crossover operator should be adapted to the problem being solved and should integrate useful problem-specific knowledge of the given problem.

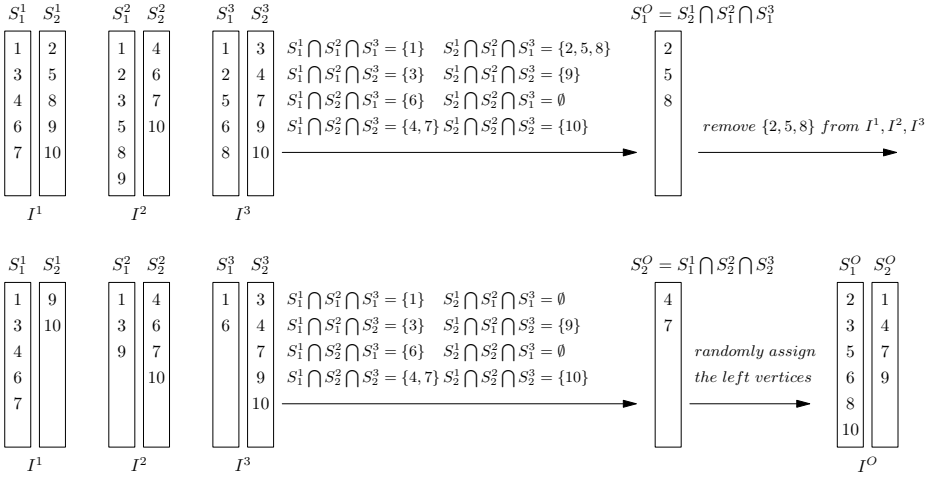


Fig. 2. An example of the multi-parent crossover operator

The max-cut problem is a grouping problem [3], i.e., a cut is composed of two distinct groups of vertices. An important principle in crossover design for grouping or partitioning problems is to manipulate promising 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 [7,17], bin packing [3] and graph partitioning [1,6].

We propose a grouping-based multi-parent crossover for the max-cut problem, the proposed crossover tries to preserve subsets (or grouping vertices) of the vertex partitions which are common to all parent individuals. More formally, given m chosen parents $\{I^1, ..., I^m\}$ ($m \geq 2$ is chosen randomly from a given range, fixed at {2,3,4} in this paper), each cut I^i can be represented as $I^i = \{S_1^i, S_2^i\}$. Then we produce an offspring solution $I^O = \{S_1^O, S_2^O\}$ using these m parent individuals as follows.

We first select one subset from each of the m parents such that the cardinality of intersection of these chosen subsets is maximal. Then we build S_1^O as the intersection of these m selected subsets, i.e., $S_1^O = arg\ max\{|S_{x_1}^1 \cap ... \cap S_{x_m}^m| : x_1, ..., x_m \in \{1, 2\}\}$. When S_1^O is built, for each $v \in S_1^O$, v is removed from all the parent individual subsets in which it occurs. Then, S_2^O is constructed in the same way as for building S_1^O such that $S_2^O = arg\ max\{|S_{x_1}^1 \cap ... \cap S_{x_m}^m| : x_1, ..., x_m \in \{1, 2\}\}$. If a vertex v is left unassigned after this procedure, v is placed either to S_1^O or S_2^O at random. Figure 2 shows an example with 3 parents.

Notice that this crossover differs from that of [22] for at least two reasons. It operates on multi-parents (instead of 2 parents) and its offspring is not required to be a balanced cut.

2.6 The Population Updating Rule

The updating procedure of *Pop* is invoked each time an offspring solution is created by the crossover operator and then improved by tabu search. Specifically, the improved solution I^O is added into *Pop* if I^O is distinct from any solution in *Pop* and the fitness $f(I^O)$ is higher (better) than the worst solution I^w in *Pop*. Under this circumstance, we update *Pop* by replacing I^w with I^O.

3 Computational Results

3.1 Experimental Protocol and Benchmark Instances

Our MACUT algorithm is coded in C and compiled using GNU GCC on a PC (Pentium 2.83GHz CPU and 8G RAM). We show our results on a selection of 30 well-known G-set benchmark graphs (see Table 1)[1] [4,13,15,16,19,21]. The first 24 instances (with at most 3000 variables) are the most popular and we include 6 additional larger instances with 5000 to 10000 variables. The edge weights of these graphs take values in the set {-1,0,1}.

The parameters of our algorithm are determined by a preliminary experiment on a selection of problem instances and are fixed as follows: population size $p = 10$, non-improvement tabu search iterations before perturbation $Piter = 500$, perturbation strength $\gamma = 150$, number of tabu search iterations applied to each offspring $MaxIter = 10^6$, number of parents for crossover $m \in \{2, 3, 4\}$. Given the stochastic nature of MACUT, each instance is independently solved 20 times, each run being limited to 30 minutes for graphs with $|V| < 5000$ and 120 minutes

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

for graphs with $|V| \geq 5000$. These timeout limits are comparable with the stop conditions used in [15,21].

3.2 Comparisons with the Best Known Results

Table 1 presents the detailed computational results of our MACUT algorithm as well as its underlying perturbation-based tabu search (PTS). The first two columns in the table indicate the name and the number of vertices of the graph. Column 3 presents the best-known objective value f_{pre} in the literature [4,13,15,16,19,21]. Columns 4 to 7 show MACUT's results including the best objective value (f_{best}), the averaged objective value (f_{avg}) over the 20 runs, the success rate (hit) for reaching f_{best} and the average CPU time in seconds ($time$) over the 20 runs for which the f_{best} value is reached. The last 4 columns present the results of its underlying perturbation-based tabu search.

Table 1. Computational results of MACUT and its underlying PTS on 30 G-set max-cut instances

Instance	$\|V\|$	f_{pre}	MACUT				PTS			
			f_{best}	f_{avg}	hit	$time(s)$	f_{best}	f_{avg}	hit	$time(s)$
G_1	800	11624	11624	11624	20/20	8.0	11624	11624	20/20	6.9
G_2	800	11620	11620	11620	20/20	6.3	11620	11620	20/20	7.5
G_3	800	11622	11622	11622	20/20	3.0	11622	11622	20/20	2.8
G_{11}	800	564	564	564	20/20	2.5	564	564	20/20	1.9
G_{12}	800	556	556	556	20/20	2.5	556	556	20/20	4.0
G_{13}	800	582	582	582	20/20	3.4	582	582	20/20	4.1
G_{14}	800	3064	3064	3063.95	19/20	450.0	3064	3063.9	18/20	661.2
G_{15}	800	3050	3050	3050	20/20	22.1	3050	3050	20/20	24.9
G_{16}	800	3052	3052	3052	20/20	11.6	3052	3052	20/20	10.7
G_{22}	2000	13359	13359	13359	20/20	74.8	13359	13359	20/20	206.2
G_{23}	2000	13342	**13344**	13344	20/20	280.0	13344	13343.2	12/20	651.7
G_{24}	2000	13337	13337	13337	20/20	252.2	13337	13335.6	20/20	815.6
G_{32}	2000	1410	1410	1410	20/20	349.2	1410	1408.3	3/20	844.6
G_{33}	2000	1382	1382	1382	20/20	391.4	1380	1379.6	18/20	667.6
G_{34}	2000	1384	1384	1384	20/20	220.7	1384	1381.6	2/20	512.4
G_{35}	2000	7685	**7686**	7685.9	18/20	895.7	7676	7674.2	2/20	1400.9
G_{36}	2000	7677	**7679**	7676.3	6/20	1395.4	7671	7669.3	1/20	1024.7
G_{37}	2000	7689	**7690**	7689.65	16/20	903.7	7678	7675.8	1/20	1175.3
G_{43}	1000	6660	6660	6660	20/20	3.6	6660	6660	20/20	3.7
G_{44}	1000	6650	6650	6650	20/20	3.7	6650	6650	20/20	3.0
G_{45}	1000	6654	6654	6654	20/20	18.2	6654	6654	20/20	20.1
G_{48}	3000	6000	6000	6000	20/20	0.2	6000	6000	20/20	0.2
G_{49}	3000	6000	6000	6000	20/20	0.4	6000	6000	20/20	0.4
G_{50}	3000	5880	5880	5880	20/20	15.0	5880	5880	20/20	13.6
G_{55}	5000	10236	**10299**	10290.8	2/20	2496.0	10235	10221	1/20	1807.3
G_{56}	5000	3934	**4016**	4006.9	2/20	2897.2	3954	3941.7	1/20	2108.9
G_{60}	7000	14057	**14186**	14171.1	1/20	5827.1	14065	14048.4	1/20	789.3
G_{65}	8000	5518	**5550**	5538.7	1/20	5879.6	5488	5479.1	1/20	1476.5
G_{66}	9000	6304	**6352**	6331.9	1/20	6203.8	6266	6255.2	1/20	2748.2
G_{67}	10000	6894	**6934**	6922.4	1/20	6761.3	6901	6892.1	1/20	1142.0

From Table 1, we observe that MACUT attains the best-known result for each of the 30 graphs. More importantly, MACUT improves on the best known results for 10 instances (indicated in bold). The average computing time required for MACUT to reach its best results f_{pre} varies from 3 seconds to 1.8 hours. It is clear that the required time to attain the current best-known objective value of column f_{pre} is shorter for the 10 graphs where MACUT finds improved solutions.

When comparing MACUT with its underlying PTS, one observes that MACUT outperforms PTS in terms of the best and average objective values. Indeed,

for 10 instances, MACUT is able to achieve better solution with much larger cut values. For 15 instances, MACUT reaches larger average objective values than PTS. In particular, it is remarkable that for each of the instances with at least 5000 vertices, MACUT performs far better than PTS. These comparative results demonstrate that the crossover operator is essential for the success of our MACUT algorithm and help MACUT to discover better solutions that are not attainable by our tabu search algorithm alone.

3.3 Comparisons with State-of-Art Max-Cut Algorithm

To further assess the performance of our MACUT approach, we now compare the results of our MACUT algorithm with the most effective heuristic algorithms in the literature. Due to the differences among the programming languages, data structures, compiler options and computers, we do not focus on computing time. Instead, we are mainly interested in solution quality for this experiment. We just mention that the timeout limits we used are quite similar to those adopted by some recent references like [15,21].

Table 2. Comparison with 6 state-of-the-art algorithms in terms of the best results obtained

Instance	f_{pre}	f_{best}	Best results of 6 reference max-cut algorithms					
			GES[19]	SS[15]	TS-UBQP[13]	VNSPR[4]	CirCut[2]	GRASP-TS/PM[21]
G_1	11624	11624	11624	11624	11624	11621	11624	11624
G_2	11620	11620	11620	11620	11620	11615	11617	11620
G_3	11622	11622	11622	11622	11620	11622	11622	11620
G_{11}	564	564	564	562	564	564	560	564
G_{12}	556	556	556	552	556	556	552	556
G_{13}	582	582	582	578	580	580	574	582
G_{14}	3064	3064	3064	3060	3061	3055	3058	3063
G_{15}	3050	3050	3050	3049	3050	3043	3049	3050
G_{16}	3052	3052	3052	3045	3052	3043	3045	3052
G_{22}	13359	13359	13359	13346	13359	13295	13346	13349
G_{23}	13342	13344	13342	13317	13342	13290	13317	13332
G_{24}	13337	13337	13337	13303	13337	13276	13314	13324
G_{32}	1410	1410	1410	1398	1406	1396	1390	1406
G_{33}	1382	1382	1382	1362	1378	1376	1360	1374
G_{34}	1384	1384	1384	1364	1378	1372	1368	1376
G_{35}	7685	7686	7685	7668	7678	7635	7670	7661
G_{36}	7677	7679	7677	7660	7660	7632	7660	7660
G_{37}	7689	7690	7689	7664	7664	7643	7666	7670
G_{43}	6660	6660	6660	6656	6660	6659	6656	6660
G_{44}	6650	6650	6650	6648	6639	6642	6643	6649
G_{45}	6654	6654	6654	6642	6652	6646	6652	6654
G_{48}	6000	6000	6000	6000	6000	6000	6000	6000
G_{49}	6000	6000	6000	6000	6000	6000	6000	6000
G_{50}	5800	5800	5880	5880	5880	5880	5880	5880
G_{55}	10236	10299	-	-	10236	-	-	-
G_{56}	3934	4016	-	-	3934	-	-	-
G_{60}	14057	14186	-	-	14057	-	-	-
G_{65}	5518	5550	-	-	5518	-	-	-
G_{66}	6304	6352	-	-	6304	-	-	-
G_{67}	6894	6934	-	-	6894	-	-	-
Better			4	18	18	18	19	12
Equal			20	6	12	6	5	12
Worse			0	0	0	0	0	0

Table 2 compares our MACUT algorithm with 6 state-of-the-art algorithms, which cover the best known results for the tested instances. Columns 2 and 3 recall the previous best known results (f_{pre}) and the best results found by MACUT (f_{best}). Columns 4 to 9 present the best results obtained by these reference algorithms. The last three rows show the summary of the comparison between our MACUT algorithm and these reference algorithms. The rows 'Better', 'Equal' and 'Worse' respectively denotes the number of instances for which our MACUT algorithm gets better, equal and worse results than the corresponding reference algorithm. From the last three rows of Table 2, it is observed that our MACUT algorithm outperforms the 6 reference algorithms in terms of the quality of the best solution found. In comparison with each of these 6 algorithm, MACUT achieves at least 4 better solutions and in no case, MACUT's result is worse than that of these reference algorithms. This experiment confirms thus the effectiveness of the proposed memetic approach to deliver high quality solutions for the tested 30 benchmark max-cut instances.

4 Conclusions

We presented an effective memetic algorithm for the NP-hard max-cut problem. The proposed MACUT algorithm integrates a grouping-based multi-parent crossover which tries to preserve groups of the vertex shared by the parent solutions and a dedicated perturbation-based tabu search procedure. The design of our crossover operator is motivated by an experimental observation (not shown in the paper due to the page limit) that groups of vertices are always shared by high quality solutions. Experimental results confirmed that the crossover operator boosts the performance of the algorithm and helps the search to discover high quality solutions unachievable by a local search algorithm alone. The experiments of MACUT on 30 well-known G-set benchmark instances demonstrated, by providing new best results for 10 instances, its competitiveness compared to 6 state-of-the-art algorithms. Additional studies are needed to better understand the proposed algorithm.

Acknowledgment. We are grateful to the referees for their comments and questions which helped us to improve the paper. The work is partially supported by the RaDaPop (2009-2013) and LigeRO (2010-2013) projects (Pays de la Loire Region, France).

References

1. Benlic, U., Hao, J.K.: A multilevel memetic approach for improving graph k-partitions. IEEE Transactions on Evolutionary Computation 15(5), 624–642 (2011)
2. Burer, S., Monteiro, R.D.C., Zhang, Y.: Rank-two relaxation heuristics for max-cut and other binary quadratic programs. SIAM Journal on Optimization 12, 503–521 (2001)
3. Falkenauer, E.: Genetic algorithms and grouping problems. Wiley, New York (1998)

4. Festa, P., Pardalos, P.M., Resende, M.G.C., Ribeiro, C.C.: Randomized heuristics for the max-cut problem. Optimization Methods and Software 7, 1033–1058 (2002)
5. Frieze, A., Jerrum, M.: Improved approximation algorithm for max k-cut and max-bisection. Algorithmica 18, 67–81 (1997)
6. Galinier, P., Boujbel, Z., Fernandes, M.C.: An efficient memetic algorithm for the graph partitioning problem. Annals of Operations Research 191(1), 1–22 (2011)
7. Galinier, P., Hao, J.K.: Hybrid evolutionary algorithms for graph coloring. Journal of Combinatorial Optimization 3(4), 379–397 (1999)
8. Gao, L., Zeng, Y., Dong, A.: An ant colony algorithm for solving Max-cut problem. Progress in Natural Science 18(9), 1173–1178 (2008)
9. Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the Association for Computing Machinery 42(6), 1115–1145 (1995)
10. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thacher, J.W. (eds.) Complexity of Computer Computation, pp. 85–103. Plenum Press (1972)
11. Karish, S., Rendl, F., Clausen, J.: Solving graph bisection problems with semidefinite programming. SIAM Journal on Computing 12, 177–191 (2000)
12. Kim, S.H., Kim, Y.H., Moon, B.Y.: A Hybrid Genetic Algorithm for the MAX CUT Problem. In: Genetic and Evolutionary Computation Conference, pp. 416–423 (2001)
13. Kochenberger, G., Hao, J.K., Lü, Z., Wang, H., Glover, F.: Solving large scale max cut problems via tabu search. Accepted to Journal of Heuristics (2012)
14. Krishnan, K., Mitchell, J.: A semidefinite programming based polyhedral cut and price approach for the Max-Cut problem. Computational Optimization and Applications 33, 51–71 (2006)
15. Marti, R., Duarte, A., Laguna, M.: Advanced scatter search for the max-cut problem. INFORMS Journal on Computing 21(1), 26–38 (2009)
16. Palubeckis, G.: Application of multistart tabu search to the MaxCut problem. Information Technology and Control 2(31), 29–35 (2004)
17. Porumbel, D.C., Hao, J.K., Kuntz, P.: An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. Computers and Operations Research 37(10), 1822–1832 (2010)
18. Rendl, F., Rinaldi, G., Wiegele, A.: Solving Max-Cut to optimality by intersecting semidefinite and polyhedral relaxations. Mathematical Programming 121, 307–335 (2008)
19. Shylo, V.P., Shylo, O.V.: Solving the maxcut problem by the global equilibrium search. Cybernetics and Systems Analysis 46(5), 744–754 (2010)
20. Wang, J.: An Improved Maximum Neural Network Algorithm for Maximum Cut Problem. Neural Information Processing 10(2), 27–34 (2006)
21. Wang, Y., Lü, Z., Glover, F., Hao, J.K.: Probabilistic GRASP-tabu search algorithms for the UBQP problem. Accepted to Computers and Operations Research (2012), http://dx.doi.org/10.1016/j.cor.2011.12.006
22. Wu, Q., Hao, J.K.: Memetic search for the max-bisection problem. Accepted to Computers and Operations Research (2012), http://dx.doi.org/10.1016/j.cor.2012.06.001