Breakout Local Search for the Max-Cut Problem

Una Benlic and Jin-Kao Hao [*]

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

Accepted to *Engineering Applications of Artificial Intelligence*, Sept. 3, 2012

Abstract

Given an undirected graph $G = (V, E)$ where each edge of E is weighted with an integer number, the maximum cut problem (Max-Cut) is to partition the vertices of V into two disjoint subsets so as to maximize the total weight of the edges between the two subsets. As one of Karp's 21 NP-complete problems, Max-Cut has attracted considerable attention over the last decades. In this paper, we present Breakout Local Search (BLS) for Max-Cut. BLS explores the search space by a joint use of local search and adaptive perturbation strategies. The proposed algorithm shows excellent performance on the set of well-known maximum cut benchmark instances in terms of both solution quality and computational time. Out of the 71 benchmark instances, BLS is capable of finding new improved results in 33 cases and attaining the previous best-known result for 35 instances, within a computational time ranging from less than one second to 5.6 hours for the largest instance with 20000 vertices.

Keywords: Max-cut; local search and heuristics; adaptive diversification; meta-heuristics.

1 Introduction

The maximum cut problem (Max-Cut) is one of Karp's 21 NP-complete problems with numerous practical applications [12]. Let $G = (V, E)$ be an undirected graph with the set of vertices V and the set of edges E, each edge

[*] Corresponding author.
Email addresses: `benlic@info.univ-angers.fr` (Una Benlic), `hao@info.univ-angers.fr` (Jin-Kao Hao).

$(i, j) \in E$ being associated with a weight w_{ij}. Max-Cut consists in partitioning the vertices of V into two disjoint subsets V_1 and V_2 such that the total weight of the edges whose endpoints belong to different subsets is maximized, i.e.,

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

Given its theoretical and practical importance, Max-Cut has received considerable attention over the last decades. Well-known exact methods for Max-Cut, such as the Branch and Price procedure [17], are capable of solving to optimality medium size instances (i.e., $|V| = 500$). For larger instances, different heuristic methods have been proposed including global equilibrium search [21], projected gradient approach [6], rank-2 relaxation heuristic [7], and greedy heuristics [11]. Other well-known algorithms are based on popular metaheuristics such as variable neighbourhood search [8], tabu search [1,16], scatter search [19], grasp [22], and different hybrid approaches [8,22].

In this work, we propose a new heuristic algorithm for Max-Cut, using the Breakout Local Search (BLS) [4,5]. Based on the framework of Iterated Local Search (ILS) [18], BLS combines local search (i.e., the steepest descent) with a dedicated and adaptive diversification mechanism. Its basic idea is to use local search to discover local optima and employ adaptive perturbations to continually move from one attractor to another in the search space. The continual exploration of new search areas is achieved by alternating between random and directed, and weak and strong perturbations depending on the current search state. Despite its simplicity, BLS shows excellent performance on the set of well-known Max-Cut instances in terms of both solution quality and computational time. Out of 71 benchmark instances, the proposed approach is capable of improving the previous best-known solutions in 33 cases and reaching the previous best-known results for 35 instances, within a computational time ranging from less than one seconds to 5.6 hours for the largest instance with 20000 vertices.

In the next section, we present in details the breakout local search approach for the Max-Cut problem. Section 3 shows extensive computational results and comparisons with the state-of-art Max-Cut algorithms. In Section 4, we provide a parameter sensitivity analysis and justify the parameter settings used to obtain the reported results. Moreover, we investigate the efficiency of the proposed diversification mechanism of BLS, and highlight the importance of excluding diversification schemes during the local search phase. Conclusions are given in the last section.

2 Breakout Local Search (BLS)

Our Breakout Local Search (BLS) approach is conceptually rather simple and transits from one basin of attraction to another basin by a combined use of local search (to reach local optima) and dedicated perturbations (to discover new promising regions).

Recall that given a search space S and an objective function f, a neighborhood N is a function $N : S \rightarrow \mathcal{P}(S)$ that associates to each solution C of S a subset $N(C)$ of S. A local optimum C^* with respect to the given neighborhood N is a solution such that $\forall C' \in N(C^*)$, $f(C^*) \geq f(C')$, where f is the maximization function. A basin of attraction of a local optimum C^* can be defined as the set B_{C^*} of solutions that lead the local search to the given local optimum C^*, i.e., $B_{C^*} = \{C \in S | LocalSearch(C) = C^*\}$ [3]. Since a local optimum C^* acts as an attractor with respect to the solutions B_{C^*}, the terms attractor and local optimum will be used interchangeably throughout this paper. Notice that in practice, for a given solution C, a neighboring solution C' is typically generated by applying a move operator to C. Let m be the move applied to C, we use $C' \leftarrow C \oplus m$ to denote the transition from the current solution C to the new neighboring solution C'.

BLS follows the general scheme of iterated local search [18] and alternates between a local search phase, which uses the steepest descent [20] to discover an attractor, and a perturbation phase, which guides the search to escape from the current basin of attraction. The general BLS algorithm is shown in Algorithm 1.

After generation of an initial solution (line 1) and an initialization step (lines 2-7), BLS applies the steepest descent to reach a local optimum (lines 9-16). Each iteration of this local search procedure identifies the best move m among those that are applicable to the current solution C, and applies m to C to obtain a new solution which replaces the current solution (lines 9-12). Updates are performed to reflect the current state of the search (lines 13-22). In particular, if the last discovered local optimum is better than the best solution found so far (recorded in C_{best}), C_{best} is updated with the last local optimum (lines 17-18).

If no improving neighbor exists, local optimality is reached. At this point, BLS tries to escape from the basin of attraction of the current local optimum and to go into another basin of attraction. For this purpose, BLS applies a number L of dedicated moves to the current optimum C (we say that C is perturbed, see Section 2.3 for details). Each time an attractor is perturbed (line 37), the perturbed solution becomes the new starting point for the next round of the local search procedure (a new round of the outer *while* structure, line 8).

The algorithm stops when a prefixed condition is satisfied. This can be, for example, a cutoff time, an allowed maximum of iterations or a target objective value to be attained. In this paper, an allowed maximum of iterations is used (see Section 3).

To determine the most appropriate perturbation (its type and strength), we distinguish two situations. First, if the search returns to the immediate previous attractor (recorded in C_p), BLS perturbs C more strongly by increasing the number of perturbation moves L to be applied (lines 28-30). Otherwise (i.e., the search succeeded in escaping from the current attractor), the number of perturbation moves L is reduced to its initial value L_0 (L_0 is a parameter). Second, if the search cannot improve the best solution found so far after visiting a certain number T (T is a parameter) of local optima, BLS applies a significantly stronger perturbation in order to drive definitively the search towards a new and more distant region in the search space (lines 24-27).

Algorithm 1 The Breakout Local Search for the Max-cut Problem

```
Require: Graph $G = (V, E)$, initial jump magnitude $L_0$, max. number $T$ of non-improving attractors
    visited before strong perturb.
Ensure: A set of vertices forming a clique.
1: $C \leftarrow generate\_initial\_solution(V)$   /* $C$ is a partition of $V$ into two subsets $V_1$ and $V_2$ */
2: $f_c \leftarrow f(C)$   /* $f_c$ Records the objective value of the solution */
3: $C_{best} \leftarrow C$   /* $C_{best}$ Records the best solution found so far */
4: $f_{best} \leftarrow f_c$   /* $f_{best}$ Records the best objective value reached so far */
5: $C_p \leftarrow C$   /* $C_p$ Records the solution obtained after the last descent */
6: $\omega \leftarrow 0$   /* Counter for consecutive non-improving local optima */
7: $Iter \leftarrow 0$   /* Counter of iterations */
8: while stopping condition not reached do
9:     Let $m$ be the best move $m$ eligible for $C$   /* See Section 2.1 */
10:    while $f(C \oplus m) > f(c)$ do
11:        $f_c \leftarrow f(C \oplus m)$   /* Records the objective value of the current solution */
12:        $C \leftarrow C \oplus m$   /* Perform the best-improving move */
13:        Update bucket sorting structure   /* Section 2.2 */
14:        $H \leftarrow Iter + \gamma$   /* Update tabu list, $\gamma$ is the tabu tenure */
15:        $Iter \leftarrow Iter + 1$
16:    end while
17:    if $f_c > f_{best}$ then
18:        $C_{best} \leftarrow C$; $f_{best} \leftarrow f_c$   /* Update the recorded best solution */
19:        $\omega \leftarrow 0$ /* Counter for consecutive non-improving local optima reset */
20:    else
21:        $\omega \leftarrow \omega + 1$
22:    end if
23:    /* Determine the number of perturbation moves $L$ to be applied to $C$ */
24:    if $\omega > T$ then
25:        /* Search seems to be stagnating, random perturbation required */
26:        $\omega \leftarrow 0$
27:    end if
28:    if $C = C_p$ then
29:        /* Search returned to previous local optimum, increase number of perturbation moves*/
30:        $L \leftarrow L + 1$
31:    else
32:        /* Search escaped from the previous local optimum, reinitialize number of perturb. moves */
33:        $L \leftarrow L_0$
34:    end if
35:    /* Perturb the current local optimum $C$ with $L$ perturbation moves */
36:    $C_p \leftarrow C$
37:    $C \leftarrow Perturbation(C, L, H, Iter, \omega)$   /* Section 2.3.1 */
38: end while
```

The success of the described method depends basically on two key factors.

First, it is important to determine the number L of perturbation moves (also called "perturbation strength" or "jump magnitude") to be applied to change or perturb the solution. Second, it is equally important to consider the type of perturbation moves to be applied. While conventional perturbations are often based on random moves, more focused perturbations using dedicated information could be more effective. The degree of diversification introduced by a perturbation mechanism depends both on the jump magnitude and the type of moves used for perturbation. If the diversification is too weak, the local search has greater chances to end up cycling between two or more locally optimal solutions, leading to search stagnation. On the other hand, a too strong diversification will have the same effect as a random restart, which usually results in a low probability of finding better solutions in the following local search phase. For its perturbation mechanism, the proposed BLS takes advantage of the information related to the search status and history. We explain the perturbation mechanism is Section 2.3.

2.1 The neighbourhood relations and its exploration

For solution transformations, BLS employs three distinct move operators (moves for short) $M_1 - M_3$ whose basic idea is to generate a new cut C by moving vertices to the opposite partition subset. To define these move operators, we first introduce the notion of move gain which indicates how much a partition is improved, according to the optimization objective, if a vertex is moved to another subset. For each vertex $v \in V$, we determine the gain g_v for moving v to the opposite partition subset. As we show in Section 2.2, the vertex with the (best) highest gain can be easily determined using a special bucket data structure that has been extensively used to the related (and different) graph partitioning problem.

Given a partition (cut) $C = \{V_1, V_2\}$, the three move operators are defined below:

M_1: Select a vertex v_m with the highest gain. Move the selected vertex v_m from its current subset to the opposite partition subset.
M_2: Select a highest gain vertex v_1 from V_1 and a highest gain vertex v_2 from V_2. Move v_1 to V_2, and v_2 to V_1.
M_3: Randomly select a vertex v. Move the selected vertex v from its current subset to the opposite partition subset.

Each iteration of the local search consists in identifying the best move m from M_1 and applying it to C to obtain a new solution. This process is repeated until a local optimum is reached (see lines 8–16 of Alg. 1). The two directed perturbations of BLS apply a move m from M_1 and M_2 respectively, while the strong perturbation, which acts as a restart, performs a move from M_3 (see Section 2.3.1 for the three perturbation strategies).

2.2 Bucket sorting

To ensure a fast evaluation of the neighbouring moves, our implementation uses the bucket sorting data structure which keeps vertices ordered by their gains. This structure is used to avoid unnecessary search for the highest gain vertex and to minimize the time needed for updating the gains of vertices affected by each move.

The bucket sorting structure was first proposed by Fiduccia and Mattheyses [9] to improve the Kerninghan-Lin algorithm [14] for the minimum graph bisection problem. We adopt this technique for our Max-Cut problem. The idea is to put all the vertices with the same gain g in a bucket that is ranked g. Then, to determine a vertex with the maximum gain, it suffices to go to the non-empty bucket with the highest rank, and select a vertex from the bucket. After each move, the bucket structure is updated by recomputing gains of the selected vertex and its neighbors, and transferring these vertices to appropriate buckets.

The bucket data structure consists of two arrays of buckets, one for each partition subset, where each bucket of an array is represented by a doubly linked list. An example of the bucket data structure is illustrated in Figure 1. The arrays are indexed by the possible gain values for a move, ranging from g_{max} to g_{min}. A special pointer *maxgain* points to the highest index in the array whose bucket is not empty, and thus enables to select the best improving move in constant time. The structure also keeps an additional array of vertices where each element (vertex) points to its corresponding vertex in the doubly linked lists. This enables a direct access to the vertices in buckets and their transfer from one bucket to another in constant time.

Each time a move involving a vertex v is performed, only the gains of the vertices adjacent to v are recalculated (in $O(1)$) and updated in the bucket structure in constant time (delete and insert operations in the bucket are both of $O(1)$ complexity). Therefore, the complexity of moving vertex v from its current subset to the opposite subset is upper-bounded by the number of vertices adjacent to v.

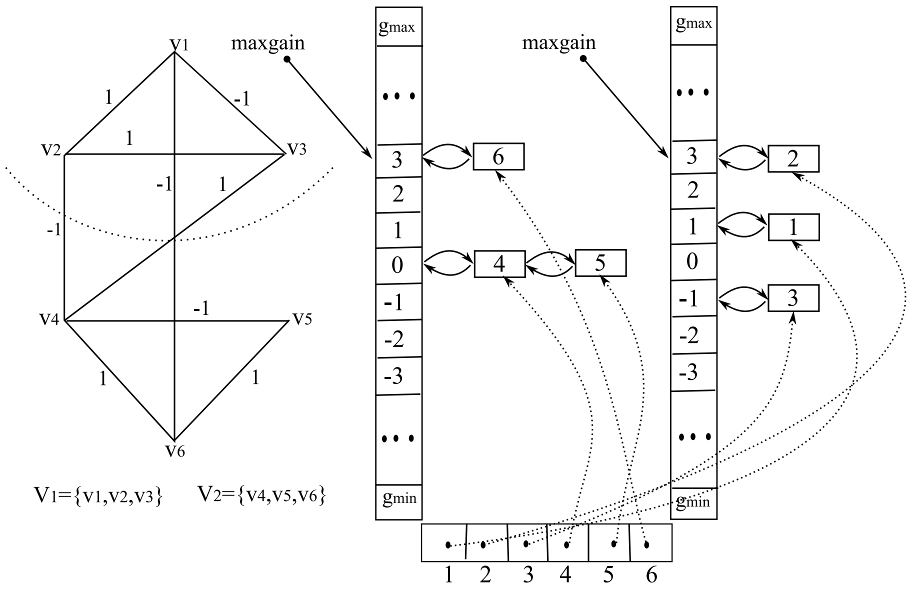

Fig. 1. An illustrative example of the bucket sorting data structure on a graph with 6 vertices

2.3 Adaptive perturbation mechanism

The perturbation mechanism plays a crucial role within BLS since the local search alone cannot escape from a local optimum. BLS thus tries to move to the next basin of attraction by applying a weak or strong, directed or random perturbation depending on the state of the search (lines 23–37 of Alg. 1). The pseudo-code of this adaptive perturbation-based diversification procedure is given in Algorithms 2 and 3.

The perturbation procedure (Alg. 2) takes as its input the following parameters: the current solution C which will be perturbed, the jump magnitude L determined in the main BLS algorithm (Alg. 1, lines 30 and 33), the tabu list H, the global iteration counter $Iter$ and the number of consecutive non-improving local optima visited ω. Based on these information, the perturbation procedure determines the type of moves to be applied. The perturbation moves can either be random or directed. First, if the search fails to update the best solution after consecutively visiting a certain number T of local optima (indicated by $\omega = 0$, Alg. 2), the search is considered to be trapped in a non-promising search-space region and a strong perturbation is applied (Alg. 2, line 3) which basically displaces randomly a certain number (fixed by the jump magnitude L) of vertices from one side of the two-way partition to the other side. Here no constrained is imposed on the choice of the displaced vertices and any vertex can take part in this perturbation process. We will explain this random perturbation in Section 2.3.1.

Second, if the number of consecutively visited local optima does not exceed the threshold T, we will allow the search to explore the current search region more thoroughly by adaptively choosing between weaker (directed) and random (stronger) perturbation moves (the adaptive mechanism is described in Section 2.3.1). Basically, the directed perturbation is more oriented towards search intensification than a random perturbation, since perturbation moves are chosen by also considering the quality criterion so as not to deteriorate too much the current solution. Two different types of directed perturbation moves are distinguished with BLS and are explained in the next section.

Once the type of perturbation is determined, BLS modifies the current solution C by applying to it L perturbation moves which are chosen from the corresponding set of moves defined in the next section (Alg. 3). Notice that as in the case of moves performed during the local search phase, perturbation moves are added into the tabu list to avoid reconsidering them for the next γ iterations (Alg. 3, line 4, see also Section 2.3.1). The perturbed solution is then used as the new starting point for the next round of the local search.

Algorithm 2 The perturbation procedure $Perturbation(C, L, H, Iter, \omega)$

```
Require: Local optimum $C$, jump magnitude $L$, tabu list $H$, global iteration counter $Iter$, number of
    consecutive non-improving local optima visited $\omega$.
Ensure: A perturbed solution $C$.
1: if $\omega = 0$ then
2:     /* Best sol. not improved after a certain num. of visited local opt.*/
3:     $C \leftarrow Perturb(C, L, B)$   /* Random perturb. with moves from set $B$, see Section 2.3.1 for the
       definition of set $B$ */
4: else
5:     Determine probability $P$ according to Formula (2) /* Section 2.3.1 */
6:     With probability $P * Q$, $C \leftarrow Perburb(C, L, A_1)$
       /* Directed perturb. with moves of set $A_1$, see Section 2.3.1 for the definition of set $A_1$ */
7:     With probability $P(1 - Q)$, $C \leftarrow Perturb(C, L, A_2)$
       /* Directed perturb. with moves of set $A_2$, see Section 2.3.1 for the definition of set $A_2$ */
8:     With probability $(1 - P)$, $C \leftarrow Perturb(C, L, B)$
       /* Random perturb. with moves of set $B$ */
9: end if
10: Return C
```

Algorithm 3 Perturbation operator $Perburb(C, L, M)$

```
Require: Local optimum $C$, perturbation strength $L$, tabu list $H$, global iteration counter $Iter$, the set of
    perturbation moves $M$.
Ensure: A perturbed solution $\pi$.
1: for $i := 1$ to $L$ do
2:     Take move $m \in M$
3:     $C \leftarrow C \oplus m$
4:     $H \leftarrow Iter + \gamma$     /* Update tabu list, $\gamma$ is the tabu tenure */
5:     Update bucket sorting structure     /* Section 2.2 */
6:     $Iter \leftarrow Iter + 1$
7: end for
8: Return C
```

2.3.1 The perturbation strategies

As mentioned above, BLS employs two types of directed perturbations and a random perturbation to guide the search towards new regions of the search space.

Directed perturbations are based on the idea of tabu list from tabu search [10]. These perturbations use a selection rule that favors the moves that minimize the degradation of the objective, under the constraint that the moves are not prohibited by the tabu list. Move prohibition is determined in the following way. Each time a vertex v is moved from its current subset V_c, it is forbidden to place it back to V_c for γ iterations (called tabu tenure), where γ takes a random value from a given range.

The information for move prohibition is maintained in the tabu list H where the i^{th} element is the iteration number when vertex i was last moved plus γ. The tabu status of a move is neglected only if the move leads to a new solution better than the best solution found so far (this is called aspiration in the terminology of tabu search). The *directed perturbations* rely thus both on 1) history information which keeps track, for each move, the last time (iteration) when it was performed and 2) the quality of the moves to be applied for perturbation in order not to deteriorate too much the perturbed solution.

The eligible moves for the first type of directed perturbation (applied in Alg. 2 line 6) are identified by the set A_1 such that:

$$A_1 = \{m | m \in M_1, max\{g_m\}, (H_m + \gamma) < Iter \text{ or } (g_m + f_c) > f_{best}\}$$

where g_m is the gain for performing move m (see Sections 2.1 and 2.2), f_c the objective value of the current solution, and f_{best} the objective value of the best found solution. Note that the first directed perturbation considers a subset of eligible move obtained by applying the move operator M_1 (see Section 2.1).

The second type of directed perturbation (applied in Alg. 2, line 7) is almost the same as the first type. The only difference is that the second directed perturbation considers eligible moves obtained with the move operator M_2 (see Section 2.1). These moves are identified by the set A_2 such that:

$$A_2 = \{m | m \in M_2, max\{g_m\}, (H_m + \gamma) < Iter \text{ or } (g_m + f_c) > f_{best}\}$$

Finally, the *random perturbation*, consists in performing randomly selected moves (i.e., M_3 from Section 2.1). More formally, moves performed during random perturbation are identified by the set B such that:

$$B = \{m | m \in M_3\}$$

Since random perturbations can displace any vertex of the partition without constraint, the quality of the resulting solution could be severely affected. In this sense, this perturbation is significantly stronger than the directed perturbations.

As soon as a search stagnation is detected, i.e., the best found solution has not been improved after consecutively visiting T local optima, BLS applies moves of random perturbation in order to drive the search towards distant regions of the search space (lines 1–3 of Alg. 2). Otherwise, BLS applies probabilistically these three types of perturbations. The probability of applying a particular perturbation is determined dynamically depending on the current number of consecutive non-improving attractors visited (indicated by ω in Alg. 1). The idea is to apply more often directed perturbations (with a higher probability) at the beginning of the search, i.e., as the search progresses towards improved new local optima (the non-improving consecutive counter ω is small). With the increase of ω, the probability of using directed perturbations progressively decreases while the probability of applying random moves increases for the purpose of a stronger diversification.

Additionally, it has been observed from an experimental analysis that it is often useful to guarantee a minimum of applications of a directed perturbation. Therefore, we constraint the probability P of applying directed perturbations to take values no smaller than a threshold P_0:

$$P = \begin{cases} e^{-\omega/T} & \text{if } e^{-\omega/T} > P_0 \\ P_0 & \text{otherwise} \end{cases} \tag{2}$$

where T is the maximum number of non-improving local optima visited before carrying out a stronger perturbation.

Given probability P of applying directed perturbation, the probability of applying the first and the second type of directed perturbation is determined respectively by $P \cdot Q$ and $P \cdot (1 - Q)$ where Q is a constant from $[0, 1]$ (see Alg. 2). BLS then generates a perturbed solution by applying accordingly the perturbation operator to make the dedicated moves from the sets A_1, A_2 or B (see Alg. 3).

In Section 4.2, we provide an experimental study showing the influence of this perturbation strategy on the performance of our search algorithm.

2.4 Discussion

The general BLS procedure combines some features from several well-established metaheuristics: iterated local search [18], tabu search [10] and simulated annealing [15]. We briefly discuss the similarities and differences between our BLS approach and these methods.

Following the general framework of ILS, BLS uses local search to discover local optima and perturbation to diversify the search. However, BLS distinguishes itself from most ILS algorithms by the combination of multiple perturbation strategies triggered according to the search status, leading to variable levels of diversification. Moreover, with BLS each locally optimal solution returned by the local search procedure is always accepted as the new starting solution no matter its quality, which completely eliminates the acceptance criterion component of ILS.

A further distinction of BLS is the way an appropriate perturbation strategy is selected at a certain stage of the search. As explained in Section 2.3.1, BLS applies a weak perturbation with a higher probability P as long as the search progresses towards improved solutions. As the number of consecutively visited non-improving local optima ω increases, indicating a possible search stagnation, we progressively decrease the probability for a weak perturbation and increase the probability for a strong perturbation. The idea of this adaptive change of probability finds its inspiration from simulated annealing and enables a better balance between an intensified and diversified search.

To direct the search towards more promising regions of the search space, BLS employs perturbation strategies based on the notion of tabu list that is borrowed from tabu search. The tabu list enables BLS to perform moves that do not deteriorate too much the solution quality in a way that the search does not return to the previous local optimum. However, BLS does not consider the tabu list during its local search (descent) phases while each iteration of tabu search is constrained by the tabu list. As such, BLS and tabu search may explore different trajectories during their respective search, leading to different local optima. In fact, one of the keys to the effectiveness of BLS is that it completely excludes diversification during local search, unlike tabu search and simulated annealing for which the intensification and diversification are always intertwined. We argue that during local search, diversification schemes are unnecessary and the compromise between search exploration and exploitation is critical only once a local optimum is reached. Other studies supporting this idea can be found in [2,13]. We show computational evidences to support this assumption in Section 4.2.2.

As we will see in the next section, BLS is able to attain highly competitive results on the set of well-known benchmarks for the Max-Cut problem in comparison with the state of the art algorithms.

3 Experimental results

In this section, we report extensive computational results of our BLS approach and show comparisons with some state of the art methods of the literature. We conduct our experiments on a set of 71 benchmark instances that has been widely used to evaluate Max-Cut algorithms. These instances can be downloaded from http://www.stanford.edu/∼yyye/yyye/Gset/ and include toroidal, planar and random graphs, with the number of vertices ranging from $|V| = 800$ to 20,000, and edge weights of values 1, 0, or -1.

3.1 Parameter settings and comparison criteria

The parameter settings of BLS used in our experiments are given in Table 1. These parameter values were determined by performing a preliminary experiment on a selection of 15 problem instances from the set of the first 54 graphs (G1-G54). In this experiment, we tested different values for each of the five parameters (L_0, T, ϕ, P_0 and Q), while fixing the rest of the parameters to their default values given in Table 1. In Section 4.1, we provide a parameter sensitivity analysis and justify the setting of parameters that is used to obtain the reported results.

Given its stochastic nature, we run our BLS approach 20 times on each of the 71 benchmark instances, each run being limited to $200000|V|$ iterations where $|V|$ is the number of vertices in the given graph instance. The assessment of BLS performance is based on two comparisons: one is against the best-known results ever reported in the literature and the other is against 4 state of the art methods. In addition to information related to the quality criteria (best objective values, average objective values and standard deviation), we also show computing times for indicative purposes. Our computing times are based on a C++ implementation of our BLS algorithm which is compiled with GNU gcc under GNU/Linux running on an Intel Xeon E5440 with 2.83 GHz and 2GB. Following the DIMACS machine benchmark[1], our machine requires 0.23 CPU seconds for r300.5, 1.42 CPU seconds for r400.5, and 5.42 CPU seconds for r500.5.

It should be noted that a fully fair comparative analysis with the existing Max-Cut algorithms from the literature is not a straight-forward task because of the differences in computing hardware, programming language, termination criterion, etc. For this reason, the evaluation is mainly based on the best-known results obtained with different state of art Max-Cut algorithms (Table 2). The comparison with individual algorithms is presented only for indicative purposes and should be interpreted with caution (Table 3). Nevertheless, our experimental study provides interesting indications about the performance of the proposed BLS algorithm relative to these state of the art approaches.

Table 1
Settings of parameters

Parameter	*Description*	*Setting*
L_0	initial jump magnitude	$0.01\|V\|$
T	max. number of non-improving attractors visited before strong perturb. (restart)	1000
ϕ	tabu tenure	$rand[3, \|V\|/10]$
P_0	smallest probability for applying directed perturb.	0.8
Q	probability for applying directed perturb. I over directed perturb. II.	0.5

Table 2
Computational results of BLS on 71 Max-Cut instances. Column f_{prev} shows the best-known results reported in the literature; columns f_{best} and f_{avg} give the best and average result obtained with BLS over 20 runs; column σ shows the standard deviation; column $t(s)$ indicated the average time (in seconds) required by BLS to reach the best result from f_{best}.

Name	$\|V\|$	f_{prev}	f_{best}	f_{avg}	σ	t(s)
G1	800	11624	11624(9)	11612.4	11.16	13
G2	800	11620	11620(11)	11615	5.74	41
G3	800	11622	11622(19)	11621.1	3.92	83
G4	800	11646	11646(16)	11642.8	6.65	214
G5	800	11631	11631(20)	11631	0	14
G6	800	2178	2178(20)	2178	0	18
G7	800	2006	2006(12)	2001.05	6.55	317
G8	800	2005	2005(18)	2004.4	1.8	195
G9	800	2054	2054(14)	2049.95	6.22	97
G10	800	2000	2000(13)	1996.05	5.84	79
G11	800	564	564(20)	564	0	1
G12	800	556	556(20)	556	0	2
G13	800	582	582(20)	582	0	2
G14	800	3064	3064(6)	3062.85	0.91	119
G15	800	3050	3050(20)	3050	0	43
G16	800	3052	3052(8)	3051.1	1.14	70
G17	800	3047	3047(16)	3046.7	0.64	96
G18	800	992	992(14)	991.7	0.46	106
G19	800	906	906(6)	904.55	1.56	20
G20	800	941	941(20)	941	0	9
G21	800	931	931(14)	930.2	1.29	42
G22	2000	13359	13359(1)	13344.45	16.14	560
G23	2000	13354	13344(10)	13340.6	4.08	278
G24	2000	13337	13337(5)	13329.8	6.76	311
G25	2000	13326	**13340**(1)	13333.4	3.68	148
G26	2000	13314	**13328**(3)	13320	6.98	429
G27	2000	3325	**3341**(10)	3332.25	9.79	449
G28	2000	3287	**3298**(8)	3293.85	5.31	432
G29	2000	3394	**3405**(1)	3388.2	5.94	17
G30	2000	3403	**3412**(10)	3404.85	9.61	283
G31	2000	3299	**3309**(8)	3305.3	4.79	285
G32	2000	1410	1410(13)	1409.3	0.95	336
G33	2000	1382	1382(1)	1380.1	0.44	402
G34	2000	1384	1384(20)	1384	0	170
G35	2000	7684	7684(2)	7680.85	0.57	442
G36	2000	7677	**7678**(1)	7673.6	1.37	604
G37	2000	7689	7689(1)	7685.85	2.1	444
G38	2000	7681	**7687**(4)	7684.95	2.33	461
G39	2000	2397	**2408**(13)	2405.35	3.72	251
G40	2000	2392	**2400**(1)	2394.6	4.65	431

Table 2
Continued.

Name	$\|V\|$	f_{prev}	f_{best}	f_{avg}	σ	t(s)
G41	2000	2398	**2405**(17)	2403	4.98	73
G42	2000	2474	**2481**(2)	2475.4	2.97	183
G43	1000	6660	6660(18)	6658.15	5.57	26
G44	1000	6650	6650(14)	6647.7	3.66	43
G45	1000	6654	6654(15)	6652.15	4.67	104
G46	1000	6649	6649(13)	6647.75	2.26	67
G47	1000	6656	**6657**(11)	6654.35	3.53	102
G48	3000	6000	6000(20)	6000	0	0
G49	3000	6000	6000(20)	6000	0	0
G50	3000	5880	5880(19)	5879.9	0.44	169
G51	1000	3847	**3848**(17)	3847.85	0.36	81
G52	1000	3850	**3851**(19)	3850.85	0.65	78
G53	1000	3848	**3850**(13)	3849.5	0.74	117
G54	1000	3850	**3852**(10)	3850.6	1.74	131
G55	5000	10236	**10294**(2)	10282.4	5.67	842
G56	5000	3949	**4012**(1)	3998.65	7.19	786
G57	5000	3460	**3492**(2)	3488.6	2.11	1440
G58	5000	19248	**19263**(1)	19255.9	4.06	1354
G59	5000	6019	**6078**(1)	6067.9	5.99	2485
G60	7000	14057	**14176**(1)	14166.8	4.53	2822
G61	7000	5680	**5789**(1)	5773.35	7.23	7420
G62	7000	4822	**4868**(2)	4863.8	2.4	5465
G63	7000	26963	**26997**(1)	26980.7	6.25	6318
G64	7000	8610	**8735**(1)	8735.0	9.6	4090
G65	8000	5518	**5558**(2)	5551.2	2.63	4316
G66	9000	6304	**6360**(1)	6350.2	4.37	6171
G67	10000	6894	**6940**(1)	6935.3	2.85	3373
G70	10000	9499	**9541**(1)	9527.1	7.89	11365
G72	10000	6922	**6998**(2)	6935.3	2.85	12563
G77	14000	–	9926(1)	9916.1	4.17	9226
G81	20000	–	14030(1)	14021.7	5.77	20422
# Improved			33			
# Matched			35			
# Worse			1			

3.2 Comparisons with the current best-known solutions

Table 2 summarizes the computational results of BLS on the set of 71 Max-Cut instances (G1-G81) in comparison with the current best-known results (column f_{prev}) which are from references [7,16,19,21,22]. For BLS, we report the best objective value f_{best}, the average objective value f_{avg}, the standard deviation σ, and the average CPU time in seconds required for reaching f_{best} over 20 executions. The results from Table 2 show that BLS is able to improve the previous best-known results for 33 instances [2], and reach the best-known solution in 35 cases. Out of the 71 instances, BLS is unable to reach the best-known result only in one case. As far as we know, solutions for the two largest Max-Cut instances (G77 and G81) have not been reported in the literature. However, for future comparisons, we include in Table 2 the result obtained by

[1] dmclique, ftp://dimacs.rutgers.edu in directory /pub/dsj/clique
[2] Our best results are available at http://www.info.univ-angers.fr/pub/hao/BLS_max_cut.html

BLS for these two instances. As for the computing time required to reach its best solution from column f_{best}, BLS takes on average a time ranging from less than one second to 10 minutes for instances with up to 3000 vertices. For the large and very large instances with 5000 to 20000 vertices, the computing time needed goes from 0.2 to 5.6 hours.

3.3 Comparisons with the current best performing approaches

To further evaluate the performance of BLS, we compare it with the following algorithms that achieve state-of-art performance:

(1) Two very recent GRASP-Tabu search algorithms [22] – a GRASP-Tabu Search algorithm working with a single solution (GRASP-TS) and its reinforcement (GRASP-TS/PM) by a population management strategy. The reported results were obtained on a PC running Windows XP with Pentium 2.83GHz CPU and 2GB RAM (the same computing platform as that we used).
(2) Scatter search (SS) [19] – an advanced scatter search incorporating several inovative features. The evaluation of SS was performed on a machine with a 3.2 GHz Intel Xenon processor and 2GB of RAM (a comparable computer as that we used).
(3) Rank-2 relaxation heuristic (CirCut) [7] – a method based on a relaxation of the problem. The results reported in this paper for CirCut were obtained under the same conditions as that of SS and are taken from [19].

Since large Max-Cut instances (G55-G81) are very rarely used in the literature for algorithm evaluation, we limit this comparison to the first 54 Max-Cut instances which are also the most commonly used in the literature.

Table 3 provides the results of this comparison with the four reference approaches. For each approach, we report the percentage deviation ρ from the best-known solution (column f_{prev} from Table 2), computed as $\%\rho = 100 * (f_{prev} - f)/f_{prev}$, where f is the best objective value attained by a given approach. Moreover, we show for each algorithm the required average time in seconds, taken from the corresponding papers.

GRASP-TS/PM is one of the current most effective algorithm for the Max-Cut problem, which attains the previous best-known result for 41 out of 54 instances, with an average percentage deviation of 0.051. GRASP-TS also provides excellent performance on these instances, compared to the current state-of-art Max-Cut heuristics. It is able to reach the previous best-known solution for 22 instances with an average percentage deviation of 0.201. The other two popular Max-Cut approaches, SS and CirCut, can obtain the previous best-known result in 11 and 12 cases respectively, with an average percentage

Table 3
Comparison of BLS with the four reference approaches on the most commonly used 54 benchmark instances. For each approach, we report the percentage deviation $\%\rho$ of the best result obtained by the given algorithm from the best-known result reported in the literature. Column $t(s)$ shows the average time in seconds required for reaching the best result.

	BLS		GRASP-TS [22]		GRASP-TS/PM [22]		SS [19]		CirCut [7]	
Name	$\%\rho$	t(s)	$\%\rho$	t(s)	$\%\rho$	t(s)	$\%\rho$	t(s)	$\%\rho$	t(s)
G1	0	13	0	100	0	47	0	139	0	352
G2	0	41	0	677	0	210	0	167	0.026	283
G3	0	83	0	854	0	297	0	180	0	330
G4	0	214	0	155	0	49	0	194	0.043	524
G5	0	14	0	235	0	232	0	205	0.034	1128
G6	0	18	0	453	0	518	0.597	176	0	947
G7	0	317	0	304	0	203	1.196	176	0.150	867
G8	0	195	0	565	0	596	0.848	195	0.099	931
G9	0	97	0	581	0	559	0.682	158	0.292	943
G10	0	79	0	845	0	709	0.35	210	0.3	881
G11	0	1	0	18	0	10	0.355	172	0.709	74
G12	0	2	0	723	0	233	0.719	241	0.719	58
G13	0	1	0	842	0	516	0.687	228	1.374	62
G14	0	119	0.065	812	0	1465	0.131	187	0.196	128
G15	0	43	0.328	419	0	1245	0.033	143	0.033	155
G16	0	70	0.098	1763	0	335	0.229	162	0.229	142
G17	0	96	0.131	1670	0	776	0.131	313	0.328	366
G18	0	106	0	977	0	81	0.403	174	1.411	497
G19	0	20	0	490	0	144	0.331	128	1.989	507
G20	0	9	0	578	0	80	0	191	0	503
G21	0	42	0.430	484	0	667	0.107	233	0	524
G22	0	560	0.097	983	0.075	1276	0.097	1336	0.097	493
G23	0.075	278	0.271	1668	0.165	326	0.277	1022	0.277	457
G24	0	311	0.104	643	0.097	1592	0.255	1191	0.172	521
G25	-0.105	148	0.083	767	0	979	0.045	1299	0	1600
G26	-0.105	429	0.060	1483	0.008	1684	0.150	1415	0	1569
G27	-0.481	449	0.271	256	0	832	0.211	1437	0.571	1456
G28	-0.335	432	0.365	81	0	1033	0.061	1314	0.821	1543
G29	-0.324	17	0.236	21	0	993	0.147	1266	0.530	1512
G30	-0.264	283	0.235	1375	0.029	1733	0	1196	0.529	1463
G31	-0.303	285	0.394	904	0	888	0.333	1336	0.424	1448
G32	0	336	1.135	903	0.284	1232	0.851	901	1.418	221
G33	0	401	1.013	1501	0.579	506	1.447	926	1.592	198
G34	0	170	0.578	1724	0.578	1315	1.445	950	1.156	237
G35	0	442	0.403	1124	0.299	1403	0.208	1258	0.182	440
G36	-0.013	604	0.404	543	0.221	1292	0.221	1392	0.221	400
G37	0	444	0.325	983	0.247	1847	0.325	1387	0.299	382

deviation of 0.292 and 0.342.

The results from Table 3 show that BLS outperforms the four reference algorithms in terms of solution quality. Indeed, the average percentage deviation of the best results obtained with BLS is -0.064, meaning that BLS improves on average the best-known result by 0.064%. Moreover, BLS also seems to be highly competitive with other approaches in terms of computing time. The average run-time required by BLS for the 54 instances is 176 seconds, which is significantly less than the average time required by the four reference approaches, considering that the reported results were obtained on comparable

Table 3
Continued.

	BLS		GRASP-TS [22]		GRASP-TS/PM [22]		SS [19]		CirCut [7]	
Name	%ρ	t(s)	%ρ	t(s)	%ρ	t(s)	%ρ	t(s)	%ρ	t(s)
G38	-0.078	461	0.365	667	0.143	1296	0	1012	0.456	1189
G39	-0.459	251	0.375	911	0	742	0.167	1311	0.083	852
G40	-0.334	431	0.585	134	0	1206	0.753	1166	0.209	901
G41	-0.292	73	1.293	612	0	1490	0.500	1016	0	942
G42	-0.283	183	0.849	1300	0	1438	0.687	1458	0.202	875
G43	0	26	0	969	0	931	0.060	406	0.060	213
G44	0	43	0.015	929	0.015	917	0.030	356	0.105	192
G45	0	104	0	1244	0	1791	0.180	354	0.030	210
G46	0	67	0.015	702	0	405	0.226	498	0.060	639
G47	-0.015	102	0	1071	0	725	0.105	359	0	633
G48	0	0	0	13	0	4	0	20	0	119
G49	0	0	0	27	0	6	0	35	0	134
G50	0	169	0	80	0	14	0	27	0	231
G51	-0.026	81	0.104	628	0	701	0.026	513	0.260	497
G52	-0.026	78	0.156	1274	0	1228	0.026	551	0.442	507
G53	-0.052	117	0.026	1317	0	1419	0.052	424	0.156	503
G54	-0.052	131	0.052	1231	0	1215	0.104	429	0.208	524
Avg.	-0.064	176	0.201	771	0.051	804	0.292	621	0.342	617

computers.

To see whether there exists significant performance difference in terms of solution quality among BLS and the reference algorithms, we apply the Friedman non-parametric statistical test followed by the Post-hoc test on the results from Table 3. From the Friedman test, we observe that there is a significant performance difference among the compared algorithms (with a p-value less than 2.2e-16). Moreover, the Post-hoc analysis shows that BLS statistically outperforms GRASP-TS, SS and CirCut with p-values of 7.081000e-09, 9.992007e-16 and 0.000000e+00 respectively. However, the performance between BLS and GRASP-TS/PM is statistically less significant with a p-value of 9.547157e-02.

4 Experimental analyses

4.1 Parameter sensitivity analysis

The performed parameter sensitivity analysis is based on a subset of 15 selected Max-Cut instances from the set of the first 54 graphs (G1-G54). For each BLS parameter (i.e., L_0, T, ϕ, P_0 and Q), we test a number of possible values while fixing the other parameters to their default values from Table 1. We test values for L_0 in the range $[0.0025|V|, 0.32|V|]$, T in the range [250,2000], P_0 in the range [0.5,0.9] and Q in the range [0.2,0.8]. Similarly, for the tabu tenure ϕ we tried several ranges which induce increasingly larger degrees of diversification into the search. For each instance and each parameter setting, we perform 20 independent runs with the time limit per run set to 30 minutes.

Table 4
Post-hoc test for solution sets obtained by varying the value of L_0

$L_0 =$	$0.0025\|V\|$	$0.005\|V\|$	$0.01\|V\|$	$0.02\|V\|$	$0.04\|V\|$	$0.08\|V\|$	$0.16\|V\|$
$L_0 = 0.005\|V\|$	0.85863						
$L_0 = 0.01\|V\|$	0.19961	0.95858					
$L_0 = 0.02\|V\|$	0.02021	0.53532	0.99145				
$L_0 = 0.04\|V\|$	0.01468	0.47563	0.98450	0.99999			
$L_0 = 0.08\|V\|$	0.65622	0.99997	0.99567	0.76698	0.71316		
$L_0 = 0.16\|V\|$	0.99986	0.97971	0.44552	0.07605	0.05944	0.89527	
$L_0 = 0.32\|V\|$	0.99999	0.74057	0.11999	0.00931	0.00704	0.50545	0.99804

Table 5
Post-hoc test for solution sets obtained by varying the value of P_0

$P_0 =$	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85
$P_0 = 0.55$	0.99999							
$P_0 = 0.60$	0.94419	0.98065						
$P_0 = 0.65$	0.77176	0.87550	0.99998					
$P_0 = 0.70$	0.77154	0.87545	0.99998	1.00000				
$P_0 = 0.75$	0.22748	0.33802	0.94411	0.99529	0.99529			
$P_0 = 0.80$	0.69552	0.81722	0.99985	0.99999	0.99999	0.99840		
$P_0 = 0.85$	0.10606	0.17438	0.81728	0.96221	0.96221	0.99999	0.98063	
$P_0 = 0.90$	0.00557	0.01134	0.22809	0.47073	0.47048	0.94417	0.55630	0.99128

We use the Friedman statistical test to see whether there is any difference in BLS performance, in terms of its average result, when varying the value of a single parameter as mentioned above. The Friedman test shows that there is no statistical difference in the performance (with *p-value* > 0.8) when varying respectively the values of T, ϕ, and Q. This implies that these three parameters exhibit no particular sensitivity. On the other hand, when varying values of parameters L_0 and P_0 respectively, the Friedman test revealed a statistical difference in performance with *p-value* $= 0.006906$ and *p-value* $= 0.005237$. We thus perform the Post-hoc test on these solution sets obtained respectively with different settings of parameters L_0 and P_0. The results of these analyses are provided in tables 4 and 5 for L_0 and P_0 respectively, where each table entry shows the *p-value* for two sets of average results obtained with two different values of the corresponding parameter.

From the results in Table 4 for L_0, we observe that for any two tested values of L_0 the *p-value* is generally significant (often *p-value* > 0.5), except in 4 case when the *p-value* < 0.05. This implies that L_0 is not highly sensitive to different settings. However, the analytical results from Table 5 show that P_0 is even less sensitive than L_0. Indeed, the p-value for two solution sets obtained with different values of P_0 is often very close to 1. Only in 2 cases, the difference is statistically significant (with *p-value* < 0.01).

To further investigate the performance of BLS with different values for L_0 and P_0, we show in figures 2 and 3 the box and whisker plots which indicate, for each tested parameter value, the distribution and range of the obtained results for the 15 used instances. For the sake of clarity, these results are expressed as the percentage deviation of the average result from the best-known solution f_{best} reported in the literature.

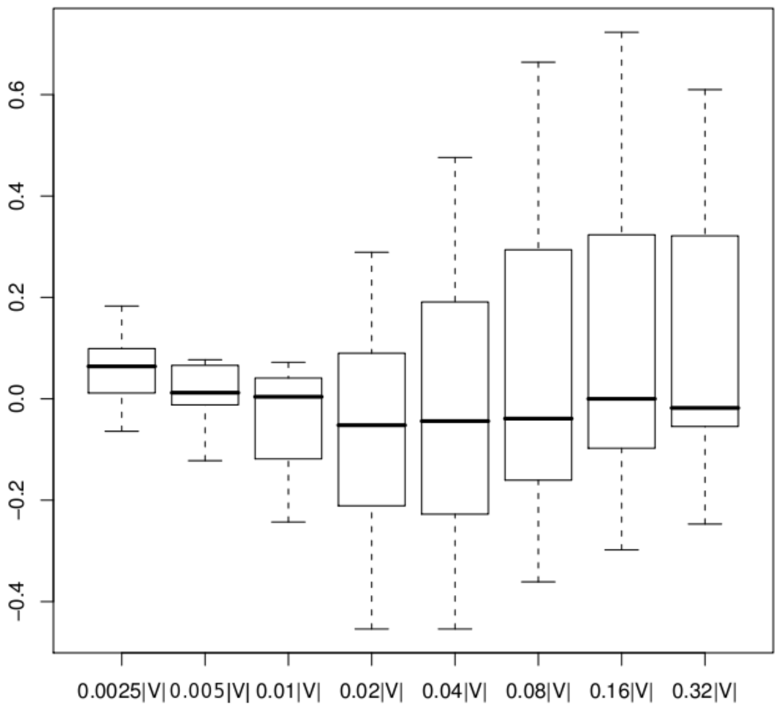

Fig. 2. Box and whisker plot of the results obtained with different settings of parameter L_0 for 15 selected instances.

From the box and whisker plot in Figure 2, we observe a visible difference in the distribution of results among the data sets obtained with different settings of parameter L_0. For the set of results generated with small values of $L_0 \leq 0.01|V|$, the plot indicates a significantly smaller variation compared to the results obtained with larger values of L_0. For instance, the comparison between the two sets of results, obtained with $L_0 = 0.01|V|$ (set S_1) and $L_0 = 0.04|V|$ (set S_2), indicates that the percentage deviations of the results from S_1 and S_2 range from -0.25% to 0.07% and from -0.45% to 0.5% respectively. More precisely, around 25% of the results from S_2 have a lower percentage deviation from f_{best} than any of the result from S_1, while another 37% of the results from S_2 have a higher percentage deviation from f_{best} than any of the result from S_1. We can thus conclude that a lower value for L_0 (e.g., $L_0 = 0.01|V|$) is a better choice since the deviations from the best-known result does not vary much from one instance to another.

From the box and whisker plot in Figure 3, we observe that the difference in the distribution and variation of results among the solution sets generated with different settings of parameter P_0 is less evident than in Figure 2. This confirms our previous observations from the Post-hoc analysis that P_0 is slightly less

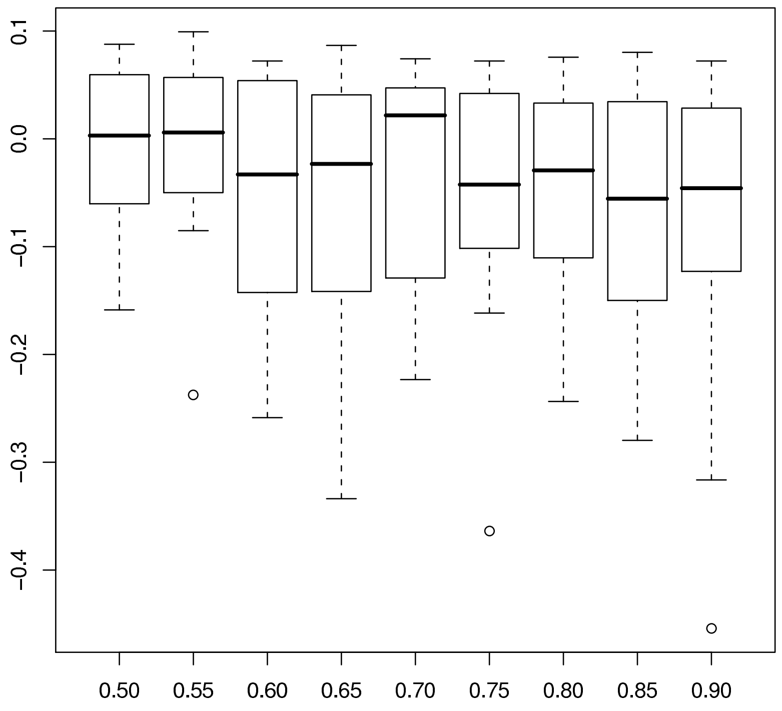

Fig. 3. Box and whisker plot of the results obtained with different settings of parameter P_0 for 15 selected instances.

sensitive than L_0.

4.2 Influence of diversification strategies: comparisons and discussions

The objective of this section is twofold. First, we wish to highlight the contribution of the proposed diversification mechanism to the overall performance of the BLS method. In Section 4.2.1, we thus provide a statistical comparison between several variants of BLS integrating different perturbation mechanisms. Second, we try to justify our statement made in Section 2.4 that diversification schemes are crucial only once a local optimum is reached and should be excluded during local search. For this purpose, we provide in Section 4.2.2 a comparison with tabu search and iterated tabu search (ITS) methods which, because of the tabu list, induce a certain degree of diversification at each iteration. These two methods are obtained with minimal changes of our BLS algorithm.

We perform the comparisons using the set of the 54 Max-Cut instances (G1-G54). In every experiment and for each instance, the reported results are obtained under the same conditions, i.e., after 20 independent executions with the maximum time limit per run set to 30 minutes.

4.2.1 Comparison with different variants of the perturbation mechanism

In this section, we perform a comparison between several versions of our algorithm, which employ different diversification strategies to escape local optima. The diversification mechanism of the first version (call it V_1) is based solely on random moves of type M_3 (selected from the set of moves B, see Section 2.3.1). The diversification mechanism of the second version (call it V_2) adaptively switches between the directed perturbation, which performs moves of type M_1 from the set A_1 (see Section 2.3.1), and the random perturbation. The third version (call it V_3) integrates a diversification strategy that combines the two types of directed perturbations, which effectuate respectively moves from the set A_1 (moves of type M_1) and the set A_2 (moves of type M_2). The last version is our default BLS algorithm detailed in Section 2. Please note that the strongest diversification is induced by V_1 since only random moves are considered for perturbation. On the other hand, the weakest diversification is introduced with V_3 since all the perturbation moves consider the quality criterion so as not to degrade too much the resulting solution.

The results of this experiment are shown in Table 6. Columns f_{best} and f_{avg} provide respectively the best and average results obtained by each algorithm. The results indicate that the three algorithms, i.e., V_2, V_3 and default BLS, clearly outperform algorithm V_1 which combines the descent local search with the most basic perturbation mechanism based on random moves. More precisely, the best results reported in columns f_{best} indicate that our default BLS algorithm reports a better solution than V_1 for 36 out of the 54 instances and attains the same result as V_1 for the other 18 instances. However, the difference between V_2, V_3 and the default BLS is much less obvious. Indeed, we observe from columns f_{best} that the default BLS outperforms V_2 and V_3 in only 5 and 6 cases respectively (out of 54), and is outperformed on 4 and 5 instances respectively. To see whether there is a statistically significant difference in average results (from column f_{avg}) obtained by the four algorithms, we perform the Friedman test followed by the Post-hoc test. The Friedman test revealed a significant difference with the $p\text{-}value < 2.2e - 16$. As expected, the Post-hoc test showed a significant difference between the sets of average solutions obtained with V_1 and the default BLS algorithm with the p-value of 0.000000e+00. Moreover, the Post-hoc test showed that the default BLS algorithm statistically outperforms V_2 in terms of average performance with the p-value of 1.701825e-03. However, there is no statistical difference in average performance between V_3 and the default BLS algorithm with the p-value of 7.734857e-01. From Table 6, we observe that V_3 outperforms the three other algorithms, in terms of average results, for a number of small instances (i.e., G1 – G10). This implies that the weakest diversification insures the best performance for these instances. On the other hand, the default BLS algorithm provides better average results than V_3 for a number of more difficult instances (e.g., G36 – G42, G51 – G54).

Table 6
Computational result obtained with four different diversification mechanisms. For each algorithm, columns f_{best} and f_{avg} show the best and average result over 20 runs.

	$M_3(V_1)$		$M_1M_3(V_2)$		$M_1M_2(V_3)$		*Complete*	
	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
G1	0	0.407	0	0.091	0	0.072	0	0.101
G2	0.026	0.478	0	0.113	0	0.038	0	0.046
G3	0.043	0.443	0	0.052	0	0	0	0.023
G4	0.172	0.505	0	0.062	0.010	0.035	0	0.012
G5	0.043	0.432	0	0.031	0	0	0	0.012
G6	0.138	2.162	0	0.191	0	0	0	0.044
G7	0.648	2.806	0	0.501	0	0	0	0.187
G8	0.998	2.117	0	0.304	0	0	0	0.067
G9	0	2.347	0	0.613	0	0	0	0.204
G10	0.4	2.123	0	0.238	0	0.120	0	0.270
G11	0	0	0	0	0	0	0	0
G12	0	0	0	0	0	0	0	0
G13	0	0	0	0	0	0	0	0
G14	0.033	0.165	0	0.042	0.033	0.119	0	0.036
G15	0	0.256	0	0.011	0.011	0.175	0	0
G16	0	0.224	0	0.034	0	0.106	0	0.021
G17	0	0.184	0	0.026	0.033	0.138	0	0.021
G18	0.101	0.968	0	0.297	0.210	0.877	0	0.066
G19	0	1.176	0	0.348	0	0.717	0	0.270
G20	0	1.509	0	0.122	0.210	1.190	0	0
G21	0	1.241	0	0.521	0	1.004	0	0.064
G22	0.467	0.844	0.067	0.467	0.030	0.367	0	0.257
G23	0.322	0.565	0.075	0.151	0.07	0.092	0.075	0.089
G24	0.067	0.473	0	0.104	0	0.017	0	0.058
G25	0.038	0.336	-0.105	-0.012	-0.105	-0.052	-0.105	-0.051
G26	0.158	0.321	-0.105	0.004	-0.105	-0.077	-0.105	-0.040
G27	0.451	1.463	-0.331	0.053	-0.481	-0.457	-0.481	-0.331
G28	1.065	1.771	-0.335	-0.090	-0.335	-0.240	-0.335	-0.129
G29	0.619	1.499	-0.324	0.355	-0.324	-0.066	-0.324	0.080
G30	0.558	1.604	-0.294	0.066	-0.294	-0.239	-0.264	-0.076
G31	0.606	1.552	-0.251	0.027	-0.303	-0.174	-0.303	-0.177
G32	0.142	0.277	0	0.007	0	0	0	0.057
G33	0.145	0.412	0	0.109	0	0.015	0	0.137
G34	0	0.195	0	0	0	0	0	0
G35	0.104	0.228	0	0.010	0.013	0.142	0	0.038
G36	0.091	0.231	0	0.032	0.026	0.138	-0.013	0.049
G37	0.104	0.256	0	0.032	0.065	0.161	0	0.029
G38	0.065	0.182	-0.091	-0.055	0.013	0.098	-0.065	-0.040
G39	0.167	0.766	-0.459	-0.200	-0.167	0.465	-0.459	-0.348
G40	0.293	1.248	-0.334	-0.115	0.084	0.717	-0.293	-0.123
G41	-0.291	1.372	0	0.056	-0.292	0.922	-0.292	-0.265
G42	0.121	1.071	-0.283	0.196	0.081	0.879	-0.283	-0.071
G43	0.090	0.437	0	0.052	0	0	0	0.045
G44	0.075	0.452	0	0.055	0	0.023	0	0.042
G45	0	0.325	0	0.106	0	0	0	0.033
G46	0.211	0.426	0	0.119	0	0.006	0	0.043
G47	0.060	0.500	-0.015	0.064	-0.015	0.002	-0.015	0.053
G48	0	0	0	0	0	0	0	0
G49	0	0.100	0	0	0	0	0	0
G50	0	0.607	0	0	0	0	0	0.003
G51	-0.026	0.207	-0.026	-0.009	-0.026	0.119	-0.026	-0.017
G52	0.026	0.148	-0.026	-0.018	-0.026	0.095	-0.026	-0.021
G53	0.026	0.173	-0.052	-0.031	-0.052	0.084	-0.052	-0.045
G54	0.052	0.197	-0.052	0.019	0.026	0.142	-0.052	-0.004

To conclude, the directed perturbation, which uses dedicated information, is more beneficial than the random perturbation for the tested Max-Cut instances. Nevertheless, for some difficult instances, an even better average performance with BLS is obtained when combining perturbation strategies that introduce varying degrees of diversification into the search process.

4.2.2 Comparison with tabu search (TS) and iterated tabu search (ITS)

As previously mentioned, one of the keys to the effectiveness of BLS is that it completely excludes diversification during its local search phase. Indeed, the descent phases of BLS are carried out by moves that are selected by considering only the quality criterion. To provide some justifications to this idea, we perform a comparison with a tabu search algorithm (TS) and an iterated tabu search algorithm (ITS) which are obtained by making minor modifications to our BLS algorithm.

The TS algorithm used for this comparison consists in performing moves identified by the two sets A_1 and A_2 (see Section 2.3.1 for the definition of sets A_1 and A_2). With equal probability, TS performs a move either from A_1 or from A_2. Note that this procedure is simply the directed perturbation of BLS. For ITS, we modify our BLS algorithm by excluding the adaptive perturbation strategy. The local search phase of ITS is the above-mentioned TS procedure, while the perturbation mechanism corresponds to the random perturbation performed by BLS. The perturbation phase of ITS is triggered if the best found solution is not improved after 10000 iterations of the TS procedure. It is important to note that the tabu list of TS and ITS insures a certain degree of diversification at each iteration.

For each approach, we report in Table 7 the best and average result in columns f_{best} and f_{avg} respectively. From column f_{best}, we observe that for 19 instances, BLS finds better results than both TS and ITS. In other 35 cases, the best solution attained by BLS is at least as good as that reached by both TS and ILS.

To see whether BLS statistically outperforms TS and ITS in terms of solution quality, we apply the Friedman non-parametric statistical test followed by the Post-hoc test on the average results from Table 7. The Friedman test discloses that there is a significant difference among the compared algorithms with a p-value of 4.724e-05. Moreover, the Post-hoc analysis shows that BLS is statistically better than TS and ITS with a p-value of 6.554058e-04 and 4.615015e-05 respectively. Although we did not include the average time required by each approach to reach the best result from f_{best}, BLS remains highly competitive with TS and ILS also in terms of computational time. While TS and ILS need on average 236 and 330 seconds respectively to reach their best

Table 7
Comparison of BLS with tabu search (TS) and iterated local search (ITS). For each algorithm, columns f_{best} and f_{avg} show the best and average result over 20 runs.

	BLS		*TS*		*ITS*	
	f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}
G1	0	0.101	0	0.001	0	0
G2	0	0.046	0	0.004	0	0.009
G3	0	0.023	0	0.008	0	0.009
G4	0	0.012	0	0.004	0	0.004
G5	0	0.012	0	0.003	0	0.004
G6	0	0.044	0	0.087	0	0.071
G7	0	0.187	0	0.032	0	0.017
G8	0	0.067	0	0.082	0	0.092
G9	0	0.204	0.049	0.119	0	0.097
G10	0	0.270	0.05	0.125	0	0.118
G11	0	0	0	0	0	0
G12	0	0	0	0	0	0
G13	0	0	0	0	0	0
G14	0	0.036	0.033	0.080	0	0.059
G15	0	0	0	0.102	0	0.113
G16	0	0.021	0	0.090	0	0.057
G17	0	0.021	0	0.062	0	0.108
G18	0	0.066	0.101	0.307	0	0.161
G19	0	0.270	0	0.199	0	0.221
G20	0	0	0	0.367	0	0.499
G21	0	0.064	0	0.585	0	0.585
G22	0	0.257	0.212	0.542	0.198	0.421
G23	0.075	0.089	0.142	0.200	0.142	0.208
G24	0	0.058	0.060	0.109	0.075	0.127
G25	-0.105	-0.051	-0.015	0.026	-0.015	0.026
G26	-0.105	-0.040	-0.030	0.035	-0.045	0.032
G27	-0.481	-0.331	-0.271	0.002	-0.241	0.036
G28	-0.335	-0.129	0	0.141	-0.122	0.105
G29	-0.324	0.080	-0.147	0.088	-0.059	0.153
G30	-0.264	-0.076	-0.088	0.101	-0.029	0.084
G31	-0.303	-0.177	-0.091	0.192	-0.061	0.196
G32	0	0.057	0	0	0	0
G33	0	0.137	0	0	0	0
G34	0	0	0	0	0	0
G35	0	0.038	0.104	0.159	0.065	0.146
G36	-0.013	0.049	0.091	0.153	0.078	0.177
G37	0	0.029	0.039	0.185	0.117	0.179
G38	-0.065	-0.040	-0.052	0.131	0.052	0.131
G39	-0.459	-0.348	-0.167	0.325	-0.417	0.280
G40	-0.293	-0.123	-0.167	0.355	-0.084	0.337
G41	-0.292	-0.265	-0.291	0.542	-0.292	0.588
G42	-0.283	-0.071	-0.081	0.400	-0.040	0.758
G43	0	0.045	0	0.022	0	0.0188
G44	0	0.042	0	0.038	0.015	0.040
G45	0	0.033	0	0.026	0	0.032
G46	0	0.043	0	0.045	0.015	0.050
G47	-0.015	0.053	0	0.025	0	0.026
G48	0	0	0	0	0	0
G49	0	0	0	0	0	0
G50	0	0.003	0	0	0	0.005
G51	-0.026	-0.017	-0.026	0.057	-0.026	0.077
G52	-0.026	-0.021	0	0.069	0	0.064
G53	-0.052	-0.045	-0.052	0.012	-0.052	0.012
G54	-0.052	-0.004	0	0.108	-0.026	0.082

results reported in f_{best}, BLS requires on average around 180 seconds for the 54 instances.

5 Conclusion

In this paper, we presented the Breakout Local Search approach for the Max-Cut problem. The BLS alternates between a local search phase (to find local optima) and a perturbation-based diversification phase (to jump from a local optimum to another local optimum). The diversification phase is of an extreme importance for the performance of BLS since the local search alone is unable to escape a local optimum. The diversification mechanism of the proposed approach adaptively controls the jumps towards new local optima according to the state of the search. This is achieved by varying the magnitude of a jump and selecting the most suitable perturbation for each diversification phase.

Experimental evaluations on a popular set of benchmark instances showed that despite its simplicity, our approach outperforms all the current Max-Cut algorithms in terms of solution quality. Out of the 71 benchmark instances, BLS improves the current best results in 33 cases and attains the previous best-known result for 35 instances. Moreover, the computing time required by BLS to reach the reported results competes very favorably compared to other state-of-the-art approaches. To attain its best results reported in the paper, BLS needs less than one second to 10 minutes for the graphs with up to 3000 vertices, and 0.2 to 5.6 hours for the large and very large instances with 5000 to 20000 vertices.

We also provided experimental evidences to highlight the importance of the adaptive perturbation strategy employed by the proposed BLS approach, and the benefit of separating completely diversification from intensification during the local search phases.

Acknowledgment

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

References

[1] E. Arráiz, O. Olivo. Competitive simulated annealing and Tabu Search algorithms for the max-cut problem. GECCO 2009: 1797–1798, 2009.

[2] R. Battiti, M. Protasi. Reactive search, a history-based heuristic for max-sat. *ACM Journal of Experimental Algorithmics*, 2:2, 1996.

[3] R. Battiti, M. Brunato, F. Mascia. Reactive search and intelligent optimization. Operations Research/Computer Science Interfaces Series 45, 2009.

[4] U. Benlic, J.K. Hao. Breakout local search for maximum clique problems. Computers & Operations Research, (in press, http://dx.doi.org/10.1016/j.cor.2012.06.002).

[5] U. Benlic, J.K. Hao. A study of breakout local search for the minimum sum coloring problem. To appear in Bui LT et al. (Eds.) SEAL 2012, Lecture Notes in Computer Science, 2012.

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

[7] S. Burer, R.D.C. Monteiro, Y. Zhang. Rank-two relaxation heuristics for MAX-CUT and other binary quadratic programs. *SIAM J. on Optimization*, 12: 503–521, 2002.

[8] P. Festa, P.M. Pardalos, M.G.C. Resende, C.C. Ribeiro. Randomized heuristics for the maxcut problem. *Optimization Methods and Software*, 17: 1033–1058, 2002.

[9] C. Fiduccia, R. Mattheyses R. A linear-time heuristics for improving network partitions. *In Proceedings of the 19th Design Automation Conference*, 171–185, 1982.

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

[11] S. Kahruman, E. Kolotoglu, S. Butenko, I.V. Hicks. On greedy construction heuristics for the MAX-CUT problem. *International Journal of Computational Science and Engineering*, 3(3): 211–218, 2007.

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

[13] J.P. Kelly, M. Laguna, F. Glover. A study of diversification strategies for the quadratic assignment problem. *Computers and Operations Research*, 21(8):885 – 893, 1994

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

[15] S. Kirkpatrick, C.D. Gelett, M.P. Vecchi. Optimization by simulated annealing. *Science*, 220: 621–630, 1983.

[16] G.A. Kochenberger, J.K. Hao, Z. Lü, H. Wang, F. Glover. Solving large scale Max Cut problems via tabu search. To appear in *Journal of Heuristics*, DOI: 10.1007/s10732-011-9189-8, 2012.

[17] K. Krishnan, J.E. Mitchell. A Semidefinite Programming Based Polyhedral Cut and Price Approach for the Maxcut Problem. *Computational Optimization and applications*, 33(1): 51–71, 2006.

[18] H.R. Lourenco, O.Martin, T.Stützle. Iterated local search. Handbook of Metaheuristics, Springer-Verlag, Berlin Heidelberg, 2003.

[19] R. Martí, A. Duarte, M. Laguna. Advanced Scatter Search for the Max-Cut Problem. *INFORMS Journal on Computing*, 21(1): 26–38, 2009.

[20] C.H. Papadimitriou, K. Steiglitz. Combinatorial Optimization : Algorithms and Complexity Second edition by Dover, 1998.

[21] V.P. Shylo, O.V. Shylo. Solving the maxcut problem by the global equilibrium search. *Cybenetics and Systems Analysis*, 46(5): 744–754, 2010.

[22] Y. Wang, Z. Lü, F. Glover, J.K. Hao. Probabilistic GRASP-tabu search algorithms for the UBQP problem. To appear in *Computers and Operations Research*, DOI: 10.1016/j.cor.2011.12.006, 2012.