Breakout Local Search for Maximum Clique Problems

Una Benlic and Jin-Kao Hao *

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

Accepted to *Computers and Operations Research*, 6 June 2012

Abstract

The maximum clique problem (MCP) is one of the most popular combinatorial optimization problems with various practical applications. An important generalization of MCP is the maximum weight clique problem (MWCP) where a positive weight is associate to each vertex. In this paper, we present Breakout Local Search (BLS) which can be applied to both MC and MWC problems without any particular adaptation. BLS explores the search space by a joint use of local search and adaptive perturbation strategies. Extensive experimental evaluations using the DIMACS and BOSHLIB benchmarks show that the proposed approach competes favourably with the current state-of-art heuristic methods for MCP. Moreover, it is able to provide some new improved results for a number of MWCP instances. This paper also reports for the first time a detailed landscape analysis, which has been missing in the literature. This analysis not only explains the difficulty of several benchmark instances, but also justifies to some extent the behaviour of the proposed approach and the used parameter settings.

Keywords: Clique; breakout local search; adaptive diversification; attractor; fitness-distance correlation.

1 Introduction

Given an undirected graph $G = (V, E)$ where V is the set of vertices and E the set of edges, a clique C of G is a subset of V such that all vertices in C are pairwise adjacent, i.e., $\forall v, u \in C, \{v, u\} \in E$. The maximum clique problem

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

(MCP) is to find a clique C of maximum cardinality. MCP is notable for its applicability to a wide range of important applications such as information retrieval, experimental design, classification theory and fault diagnosis [4].

An important generalisation of MCP is the maximum weight clique problem (MWCP), where a positive weight w_v is associated to each vertex $v \in V$ and the weight $W(C)$ of a clique C is defined as $W(C) = \sum_{v \in C} w_v$. MWCP then consists in finding a clique C of G which maximizes $W(C)$. Note that the maximum weight clique is not necessarily a clique of the highest cardinality. However, MCP can be viewed as a special instance of MWCP when a unit weight is assigned to each vertex of V. Applications of MWCP occur in computer vision, robotics, and pattern recognition [3].

Both MCP and MWCP are NP-complete [14]. Over the past decades, much effort has been made in devising exact algorithms [17,20,26] as well as powerful heuristics including tabu search [10,18,30], reactive search [5,23,24], stochastic local search [9,15,22–24], variable neighbourhood search [12], ant colony optimization [28] and hybrid methods [27,31]. Most of these approaches have mainly been applied to MCP.

In this work, we propose a new heuristic approach, called Breakout Local Search (BLS), which we apply without any particular adaptation to both the weighted and unweighted maximum clique problems. BLS can be considered as an iterated local search algorithm [16] which uses the tabu list idea from tabu search [11] for its directed diversification. The basic idea behind BLS is to use local search [21] to discover local optima and employ adaptive diversification strategies to continually move from one local optimum to another in the search space. The continual exploration of new search areas is achieved by alternating between random or directed, and weak or strong perturbations depending on the current search state. Despite its simplicity, BLS shows excellent performance on the set of well-known benchmark instances from the DIMACS and BOSHLIB libraries for the maximum clique problem. BLS is also capable of attaining new improved results for 14 maximum weight clique instances from the DIMACS-W benchmark. Additionally, we show for the first time a thorough landscape analysis on selected MCP and MWCP instances. Based on this analysis, we explain to some extent the behaviour of the proposed algorithm and justify our choice for the parameter settings used to obtain the reported results.

The remained of the paper is organized as follows. In the next section, we present in details the breakout local search approach for MCP and MWCP. Section 3 shows extensive computational results and comparisons on MCP and MWCP benchmark instances. In Section 4, we perform a landscape analysis on selected instances and relate this analysis to the performance of BLS.

2 Breakout Local Search (BLS) for MCP and MWCP

2.1 The BLS procedure

Recall that a local optimum with respect to a given neighborhood N is a solution s^* such that $\forall s \in N(s^*)$, $f(s^*) \geq f(s)$, where f is the objective function to be maximized. A basin of attraction of a local optimum l can be defined as the set B_l of solutions that lead the local search to the given local optimum l, i.e., $B_l = \{s \in S | LocalSearch(s) = l\}$ [6]. Since a local optimum l acts as an attractor with respect to the solutions B_l, the terms attractor and local optimum will be used interchangeably throughout this paper.

Basically, our Breakout Local Search (BLS) approach moves from one basin of attraction formed by a local optimum to another basin by applying directed or undirected, large or small jumps depending on the search state. BLS starts from an initial solution C_0 and applies to it local search to reach a local optimum or an attractor C. Each iteration of the local search algorithm scans the whole neighborhood and selects the best improving neighboring solution to replace the current solution [21]. 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 move into a neighboring basin of attraction. For this purpose, BLS applies a number of dedicated moves to the current optimum C (we say that C is perturbed). Each time an attractor is perturbed, the perturbed solution is used as the new starting point for the next round of the local search procedure.

If the search returns to the attractor C, BLS perturbs C more strongly by increasing the number of moves to be applied for perturbation. After visiting a certain number of local optima without improving the best solution found so far, BLS applies a significantly stronger perturbation in order to drive definitively the search toward a new and more distant region in the search space.

The success of the described method depends crucially on two 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 2.2. Algorithm 1 presents the BLS algorithm, whose components are detailed in the following sections. Each time a local optimum C is reached (lines 9-16, Sections 2.1.1 and 2.1.2), the jump magnitude L is determined depending on whether the search escaped or returned to the previous local optimum, and whether the search is stagnating in a non-promising region (lines 23-34, Section 2.2). BLS then applies L perturbation moves to C in order to get a new starting point for the local search procedure (line 37).

Algorithm 1 Breakout Local Search for the Maximum Clique Problem

Require: Graph $G = (V, E)$, initial and maximal jump magnitude L_0 and L_{Max}, max. number T of non-improving attractors visited before strong perturb., coefficients α_r and α_s for random and strong random perturbations.
Ensure: The largest clique (for MCP) or the largest weight clique (for MWCP).
1: $C \leftarrow generate\_initial\_solution(G)$
2: Create initial PA, OM and OC sets /* Section 2.1.2 */
3: $f_c \leftarrow f(C)$ /* f_c records the objective value of the solution */
4: $C_{best} \leftarrow C$ /* C_{best} records the best solution found so far */
5: $f_{best} \leftarrow f_c$ /* f_{best} records the best objective value reached so far */
6: $C_p \leftarrow C$ /* C_p records the last local optimum */
7: $\omega \leftarrow 0$ /* Set counter for consecutive non-improving local optima */
8: **while** stopping condition not reached **do**
9: Select the best move m from the set of moves formed by the union $M_1 \cup M_2$ /* Section 2.1.2 */
10: **while** $f(C \oplus m) > f_c$ **do**
11: $C \leftarrow C \oplus m$ /* Perform the best-improving move */
12: $f_c \leftarrow f(C \oplus m)$
13: Update PA, OM and OC vertex sets
14: $TL \leftarrow update\_tabu\_list(m, Iter)$ /* Section 2.2.2 */
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 best solution found so far */
19: $\omega \leftarrow 0$ /* Reset the counter of consecutive non-improving local optima */
20: **else**
21: $\omega \leftarrow \omega + 1$
22: **end if**
23: /* Determine the perturbation strength L to be applied to C */
24: **if** $\omega > T$ **then**
25: /* Search seems to be stagnating, strong perturbation required */
26: $L \leftarrow L_{Max}$
27: $\omega \leftarrow 0$
28: **else if** $C = C_p$ **then**
29: /* Search returned to the previous local optimum, increment perturbation strength */
30: $L \leftarrow L + 1$
31: **else**
32: /* Search escaped from the previous local optimum, reinitialize perturbation strength */
33: $L \leftarrow L_0$
34: **end if**
35: /* Perturb the current local optimum C with perturbation strength L */
36: $C_p \leftarrow C$
37: $C \leftarrow Perturbation(C, L, TL, Iter, \omega, \alpha_r, \alpha_s)$ /* Section 2.2.2 */
38: **end while**

2.1.1 Initial solution

The initial solution C used by BLS is generated in the following way. Select uniformly at random a vertex $v \in V$ and place it into C. While there exists a vertex $u \in V \setminus C$ such that $\forall c \in C, \{u, c\} \in E$, add u to C. This procedure stops when no vertex can be added to C, giving a valid clique.

2.1.2 The neighbourhood relations and its exploration

For solution transformations, BLS employs four distinct move operators (moves for short) whose basic idea is to generate a new clique from the current clique C by adding vertices $v \in V \setminus C$ to C, swapping vertices u and v such that $u \in C$ and $v \in V \setminus C$, or removing vertices $v \in C$ from C.

Three sets PA, OM and OC are involved in the definition of these moves. The vertex set PA consists of nodes excluded from the clique C that are connected to all the vertices in C, i.e., $PA = \{v : v \notin C, \forall u \in C, \{v, u\} \in E\}$.

The OM set consists of vertex pairs (v, u) such that v is excluded from C and is connected to all vertices in C except to vertex $u \in C$, i.e.,
$OM = \{(v, u) : v \notin C \text{ and } u \in C, |N(v) \cap C| = |C| - 1, \{v, u\} \notin E\}$, where $N(v) = \{i : i \in V, \{i, v\} \in E\}$.

The OC set consists of all the vertices excluded from the clique C, i.e., $OC = \{v : V \setminus C\}$.

The four moves M_1 to M_4 can then be defined as follows:

M_1: Select a vertex $v \in PA$ and insert it into C. After this move, the change in the objective function is given by the following expression: $\Delta = w_v$.

M_2: Select a vertex pair $(v, u) \in OM$. Insert v into C and remove u from C. The change in the objective function can be computed as: $\Delta = w_v - w_u$.

M_3: Select a vertex $v \in C$ and remove it from C. The change in the objective function is given as: $\Delta = -w_v$.

M_4: Select a vertex $v \in OC$ such that $(w_v + \sum_{\{v,u\} \in E, u \in C} w_u) \geq \alpha * f(C)$, where $f(C)$ is the current solution cost and $0 < \alpha < 1$. Add v to C. Repair the resulting clique C by removing from C all vertices x such that $\{v, x\} \notin E$.

During the local search phase, BLS explores the union of the moves M_1 and M_2. In other words, each iteration of the local search consists in identifying the best move m from $M_1 \cup M_2$ and applying it to C to obtain a new solution (line 11 in Algorithm 1, denoted by $C \leftarrow C \oplus m$ at line). This process is repeated until a local optimum is reached (see lines 9–15 of Algo. 1). The directed perturbation of BLS (see Section 2.2.2) applies a move m from $M_1 \cup M_2 \cup M_3$.

For random and strong random perturbation that acts as a restart (see Section 2.2.2), m is selected from M_4 with possibly different values for coefficient α (denoted by α_r for random and α_s for strong random perturbation).

2.2 Adaptive perturbation mechanism

2.2.1 General principle

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 search. The pseudo-code of this adaptive perturbation-based diversification procedure is given in Algorithms 2 and 3.

The idea of BLS is to first explore neighboring attractors, and move to a new distant area only if the search seems to be stagnating. Therefore, after each local search phase, BLS performs most of the time a weak perturbation (by applying a small number L of moves) that is hopefully just strong enough to escape the current basin of attraction and to fall under the influence of a neighboring local optimum. If the jump was not sufficient to escape the current attractor, the perturbation strength L is incremented and the perturbation is applied again to the current attractor (see lines 28–30 of Alg. 1). A strong perturbation (with a large number L of moves) is carried out only after visiting a certain number T of attractors without any improvement on the quality of the best solution found (see lines 23–27 of Alg. 1).

In addition to the number of moves to be applied for each perturbation, we also determine the type of moves. In the next section, we describe in details the different ways to select moves for perturbation.

2.2.2 The perturbation strategies

BLS employs both random and directed perturbations to guide the search towards new regions of the search space.

The *directed perturbation* is based on the idea of tabu list from tabu search. It uses a selection rule that favors moves that minimize the cost degradation, under the constraint that they are not prohibited by the tabu list. Move prohibition is determined in the following way. Each time a vertex v is placed into the clique C, it can be removed from C without restrictions. However, each time v is dropped from C, it is forbidden to place it back to C for γ iterations. The value of γ is determined by the following relation:

Algorithm 2 Perturbation procedure $Perturbation(C, L, TL, Iter, \omega, \alpha_r, \alpha_s)$

Require: Local optimum C, perturbation strength L, tabu list TL, global iteration counter $Iter$, number of consecutive non-improving local optima visited ω, coefficients α_r and α_s for random and strong random perturbations.

Ensure: A perturbed solution C.

1: **if** $\omega = 0$ **then**
2: /* *Best sol. not improved after a certain num. of visited local opt.* */
3: $C \leftarrow Perturb(C, L, M_4)$ /* *Strong random perturb. with moves from set* M_4 *when* $\alpha = \alpha_s$*, see Sect. 2.1.2 for the definition of set* M_4 */
4: **else**
5: Determine probability P according to Formula (1)
6: With probability P, $C \leftarrow Perburb(C, L, A)$
/* *Directed perturbation with moves from set A, see this section for the definition of set A* */
7: With probability $(1 - P)$, $C \leftarrow Perturb(C, L, M_4)$
/* *Rand. perturb. with moves from set* M_4 *when* $\alpha = \alpha_r$*, see Sect. 2.1.2 for the definition of set* M_4 */
8: **end if**
9: *Return C*

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

Require: Local optimum C, perturbation strength L, tabu list TL, global iteration counter $Iter$, the set of perturbation moves M.

Ensure: A perturbed solution π.

1: **for** $i := 1$ to L **do**
2: Take move $m \in M$
3: $C \leftarrow C \oplus m$ /* *Apply move* m *to* C */
4: $TL \leftarrow update\_tabu\_list(m, Iter)$ /* *Section 2.2.2* */
5: Update the PA, OM, OC vertex sets
6: $Iter \leftarrow Iter + 1$
7: **end for**
8: *Return C*

$$\gamma = \phi + random(|OM|),$$

where ϕ is a coefficient and *random* is a function which returns at random a value ranging from 1 to $|OM|$ (the number of elements in the OM set, see Section 2.1.2).

The information for move prohibition is maintained in the tabu list TL where the i^{th} element in TL is the iteration number when vertex i was dropped from a clique C. 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. The *directed perturbation* relies 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 directed perturbation

are identified by the set A such that:

$$A = \{m | m \in \{M_1 \cup M_2 \cup M_3\}, max\{\Delta_m\}, prohibited(m) = false \text{ or } (\Delta_m + f(C)) > f_{best}\}$$

where Δ_m is the change in the objective function after performing move m (see Section 2.1.2). Note that the directed perturbation considers all the eligible moves from the union of three types of moves M_1, M_2 and M_3 (see Section 2.1.2).

The random perturbation, which is significantly stronger than the directed perturbation, consists in performing moves randomly selected from the set of moves M_4 (see Section 2.1.2). The degree of random perturbation can be adjusted by changing the value of parameter α ($0 < \alpha < 1$). If $\alpha \approx 0$, the random perturbation is very strong and can be compared to a random restart. If $\alpha \approx 1$, the strength of random perturbation is insignificant.

As soon as a search stagnation is detected, i.e., the best found solution has not been improved after visiting a certain number of local optima, BLS applies the random perturbation (lines 1–4 of Algo. 2) in order to drive the search towards distant regions of the search space. Otherwise, BLS takes turns probabilistically between these two types of perturbations (lines 4–9 of Algo. 2). The probability of applying a particular perturbation is determined depending on the search state, i.e., the current number of consecutive non-improving attractors visited ω. The idea is to apply the directed perturbation with a higher probability at the beginning of the search, i.e., whenever the search progresses towards improved new local optima (counter ω is small). With the increase of ω, the probability of using the directed perturbation 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 useful to guarantee a minimum of applications of the directed perturbation. Therefore, we constrain the probability P of applying the directed perturbation to take values no smaller than a threshold P_0:

$$P = \begin{cases} e^{-\omega/T} & \text{if } e^{-\omega/T} > P_0 \\ P_0 & \text{otherwise} \end{cases} \quad (1)$$

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

2.3 Discussion

The general BLS procedure inherits some features from the two well-established metaheuristics: iterated local search [16] and tabu search [11]. We briefly discuss the similarities and differences between our BLS approach and these two methods.

Like ILS, BLS uses local search to discover local optima and perturbation to diversify the search. BLS distinguishes itself from ILS by the two types of perturbations that are triggered according to the search status, leading to variable levels of diversification. Moreover, a locally optimal solution returned by the local search procedure is always accepted as the new starting solution for BLS no matter its quality, which completely eliminates the acceptance criterion component of iterated local search.

The directed perturbation of BLS is based on the notion of tabu list that is borrowed from tabu search. However, BLS does not consider the tabu list during its local search 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 respectiv search, leading to different local optima. 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 maximum clique and the maximum weight clique problems.

3 Experimental results

3.1 Benchmark instances

We perform an extensive experimental evaluation of the proposed algorithm on both the maximum clique and the maximum weight clique problems. For MCP, we conduct experiments on the following two sets of benchmark instances:

DIMACS benchmark: These instances [1], presented at the Second DIMACS Implementation Challenge, are the most frequently used for comparison and evaluation of MC algorithms. More specifically, we use the following popular families:

- *Brock* – instances where the optimal clique is "hidden" by incorporating low-degree vertices.

[1] http://cs.hbg.psu.edu/txn131/clique.html

- *pHat* – random instances having a wider range of vertex degrees. The generator for these instances is a generalization of the classical uniform random graph generator.
- *MANN* – clique formulation of the Steiner Triple Problem. These instances are created using Mannino's code to convert the set covering formulation of the Steiner Triple Problem to clique problems.
- *Kel* – instances based on Keller's conjecture on tilings using hypercubes.
- *Ham* and *Joh* – graphs steaming from the coding theory.
- *Gen* – random instances with a unique known optimal solution.
- Other families such as *C*, *San*, *SanR* which are randomly generated using different methods.

The size of the DIMACS instances ranges from less than 50 vertices and 1,000 edges up to more than 3,300 vertices and 5,000,000 edges.

BOSHLIB benchmark: Although the DIMACS benchmark is still the most popular for evaluating maximum clique algorithms, we also consider the more recent frb test instances [2] arising from the SAT'04 Competition. BOSHLIB instances are randomly generated graphs with hidden optimal solutions and appear to be more difficult than most of the instances from the DIMACS suite. The instance size ranges from 450 vertices and 17,794 edges up to 1,534 vertices and 12,7011 edges.

For the less studied maximum weight clique problem, we use the DIMACS-W benchmark which is easily converted from the DIMACS instances by allocating weights to vertices in the following way. For vertex i, w_i is set equal to i *mod 200 + 1*. This method for transforming DIMACS instances to weighted DIMACS-W instances was initially proposed in [24]. In order to evaluate the proposed approach on more test problems with different characteristics, we create a set of additional instances by transforming the unweighted (and more recent) BOSHLIB instances to the weighted instances (that we call BOSHLIB-W). This is realized in the same way as for the DIMACS instances by applying the above described weighting function from [24]. As such, these BOSHLIB-W instances can easily be reproduced and used to test new weighted clique algorithms. Notice that other maximum weight clique algorithms from the literature [2,18] use randomly generated graphs with randomly generated vertex weights, which makes it impossible to replicate exactly the benchmark experiments for comparisons.

[2] http://www.nlsde.buaa.edu.cn/∼kexu/benchmarks/graph-benchmarks.htm

3.2 Experimental protocol

Our BLS algorithm is programmed in C++, and compiled with GNU gcc on a Xeon E5440 with 2.83 GHz and 2 GB. Following the DIMACS machine benchmark [3], 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. In the following, all CPU times are scaled with respect to the reference machine. For all tested instances, we execute our BLS approach 100 times. To increase the chances for reaching the best-known/optimal solution, we set the maximum number of iterations per run to 16×10^7 for both problems. For the unweighted clique problem, BLS terminates as soon as the best-known result from the literature is attained. This stopping condition is also used by several other state-of-art maximum clique approaches. However, the early termination criterion does not apply for the weighted clique problem since the weighted case is much less studied than the unweighted case and thus allows some space for improvement. Given the different structures of the used benchmark instances (see Section 4.2) and two separate problems, we run BLS under three different parameter settings to reach its peak performance. Table 1 gives the parameter settings, determined by a preliminary experimentation, for the unweighted problem (on the DIMACS and BOSHLIB benchmarks) and the weighted problem (on the DIMACS-W and BOSHLIB-W benchmarks). In the case of the unweighted clique problem, we use the parameter settings presented in column 'Settings 1' for instances from families *Brock* and $MANN$. The setting of parameters for all the other MC instances is given in column 'Settings 2'. In the case of MWCP, we use the parameter settings from column 'Settings 2' for the BOSHLIB-W instances, and the parameter settings from column 'Settings 3' for the DIMACS-W instances.

Note that the directed and random perturbations are combined as indicated in Section 2.2.2 when the first and third parameter settings are used. On the other hand, random perturbation is disabled (except during strong perturbation) in the case of the second parameter settings by setting the value of P_0 to 1. This obviously decreases the amount of diversification which generally improves BLS performance for some hard unweighted instances. In Section 4.3, we try to provide an explanation to this behaviour.

To evaluate the performance of BLS, we compare it with several state-of-art algorithms using the following criteria from the literature:

(1) The best solution obtained over a given number of runs;
(2) The average solution over a given number of runs;
(3) The number of instances for which an optimal or best-known solution is

[3] dmclique, ftp://dimacs.rutgers.edu in directory /pub/dsj/clique

Table 1
Settings of important parameters for MCP and MWCP

Para.	*Description*	*Settings 1*	*Settings 2*	*Settings 3*
L_0	initial jump magnitude	$0.01 * \|V\|$	$0.1 * \|V\|$	4
T	max. number of non-improving attractors visited before strong perturb. (restart)	1000	1000	1000
L_{Max}	maximal jump magnitude for restart	$0.1 * \|V\|$	$0.1 * \|V\|$	4
α_s	coefficient α for strong pertur. (restart)	0.8	0.8	0.7
ϕ	coefficient for tabu tenure	7	7	7
P_0	smallest probability for applying directed perturb.	0.75	1	0.75
α_r	coefficient α for random pertur.	0.8	–	0.92

reached within a reasonable computing time.

Beside the aforementioned criteria, the scaled average computing times needed to reach the best-known solution from the literature are also mentioned for indicative purposes.

It should be noted that a comparative analysis of the results found in the literature on state-of-art clique algorithms is not a straight-forward task because of the differences in computing hardware, programming language, termination criterion, etc. The comparisons in the following sections are thus presented only for indicative purposes and should be interpreted with caution. Nevertheless, our experimental study shows to some extend the performance of the proposed BLS algorithm relative to these state-of-the-art approaches.

3.3 Comparative results for MCP

To evaluate the performance of BLS on the unweighted maximum clique problem, we compare it with the following algorithms that achieve state-of-art performances:

(1) Reactive Local Search (RLS) [5]– an advanced tabu search method which is complemented by a feedback scheme to determine the amount of diversification;
(2) Variable Neighbourhood Search (VNS) [12] – a basic variable neighorhood search heuristic that combines greedy with the simplicial vertex test in its descent step;
(3) Phased Local Search (PLS) [23]– a stohastic local search algorithm which alterantes between phases of iterative improvement and plateau search.

Tables 2 and 3 show performance results for BLS in comparison with the results reported by the three reference approaches on the set of DIMACS instances. Column 'BR' gives the optimal clique size (indicated by '*') or the best-known clique size reported in the literature. For each approach, column

'Clique size' shows the best result obtained with the given algorithm over 100 runs, followed by the average and worse clique size values indicated in parentheses. For indicative purposes, we further provide the success rate (column 'Success') of BLS for attaining the best-known result from the literature. From the results in Tables 2 and 3, we can make the following observations.

For the instances from the *Brock* family, BLS reaches the best reported results from the literature with a perfect success rate for all the instances except in three cases (*brock*800_1, *brock*800_2, and *brock*800_3) where the success rate for reaching the best-known result is 70%, 68% and 84% respectively. These *Brock* instances seem to be particularly difficult for RLS and VNS approaches which fail to attain the best-known clique for the largest *Brock* instances (i.e., *brock*800_x). On these instances, the PLS algorithm attains the best-known solution in every single trial, and requires considerably less time than BLS.

For the instances from the $MANN$ family, RLS and VNS show better performance compared to BLS and PLS. Particularly, VNS is able to attain the best-known clique for both large instances $MANN\_a45$ ($|C| = 345$) and $MANN\_a81$ ($|C| = 1100$) within short computing time. BLS shows inferior performance, compared to the three reference approaches, on the two largest $MANN$ instances and is able to attain a maximum clique size of only 342 and 1094 for *MANN_a45* and *MANN_a81* respectively.

For the instances from the Kel family, BLS and RLS are equally effective and attain in every single trial the best reported clique within reasonable time. The two other reference approaches (VNS and PNS) are also able to reach the best-known solution for all the *keller* instances. However, compared to BLS and RLS, they show worse performance on the largest instance *keller*6 in terms of the average result.

For the instances from the family C, BLS attains the best-known clique in every single trial for all the instances except for instance $C2000.9$ where the success rate for reaching the best-known result is 10%. It should be noted that $C2000.9$ is probably the hardest DIMACS instance and only few algorithms from the literature [30] are able to obtain the clique size of 80 for $C2000.9$.

For the instances of other families (i.e., *Gen, Ham, Joh, pHat, San and SanR*), BLS is able to reach the best-known clique size for all the instances in every single trial within a time which is often less than a second. PLS also matches the performance of BLS on these instances. The performance of RLE and VNS has not been reported for a number of instances from these families. However, on some tested instances, VNS and RLS perform as good as BLS and PLS.

In summary, BLS reaches the best-known result for 68 out of the 70 DIMACS instances, while PLS is able to obtain the best-known solution for 67 of these instances. Out of the 34 used DIMACS instance, RLS attains the best-known

result for 30 instance while VNS reaches the best-known result for 31 of these instances. These results imply that BLS is quite competitive with the three reference approaches on the DIMACS instances and reaches performance comparable to the latest maximum clique algorithm described in [30].

Table 4 reports results of our BLS approach in comparison with those obtained with the PLS algorithm on the set of instances from the BOSHLIB collection. The two other reference approaches, RLS and VNS, are excluded from this comparison since their performance on BOSHLIB instances is not reported. From these results, we can make the following conclusions.

For small to medium size BOSHLIB instances (i.e., *frb30-X* to *frb45-X*) with up to 945 vertices, both BLS and PLS attain the optimal result in every single trial. For large instance (i.e., *frb50-X* to *frb59-X*), BLS attains the optimal clique value for every instance while PLS is unable to reach the optimal solution in two cases (i.e., *frb59-26-1* and *frb59-26-2*). Moreover, BLS reports better average results than PLS on these large BOSHLIB instances except in three cases where BLS provides a slightly worse average result. As for the computing time, BLS requires on average from 0.08 to 2399.92 seconds to reach the best-known results from the literature, while PLS needs on average from 0.03 to 1259.03 seconds to attain its best results. Therefore, on the BOSHLIB benchmark, PLS is more effective than BLS in terms of computing time, while BLS shows to be more effective in terms of solution quality.

3.4 Comparative results for MWCP

This section is dedicated to an evaluation of BLS performance on the maximum weight clique problem using the DIMACS-W and BOSHLIB-W benchmarks. For this purpose, we perform a comparison between BLS and the leading MWCP algorithm reported in [24] on the DIMACS-W instances.

Table 5 and 6 report the comparative results on the DIMACS-W benchmark. For the BLS approach, column 'Clique weight' shows the largest, average and smallest weight clique obtained over 100 executions. In [24], the author reports only the largest weight clique obtained by PLS over 100 runs (indicated in column 'Best clq.'). For each algorithm, column 'Success' provides the number of times the reported largest weight clique is obtained, while column 'CPU(s)' is the run-time in seconds averaged over all successful trials. Finally, the last column $\Delta_{(BLS-PLS)}$ provides the difference between the objective values (weight of the clique) of the best solutions attained by BLS and PLS respectively.

From the results in tables 5 and 6, we observe that BLS improves the best results of PLS for 14 out of the 70 DIMACS-W instances. Only in two cases, the best solution attained by BLS is worse than that obtained by PLS. Moreover,

Table 2. Comparative results of BLS with some of the best-performing maximum clique algorithms on the set of DIMACS instances. CPU(s) is the run-time in seconds averaged over all successful trials (i.e., trials when a best-known solution is reached).

		BLS			RLS [5]		VNS [12]		PLS [23]	
Instance	*BR*	*Success*	*Cliquesize*	*CPU(s)*	*Cliquesize*	*CPU(s)*	*Cliquesize*	*CPU(s)*	*Cliquesize*	*CPU(s)*
brock200_1	21⋆	100	21	0.01	–	–	–	–	21	0.00
brock200_2	12⋆	100	12	0.18	12	0.51	12(11.3;11)	0.05	12	0.01
brock200_3	15⋆	100	15	0.57	–	–	–	–	15	0.01
brock200_4	17⋆	100	17	0.43	17	1.03	17(16.9;16)	0.33	17	0.03
brock400_1	27⋆	100	27	121.4	–	–	–	–	27	0.11
brock400_2	29⋆	100	29	17.4	29(26.1;25)	4.58	29(27.4;25)	0.33	29	0.04
brock400_3	31⋆	100	31	5.08	–	–	–	–	31	0.02
brock400_4	33⋆	100	33	3.17	33(32.4; 25)	11.83	33	1.8	33	0.01
brock800_1	23⋆	70	23(22.4;21)	1568.24	–	–	–	–	23	11.25
brock800_2	24⋆	68	24(23.04;21)	1078.13	21	–	21	–	24	9.12
brock800_3	25⋆	84	25(24.52;22)	1020.11	–	–	–	–	25	5.63
brock800_4	26⋆	100	26	601.74	21	–	21	–	26	2.44
gen200_p0.9_44	44⋆	100	44	0.00	44	0.00	44	0.04	44	0.00
gen200_p0.9_55	55⋆	100	55	0.00	55	0.00	55	0.01	55	0.00
gen400_p0.9_55	55	100	55	2.21	55	0.13	55(54.8;54)	1.68	55	0.02
gen400_p0.9_65	65	100	65	0.00	65	0.01	65	0.06	65	0.00
gen400_p0.9_75	75	100	75	0.00	75	0.01	75	0.05	75	0.00
hamming6-2	32⋆	100	32	0.00	–	–	–	–	32	0.00
hamming6-4	4⋆	100	4	0.00	–	–	–	–	4	0.00
hamming8-2	128⋆	100	128	0.00	–	–	–	–	128	0.00
hamming8-4	16⋆	100	16	0.00	16	0.00	16	0.00	16	0.00
hamming10-2	512⋆	100	512	0.02	–	–	–	–	512	0.00
hamming10-4	40	100	40	0.02	40	0.00	40	0.01	40	0.00
MANN_a27	126⋆	100	126	35.2	126	0.34	126	0.00	126	0.00
MANN_a45	345⋆	0	342(340.82;340)	–	345(343.6; 343)	37.64	345(344.5; 344)	1.02	344	–
MANN_a81	1100	0	1094(1092.17; 1091)	–	1098	–	1100(1099.3;1098)	43.87	1098	–
keller4	11⋆	100	11	0.00	11	0.00	11	0.00	11	0.00
keller5	27	100	27	0.09	27	0.02	27	0.02	27	0.01
keller6	59	100	59	24.8	59	17.92	59(58.2;57)	12.0	59(57.75;57)	205.9
johnson8-2-4	4⋆	100	4	0.00	–	–	–	–	4	0.00
johnson8-4-4	14⋆	100	14	0.00	–	–	–	–	14	0.00
johnson16-2-4	8⋆	100	8	0.00	–	–	–	–	8	0.00
johnson32-2-4	16⋆	100	16	0.00	–	–	–	–	16	0.00
p_hat300-1	8⋆	100	8	0.00	8	0.00	8	0.00	8	0.00
p_hat300-2	25⋆	100	25	0.00	25	0.00	25	0.00	25	0.00
p_hat300-3	36⋆	100	36	0.00	36	0.00	36	0.00	36	0.00

Table 3. Comparative results of BLS with some of the best-performing maximum clique algorithms on the set of DIMACS instances. CPU(s) is the run-time in seconds averaged over all successful trials (i.e.,trials when a best-known solution is reached).

		BLS			RLS [5]		VNS [12]		PLS [23]	
Instance	*BR*	*Success*	*Cliquesize*	*CPU(s)*	*Cliquesize*	*CPU(s)*	*Cliquesize*	*CPU(s)*	*Cliquesize*	*CPU(s)*
p_hat500-1	9⋆	100	9	0.00	–	–	–	–	9	0.00
p_hat500-2	36⋆	100	36	0.00	–	–	–	–	36	0.00
p_hat500-3	50	100	50	0.01	–	–	–	–	50	0.00
p_hat700-1	11⋆	100	11	0.02	11	0.02	11	0.03	11	0.00
p_hat700-2	44⋆	100	44	0.00	44	0.00	44	0.00	44	0.00
p_hat700-3	62	100	62	0.00	62	0.00	62	0.00	62	0.00
p_hat1000-1	10	100	10	0.00	–	–	–	–	10	0.00
p_hat1000-2	46	100	46	0.01	–	–	–	–	46	0.00
p_hat1000-3	68	100	68	0.04	–	–	–	–	68	0.01
p_hat1500-1	12⋆	100	12	4.12	12	2.86	12	20.33	12	1.22
p_hat1500-2	65	100	65	0.03	65	0.01	65	0.01	65	0.00
p_hat1500-3	94	100	94	0.03	94	0.02	94	0.03	94	0.01
san1000	15⋆	100	15	95.33	–	–	–	–	15	1.76
san200_0.7_1	30⋆	100	30	0.05	–	–	–	–	30	0.00
san200_0.7_2	18⋆	100	18	0.43	–	–	–	–	18	0.01
san200_0.9_1	70⋆	100	70	0.00	–	–	–	–	70	0.00
san200_0.9_2	60⋆	100	60	0.01	–	–	–	–	60	0.00
san200_0.9_3	44⋆	100	44	0.01	–	–	–	–	44	0.00
san400_0.5_1	13⋆	100	13	15.07	–	–	–	–	13	0.01
san400_0.7_1	40⋆	100	40	61.05	–	–	–	–	40	0.01
san400_0.7_2	30⋆	100	30	2.65	–	–	–	–	30	0.01
san400_0.7_3	22⋆	100	22	1.1	–	–	–	–	22	0.02
san400_0.9_1	100⋆	100	100	0.04	–	–	–	–	100	0.00
sanr200_0.7	18⋆	100	18	0.01	–	–	–	–	18	0.00
sanr200_0.9	42⋆	100	42	0.01	–	–	–	–	42	0.00
sanr400_0.5	13⋆	100	13	0.05	–	–	–	–	13	0.00
sanr400_0.7	21	100	21	0.01	–	–	–	–	21	0.00
C125.9	34⋆	100	34	0.00	34	0.00	34	0.00	34	0.00
C250.9	44⋆	100	44	0.00	44	0.00	44	0.01	44	0.00
C500.9	57	100	57	0.00	57	0.29	57	0.18	57	0.07
C1000.9	68	100	68	35.7	68	3.93	68	3.47	68	0.70
C2000.5	16	100	16	2.9	16	0.94	16	0.94	16	0.27
C2000.9	80	1	80(78.6;78)	4811.17	78(77.6; 77)	–	78(77.2; 76)	–	78	–
C4000.5	18	100	18	654.6	18	206.07	18	208.19	18	55.94

Table 4. Comparative results of BLS with PLS on the whole set of BOSHLIB instances. Column 'BK' shows the value of the optimal clique. Column 'Clique size' indicates the value of the best clique obtained over 100 trials, followed by the average and worst clique value given in parenthesis. CPU(s) is the run-time in seconds averaged over all successful trials (i.e., trials when a best-known solution is reached).

Instance		BLS			PLS		Instance		BLS			PLS	
Name	$\|C\|$	*Success*	*Clique size*	$CPU(s)$	*Clique size*	$CPU(s)$	*Name*	$\|C\|$	*Success*	*Clique size*	$CPU(s)$	*Clique size*	$CPU(s)$
frb30-15-1	30⋆	100	30	0.10	30	0.05	frb50-23-1	50⋆	96	50(49.96; 49)	882.22	50(49.72; 49)	771.03
frb30-15-2	30⋆	100	30	0.13	30	0.07	frb50-23-2	50⋆	56	50(49.56; 49)	1074.17.21	50(49.45; 49)	863.62
frb30-15-3	30⋆	100	30	1.49	30	0.05	frb50-23-3	50⋆	8	50(49.08; 49)	1037.68	50(49.16; 49)	767.94
frb30-15-4	30⋆	100	30	0.08	30	0.03	frb50-23-4	50⋆	100	50	55.51	50	93.24
frb30-15-5	30⋆	100	30	1.13	30	0.24	frb50-23-5	50⋆	100	50	142.93	50(49.99; 49)	321.73
frb35-17-1	35⋆	100	35	8.14	35	3.07	frb53-24-1	53⋆	4	53(52.04; 52)	2306.74	53(52.06; 52)	1259.02
frb35-17-2	35⋆	100	35	2.25	35	0.91	frb53-24-2	53⋆	16	53(52.16; 52)	2015.13	53(52.23; 52)	1142.17
frb35-17-3	35⋆	100	35	0.37	35	0.19	frb53-24-3	53⋆	88	53(52.88; 52)	1199.95	53(52.66; 52)	874.34
frb35-17-4	35⋆	100	35	6.82	35	4.17	frb53-24-4	53⋆	54	53(52.54; 52)	1361.23	53(52.48; 52)	1049.54
frb35-17-5	35⋆	100	35	0.96	35	0.58	frb53-24-5	53⋆	90	53(52.9; 52)	1100.00	53(42.46; 52)	722.56
frb40-19-1	40⋆	100	40	1.45	40	2.45	frb56-25-1	56⋆	20	56(55.2; 55)	2304.83	56(55.1; 54)	914.54
frb40-19-2	40⋆	100	40	49.61	40	39.89	frb56-25-2	56⋆	6	56(55.06; 55)	1500.44	56(54.93; 54)	1255.37
frb40-19-3	40⋆	100	40	5.97	40	3.56	frb56-25-3	56⋆	21	56(55.2; 55)	1409.09	56(55.08; 55)	1088.87
frb40-19-4	40⋆	100	40	35.98	40	17.00	frb56-25-4	56⋆	86	56(55.86; 55)	999.51	56(55.66; 54)	961.61
frb40-19-5	40⋆	100	40	167.36	40	73.55	frb56-25-5	56⋆	100	56	591.49	56(55.81; 55)	803.10
frb45-21-1	45⋆	100	45	37.11	45	30.49	frb59-26-1	59⋆	1	59(57.96; 57)	3298.21	58(57.85; 57)	–
frb45-21-2	45⋆	100	45	83.68	45	60.92	frb59-26-2	59⋆	1	59(58.00; 57)	2399.92	58(57.63; 57)	–
frb45-21-3	45⋆	100	45	341.73	45	305.31	frb59-26-3	59⋆	31	59(58.31; 58)	2338.59	59(57.77; 57)	1422.51
frb45-21-4	45⋆	100	45	52.60	45	43.71	frb59-26-4	59⋆	20	59(58.2; 58)	1823.63	59(57.71; 57)	1507.95
frb45-21-5	45⋆	100	45	124.74	45	80.29	frb59-26-5	59⋆	100	59	403.30	59(58.77; 57)	879.90

BLS reports a better success rate (average result) than PLS for another 12 instances. Only in one case, BLS has a lower success rate. As for the CPU time, BLS requires on average from 0 up to 2942.54 seconds to attain its best solution. The maximum run-time for PLS is not reported in [24] since, for some instances, the normal CPU time allowed for each trial prevented PLS from reaching its best result from column 'Best clq'.

Table 7 shows the performance of BLS on the BOSHLIB-W instances. Given that the BOSHLIB-W instances can easily be reproduced, our results on these instances are useful to compare other weighted methods.

4 Analysis and discussion

It is well known that the performance of any heuristic algorithm depends on the balance between intensification and diversification. However, the desired balance is influenced by the landscape structure such as the distribution of local optima, the correlation between solutions, the number of global optima, etc. In this section, we wish to obtain some insight on the search space of MC and MWC problem instances in order to understand the behaviour of our BLS algorithm. For this purpose, we analyse the distribution of local optima and perform a fitness distance analysis (FDA) which investigates correlation between the quality (fitness) of local optima and their distances to the optimum [13]. Moreover, we study the impact of different diversification degrees on the performance of the proposed approach.

4.1 Analysis protocol

We perform a landscape analysis for both MCP and MWCP on 35 selected instances. The results reported for each instance are based on a set of distinct solutions obtained by 4000 independent runs of BLS. To measure distance between solutions, we use the hamming distance which is the minimum number of substitutions required to change one solution into another. If the optimal solution is unknown for an instance, we use the best-known local optima to compute fitness-distance correlation and refer to them as global optima. We take into account for this analysis that some instances have multiple global optima.

Table 5
Comparative results of BLS and PLS on the set of DIMACS-W instances. Column 'Clique weight' shows the best, average and worst result obtained over 100 executions by BLS; column 'Best clq.' shows the best clique weight value obtained with PLS over 100 trials; 'Success' provides the number of times the reported maximum weighted clique is obtained; 'CPU(s)' is the run-time in seconds averaged over all successful trial; column $\Delta_{(BLS-PLS)}$ provides the difference between the weight values of the best solutions attained by BLS and PLS respectively.

	BLS			PLS			
Name	*Success*	*Clique weight*	$CPU(s)$	*Success*	*Best clq.*	$CPU(s)$	$\Delta_{(BLS-PLS)}$
brock200_1	100	2821	0.00	100	2821	0.14	0
brock200_2	100	1428	0.03	100	1428	0.02	0
brock200_3	100	2062	0.01	100	2062	0.01	0
brock200_4	100	2107	0.01	100	2107	0.52	0
brock400_1	100	3422	0.05	32	3422	321.66	0
brock400_2	100	3350	0.08	61	3350	306.04	0
brock400_3	100	3471	0.26	100	3471	8.86	0
brock400_4	100	3626	7.60	100	3626	0.04	0
brock800_1	100	3121	0.13	100	3121	23.20	0
brock800_2	100	3043	0.51	69	3043	658.82	0
brock800_3	100	3076	0.50	100	3073	2.47	3
brock800_4	100	2971	339.07	100	2971	2.78	0
gen200_p0.9_44	100	5043	0.01	100	5043	3.29	0
gen200_p0.9_55	100	5416	1.75	100	5416	0.04	0
gen400_p0.9_55	100	6718	0.18	2	6718	250.24	0
gen400_p0.9_65	100	6940	0.05	4	6935	147.73	5
gen400_p0.9_75	100	8006	0.43	100	8006	0.00	0
hamming6-2	100	1072	0.00	100	1072	0.00	0
hamming6-4	100	134	0.00	100	134	0.00	0
hamming8-2	100	10976	0.12	100	10976	0.00	0
hamming8-4	100	1472	0.00	100	1472	0.00	0
hamming10-2	100	50512	6.64	100	50512	0.00	0
hamming10-4	100	5129	26.86	1	5086	1056.77	43
MANN_a27	16	12281(12276.9; 12273)	396.58	–	12264	–	17
MANN_a45	1	34229(34211.3; 34201)	929.41	–	34129	–	100
MANN_a81	1	111237(111188; 111162)	2942.54	–	110564	–	673
keller4	100	1153	0.04	100	1153	0.02	0
keller5	100	3317	0.65	100	3317	87.93	0
keller6	44	8062(8027.2; 7923)	1980.16	–	7382	–	680
johnson8-2-4	100	66	0.00	100	66	0.00	0
johnson8-4-4	100	511	0.00	100	511	0.00	0
johnson16-2-4	100	548	0.01	100	548	0.00	0
johnson32-2-4	100	2033	0.48	100	2033	32.95	0

Table 6
Continued

	BLS			PLS			
Name	*Success*	*Clique weight*	$CPU(s)$	*Success*	*Best clq.*	$CPU(s)$	$\Delta_{(BLS-PLS)}$
p_hat300-1	100	1057	0.01	100	1057	0.01	0
p_hat300-2	100	2487	0.02	100	2487	14.36	0
p_hat300-3	100	3774	0.01	47	3774	310.21	0
p_hat500-1	100	1231	0.04	100	1231	0.31	0
p_hat500-2	100	3920	0.01	–	3925	–	-5
p_hat500-3	100	5375	0.05	–	5361	–	14
p_hat700-1	100	1441	0.01	100	1441	0.15	0
p_hat700-2	100	5290	0.02	100	5290	57.89	0
p_hat700-3	100	7565	0.13	12	7565	529.76	0
p_hat1000-1	100	1514	0.07	100	1514	5.61	0
p_hat1000-2	100	5777	0.04	87	5777	693.63	0
p_hat1000-3	100	8111	0.41	–	7986	–	125
p_hat1500-1	100	1619	0.14	100	1619	36.07	0
p_hat1500-2	100	7360	0.18	4	7328	778.85	32
p_hat1500-3	100	10321	1.78	–	10014	–	307
san1000	100	1716	4.94	–	1716	–	0
san200_0.7_1	100	3370	30.65	100	3370	0.00	0
san200_0.7_2	100	2422	0.01	66	2422	294.83	0
san200_0.9_1	100	6825	23.68	100	6825	0.00	0
san200_0.9_2	100	6082	0.19	100	6082	0.00	0
san200_0.9_3	100	4748	0.02	72	4748	162.99	0
san400_0.5_1	100	1455	0.22	100	1455	147.47	0
san400_0.7_1	98	3641(3640.64; 3623)	–	100	3941	0.02	-300
san400_0.7_2	33	3110(3002.56; 2952)	166.00	100	3110	0.04	0
san400_0.7_3	100	2771	0.05	100	2771	3.24	0
san400_0.9_1	100	9776	6.25	100	9776	0.00	0
sanr200_0.7	100	2325	0.01	100	2325	0.46	0
sanr200_0.9	100	5126	0.00	5	5126	134.61	0
sanr400_0.5	100	1835	0.04	100	1835	0.49	0
sanr400_0.7	100	2992	0.03	100	2992	104.11	0
C125.9	100	2529	0.01	100	2529	5.99	0
C250.9	100	5092	0.06	17	5092	182.24	0
C500.9	100	6955	0.25	–	6822	–	133
C1000.9	100	9254	12.33	5	8965	254.22	289
C2000.5	100	2466	2.10	18	2466	524.50	0
C2000.9	74	10999(10989.9; 10964)	1152.78	–	10028	–	971
C4000.5	100	2792	179.89	–	2792	–	0

Table 7. Computational results of BLS on the set of BOSHLIB-W instances. Column 'Success' provides the number of times the reported maximum weight clique is obtained; 'Clique weight' shows the best, average and worst result obtained over 100 executions; 'CPU(s)' is the run-time in seconds averaged over all successful trial; column 'Avg. Iter' provides the number of BLS iterations required to reach the best solution averaged over all successful trial.

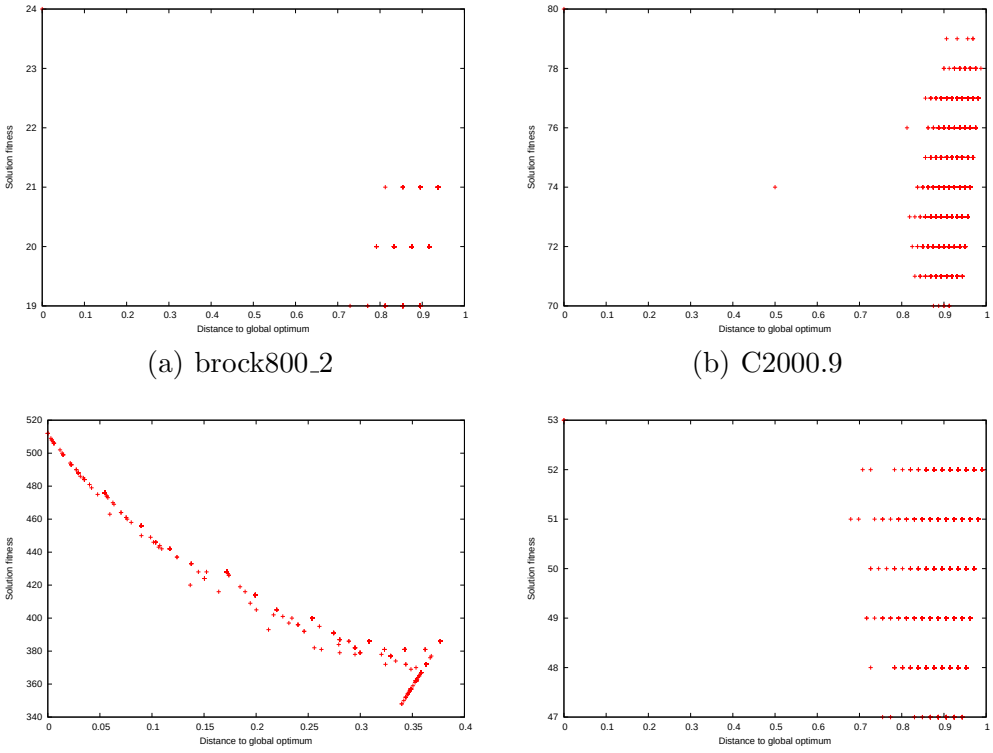
Name	*Success*	*Clique weight*	*CPU(s)*	*Avg. Iter*	*Name*	*Success*	*Clique weight*	*CPU(s)*	*Avg. Iter*
frb30-15-1	100	2990	1.12	150606	frb50-23-1	11	5494(5486.41; 5485)	1221.72	77739303
frb30-15-2	100	3006	8.15	1129585	frb50-23-2	5	5462(5440.22; 5426)	2837.74	129335094
frb30-15-3	100	2995	11.67	1564603	frb50-23-3	98	5486(5485.98; 5485)	537.96	35125348
frb30-15-4	100	3032	0.33	43693	frb50-23-4	14	5454(5453.14; 5453)	1190.43	76154594
frb30-15-5	100	3011	3.64	483863	frb50-23-5	100	5498	388.18	24983985
frb35-17-1	100	3650	68.45	8961529	frb53-24-1	13	5670(5652.18; 5640)	1056.82	64130771
frb35-17-2	100	3738	197.42	23867783	frb53-24-2	3	5707(5685.32; 5671)	147.65	8864390
frb35-17-3	100	3716	11.58	1451092	frb53-24-3	48	5640(5629.38; 5604)	984.53	62353262
frb35-17-4	100	3683	232.36	30995122	frb53-24-4	13	5714(5676.16; 5636)	1604.50	69504498
frb35-17-5	100	3686	20.00	2595520	frb53-24-5	4	5659(5642.5; 5627)	278.91	16400953
frb40-19-1	96	4063(4062.8; 4058)	291.14	31630105	frb56-25-1	5	5916(5860.82; 5834)	1764.87	77113037
frb40-19-2	100	4112	439.81	36351497	frb56-25-2	1	5886(5838.96; 5803)	1013.85	39694846
frb40-19-3	46	4115(4111.72; 4107)	778.75	76596837	frb56-25-3	1	5859(5811; 5778)	101.48	3684299
frb40-19-4	98	4136(4135.92; 4132)	333.89	32834888	frb56-25-4	12	5892(5860.86; 5834)	1256.9	70600583
frb40-19-5	88	4118(4117.52; 4114)	343.82	35297705	frb56-25-5	1	5853(5787.04; 5759)	4386.6	154764648
frb45-21-1	58	4760(4754.3; 4739)	982.32	82392194	frb59-26-1	17	6591(6571.6; 6548)	1435.99	72411248
frb45-21-2	100	4784	307.06	23928814	frb59-26-2	13	6645(6602.34; 6568)	1834.93	93343723
frb45-21-3	88	4765(4764.76; 4763)	641.03	53288518	frb59-26-3	1	6608(6542.74; 6502)	507.93	26106825
frb45-21-4	96	4799(4797.24; 4755)	576.80	49992432	frb59-26-4	6	6592(6526.5; 6493)	952.34	48756880
frb45-21-5	100	4779	206.60	16637346	frb59-26-5	5	6584(6546.94; 6527)	1512.09	77719959

Table 8
FDC analysis on 35 selected instances for the maximal clique and the maximal weighed clique problems. For each instance, we provide the number of distinct global optima found (column '$\#go$'), the size of the analysed sample (column '$\#lo$'), the FDC coefficient (column 'ρ'), the normalized average distance between local optima (column '$avg\ d_{lo}$'), and the normalized average distance between a local optimum and the nearest global optimum (column '$avg\ d_{go}$').

	Unweighted maximum clique problem					Weighted maximum clique problem				
Instance	$\#go$	$\#lo$	ρ	$avg\ d_{lo}$	$avg\ d_{go}$	$\#go$	$\#lo$	ρ	$avg\ d_{lo}$	$avg\ d_{go}$
brock800_1	1	2300	0.177	0.830	0.915	1	451	-0.242	0.703	0.735
brock800_2	1	2444	0.134	0.800	0.898	1	476	-0.003	0.681	0.686
brock800_3	1	2300	0.116	0.766	0.880	1	444	-0.150	0.647	0.676
brock800_4	1	2347	0.126	0.736	0.865	1	336	-0.065	0.625	0.814
MANN_a27a	483	4000	-0.657	0.660	0.415	3	1409	-0.175	0.395	0.413
keller5	2141	3945	-0.851	0.931	0.231	2	975	-0.350	0.806	0.816
keller6	30	3918	-0.069	0.878	0.860	1	3591	-0.294	0.778	0.813
C1000.9	22	4000	-0.298	0.849	0.740	1	1510	-0.365	0.654	0.657
C2000.5	1188	3311	-0.986	0.951	0.491	1	227	-0.044	0.837	0.834
C2000.9	1	4000	0.308	0.876	0.920	1	3072	-0.056	0.358	0.381
san1000	1	3795	0.608	0.577	0.782	1	120	-0.307	0.534	0.521
san400_0.7_2	1	2937	-0.003	0.543	0.736	1	44	-0.655	0.392	0.676
san400_0.7_3	1	2699	0.145	0.713	0.837	1	17	-0.747	0.523	0.499
hamming10-2	3	816	-0.977	0.481	0.227	1	3456	-0.013	0.461	0.492
hamming10-4	3466	3998	-0.976	0.954	0.085	8	1807	-0.070	0.788	0.651
gen400_p0.9_65	1	747	-0.761	0.602	0.700	1	132	-0.585	0.422	0.364
gen400_p0.9_75	1	311	-0.873	0.514	0.655	1	307	-0.826	0.400	0.435
p_hat1000-1	276	302	-0.978	0.946	0.053	1	4	-0.932	0.750	0.600
p_hat1000-2	482	523	-0.943	0.306	0.023	2	38	-0.778	0.435	0.303
p_hat1000-3	24	1186	-0.566	0.435	0.423	1	646	-0.353	0.408	0.385
p_hat1500-1	1	735	0.155	0.861	0.913	1	11	-0.332	0.719	0.633
p_hat1500-2	1169	1413	-0.965	0.253	0.046	1	196	-0.551	0.374	0.311
p_hat1500-3	1989	3152	-0.911	0.199	0.077	1	2280	-0.473	0.390	0.329
frb50-23-1	32	3866	-0.205	0.881	0.863	1	3047	-0.059	0.781	0.804
frb50-23-2	31	3913	-0.205	0.878	0.874	1	3205	0.072	0.782	0.805
frb50-23-3	2	3909	0.184	0.883	0.911	1	3088	-0.199	0.780	0.792
frb50-23-4	309	3548	-0.727	0.874	0.775	1	2937	-0.031	0.786	0.806
frb53-24-1	2	3997	0.135	0.882	0.907	1	3593	-0.100	0.805	0.814
frb53-24-2	10	3968	0.051	0.920	0.928	1	3518	-0.014	0.807	0.805
frb53-24-3	35	3993	-0.205	0.882	0.867	1	3637	-0.031	0.809	0.816
frb53-24-4	17	3961	-0.105	0.880	0.890	1	3838	0.024	0.805	0.818
frb56-25-1	6	3970	0.079	0.884	0.911	1	3977	-0.054	0.812	0.835
frb56-25-2	10	3983	0.059	0.883	0.913	1	4000	-0.130	0.807	0.802
frb56-25-3	5	3976	-0.007	0.886	0.902	1	3982	0.106	0.816	0.843
frb56-25-4	86	3977	-0.317	0.886	0.884	1	3972	0.039	0.810	0.819

4.2 FDC and distribution of local optima

The fitness-distance correlation (FDC) coefficient ρ [13] is a well-known tool for landscape analysis and can provide useful indications about the problem hardness, even if such an analysis has some known shortcomings and limitations. The FDC captures the correlation between the fitness (i.e., quality) of a solution and its distance to the nearest global optimum (or best-known solution if no global optimum is available). For a maximization problem, a value of $\rho = -1$ indicates a perfect correlation between fitness and distance to the optimum, implying that improving the fitness reduces the distance to the global optimum. For landscapes with $-0.15 < \rho < 0.15$, there is virtually no correlation between fitness and distance, while for correlation of $\rho = 1$, there is no correlation at all. In this case, using the fitness to guide the search towards global optimum may be misleading. The FDC can also be visualized with a fitness-distance (FD) plot, where the same data used for estimating ρ is displayed graphically. Such plots have been used to estimate the distribution of local optima for a number of problems including for instance the TSP problem [8], graph partitioning problem [7,19], flow-shop scheduling problem [25], and the QAP [29].



(a) brock800_2 (b) C2000.9

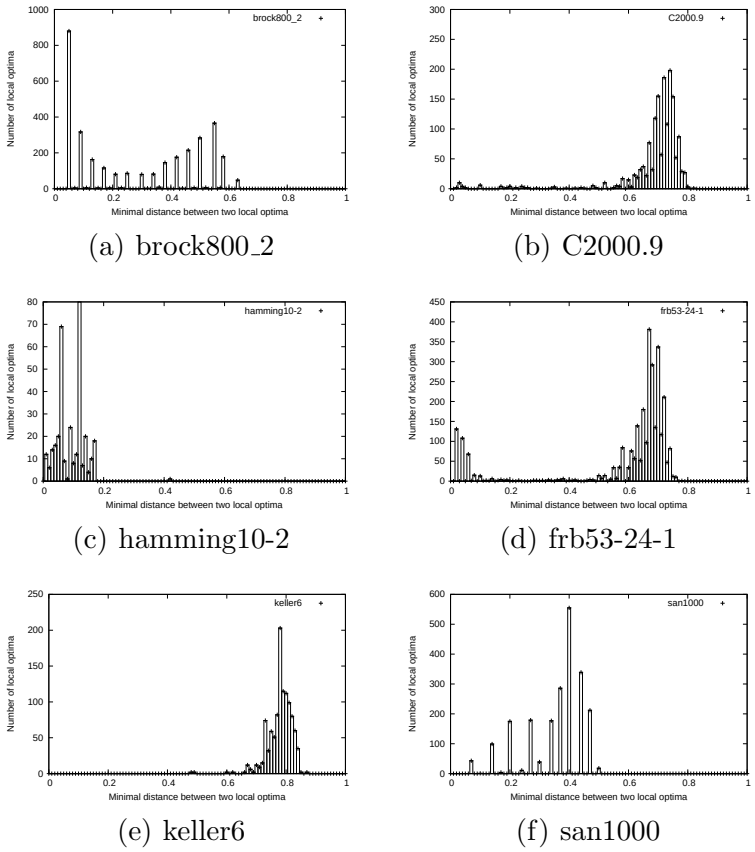
(c) hamming10-2 (d) frb53-24-1

Fig. 1. Distance of local optima to the best-known solution based on solutions sampled by BLS algorithm for four unweighted maximum clique instances.

Table 8 reports the results of our FDC analysis on the 35 selected instances for the maximum clique (left part) and the maximum weight clique (right part) problems. For each instance, we collect 4000 solutions and indicate the number of distinct global optima found (column '$\#go$'), the size of the analysed sample (i.e., the number of distinct local optima, column '$\#lo$'), the FDC coefficient (column 'ρ'), the average distance between local optima (column '$avg\ d_{lo}$'), and the average distance between a local optimum and the nearest global optimum (column '$avg\ d_{go}$'). For the sake of clarity, we normalize the distances $avg\ d_{lo}$ and $avg\ d_{go}$ by dividing them by the maximal possible distance. For illustrative purposes, FD plots for four instances of the unweighted maximum clique problem (*brock800_2, C2000.9, hamming10-2* and *frb53-24-1*) are given in Figure 1.

From the results for the unweighted maximum clique problem, we observe that there are significant differences in landscapes of problem instances. For some MCP instances, our search detected only one global optimum, while for other instances there may exist even thousands of distinct globally optimal solutions. Moreover, some landscapes have highly correlated local optima (i.e., $\rho < -0.15$) meaning that the fitness (evaluation function) provides a good guidance for the search. Indeed, the FD plot in Figure 1c indicates an almost perfect correlation for *hamming10-2* whose ρ coefficient is very close to -1. On the other hand, some MCP instances (e.g., *brock800_X, C2000.9, frb53-24-1*) have highly uncorrelated landscapes (i.e., $\rho > 0.15$), implying that the search may be difficult for many state-of-art approaches. From the plots in figures 1a, 1b, 1d, it is clear that there is no correlation for instances *brock800_2, C2000.9,* and *frb53-24-1*. This explains why these instances are particularly hard for many algorithms.

From the results in Table 8, it can also be observed that the values of $avg\ d_{lo}$ and $avg\ d_{go}$ are often very large (i.e., > 0.8) in the case of MCP. To gain an even better insight in the distribution of local optima in the search space, we investigate the minimal distance between pairs of medium or high quality local optima. These solutions may be viewed as 'strong' attractors since it is more likely that they may be visited during the search than a low quality local optimum. The results of this study for 6 unweighted maximum clique instances are given in Figure 2. The x-axis shows the normalized minimal distance between a pair of 'strong' attractors, while the y-axis shows the number of pairs of 'strong' attractors separated by the given distance. Figure 2 indicates that there exists a significant difference in the distribution of medium and high quality local optima for MC instances. For *brock800_2, hamming10-2* and *san1000*, the minimal distance between pairs of 'strong' attractors is generally significantly smaller than in the case of *C2000.9* and *keller6*. Intuitively, a weaker diversification introduced into the search for such instances may cause the search to cycle between 'strong' attractors that are not globally optimal solutions. For an effective solving of these instances, strong diversifications are



(a) brock800_2 (b) C2000.9

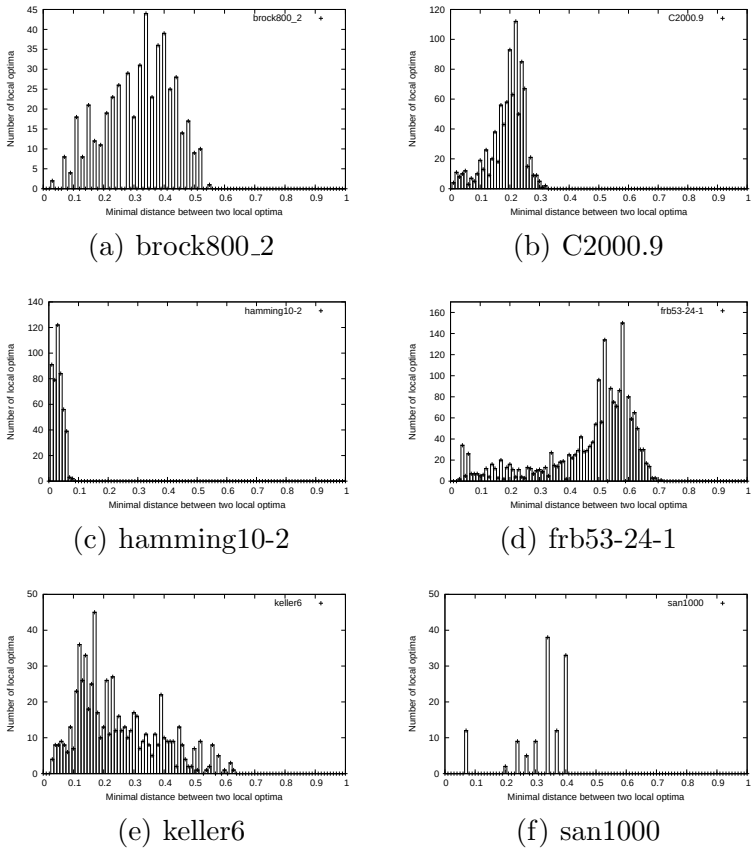
(c) hamming10-2 (d) frb53-24-1

(e) keller6 (f) san1000

Fig. 2. Distribution of medium and high quality local optima (i.e., 'strong' attractors) for 6 maximum clique instances. The x-axis shows the normalized minimal distance between pairs of 'strong' attractors, while the y-axis shows the number of pairs of 'strong' attractors separated by the given distance.

needed.

The analytical results from Table 8 for the maximum weight clique problem indicate that the MWCP instances also exhibit different degrees of landscape correlation. For some problems (e.g., *Brock800_X, C2000.9, san1000*), we note that local optima are much better correlated than in the case of the unweighted clique problem. However, the landscapes of *frb* instances are rather uncorrelated. We further observe that the distances *avg* d_{lo} and *avg* d_{go} are often significant, but generally slightly smaller than in the case of MCP. In Figure 3, we study the minimal distance between pairs of medium or high quality local optima for 6 weighted maximum clique instances. As in the case of MCP, we observe that for some instances (e.g., *brock800_2, C2000.9, hamming10-2, keller6*), the minimal distance between pairs of 'strong' attractors is generally smaller than for other instances (e.g., *frb53-24-1*).



(a) brock800_2 (b) C2000.9

(c) hamming10-2 (d) frb53-24-1

(e) keller6 (f) san1000

Fig. 3. Distribution of medium and high quality local optima (i.e., 'strong' attractors) for 6 maximum weight clique instances. The x-axis shows the normalized minimal distance between pairs of 'strong' attractors, while the y-axis shows the number of pairs of 'strong' attractors separated by the given distance.

4.3 Influence of different diversification degrees to BLS performance

In this section, we investigate the impact of different diversification degrees and try to provide some explanations based on the observations made from the landscape analysis of Section 4.2. Because of space limitations, we focus this study only on the more popular unweighted maximum clique problem.

In Figure 4, we show the performance of BLS on 6 maximum clique instances over 50 independent executions when four different degrees of diversification ($d1 - d4$) are introduced into the search. The diversification $d1$ is the strongest one used for this experiment. The parameter settings used to obtain this degree of diversification are provided in column 'Settings 1' of Table 1. The diversification $d2$ is the second strongest which uses the same parameter settings as indicated in column 'Settings 2' of Table 1, except for parameters P_0 and α_r that we set to 0.75 and 0.92 respectively. The diversification $d3$ uses the same parameter settings as in column 'Settings 2' of Table 1, while the

weakest diversification $d4$ is obtained with the parameter settings from column 'Settings 1' of Table 1 with the exception of $P_0 = 1$. Note that for $d1$ and $d2$, the directed and random perturbations are combined, while for $d3$ and $d4$, the random perturbation is disabled.

From Figure 4*a*, we observe that a stronger diversification is beneficial for instance *brock800_2* and that the best-known solution (i.e., $|C| = 23$) cannot be attained if a weaker diversification is used. Although we did not show plots for other instances of the *Brock* family, they generally show the same behaviour. As noted in Section 4.2, the minimal distance between pairs of 'strong' attractors for instance *brock800_2* is rather small for most attractor pairs. It is thus most likely for the search to cycle and waste time in non-promising regions if a weaker diversification is applied.

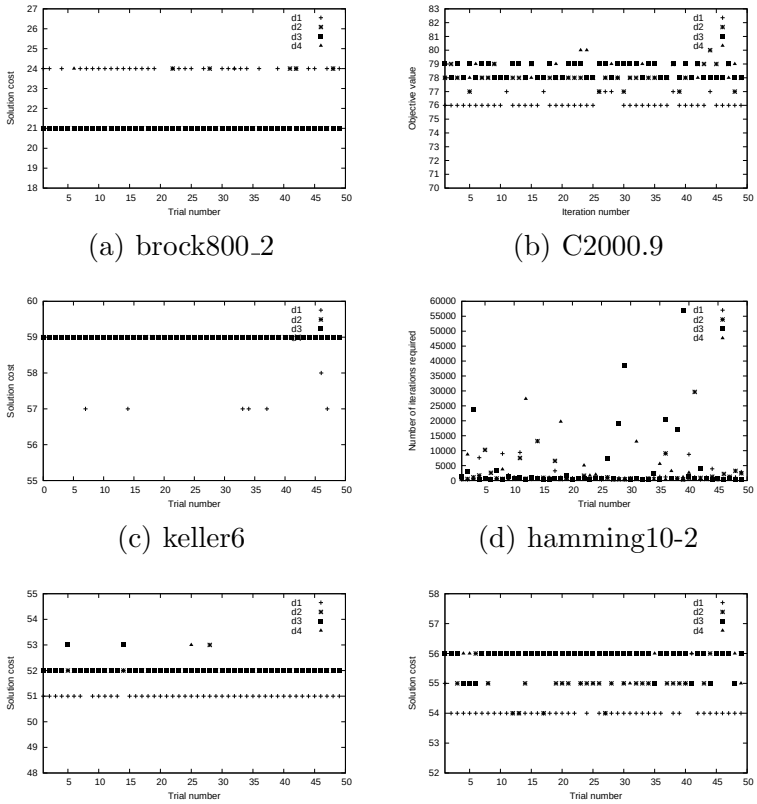
We note that the situation is quite opposite in the case of $C2000.9$. Indeed, BLS attained even twice over 50 runs the best-known clique size of 80 with the weakest diversification $d4$. From the analytical results, we observed that, unlike for the *Brock* instances, the minimal distance between pairs of 'strong' attractors is significantly larger for *C2000.9*, which avoids cycling even with a much weaker diversification.

For *keller6*, we note from plot 2*e* that the minimal distance between pairs of 'strong' attractors is large as in the case of *C2000.9*. This explains why BLS is unable to reach the best-known clique size ($|C| = 59$) in every single trial with the strongest diversification $d1$. Moreover, we observe that BLS performance on $keller6$ does not depend that much on different degrees of diversification as in the case of $C2000.9$. One possible explanation is that local optima of $C2000.9$ are much more uncorrelated than those of $keller6$. In other words, the value of the FDC coefficient ρ for $C2000.9$ (see Table 8) is significantly higher than for $keller6$ implying that $C2000.9$ is much harder for most algorithms.

Plot 4*d* shows the impact of different diversification degrees for *hamming10-2*. For this instance, BLS is able to reach the optimum regardless of the amount of diversification introduced into the search. In this case, we thus study the performance of BLS in terms of the number of iterations required to attain the optimum. From Figure 4*d*, we observe that the performance of BLS on *hamming10-2* is slightly better when stronger diversifications ($d1$ and $d2$) are used. This can partially be justified by the fact that the landscape of *hamming10-2* has a very large number of globally optimal solutions (over 3466 out of 4000 sampled solutions) compared to most instances. For such landscapes, greedy multi-start algorithms should provide excellent results.

For instances *frb53-24-1* and *frb56-25-4* from the BOSHLIB benchmark, plots 4*e* and 4*f* indicate that a weaker diversification is more useful than a stronger one. Indeed, the largest clique size for these two instances cannot be attained

by BLS if $d1$ is employed. From plot $2d$, we observe that the minimal distance between most pairs of 'strong' attractors is significant (i.e., < 0.5), while only a small number of attractor pairs are separated by a distance which is less than 0.1. The distribution of medium and high quality local optima for *frb* instances is thus very similar to that of *C2000.9* and *keller6*, which explains to some extent why a weaker diversification is more appropriate for *frb* instances.



(a) brock800_2 (b) C2000.9

(c) keller6 (d) hamming10-2

(e) frb53-24-1 (f) frb56-25-4

Fig. 4. Influence of different diversification degrees on the performance of BLS. Diversification $d1$ is the strongest, while $d4$ is the weakest.

5 Conclusion

In this paper, we presented the Breakout Local Search approach for the maximum clique (MC) and the maximum weight clique (MWC) problems. 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 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 wide set of benchmark instances showed that despite its simplicity, our approach competes very favourably with the current most effective (often more complex) algorithms for the maximum clique problem. In particular, BLS is able to reach the current best solution for all the benchmark instances form the DIMACS and BOSHLIB benchmark collections, except for 2 cases (*MANN_a45* and *MANN_a81*). Moreover, it provides new improved results for 14 MWCP instances from the DIMACS-W benchmark. Only rare approaches are able to attain such a performance on both MCP and MWCP without any particular adaptation.

Another contribution of this paper is a detailed landscape analysis that we performed on a number of selected instances for the MC and the MWC problems. The analysis revealed that there exist significant differences in problem landscapes for both problems. Based on the observations made from this analysis, we explained to some extent the behaviour of BLS when different diversification degrees are introduced into the search.

Acknowledgment

This work was partially supported by "Angers Loire Métropole" and the Region of "Pays de la Loire" within the MILES, RADAPOP and LigeRO Projects. Special thanks to Qinghua Wu for his constructive comments. We are also grateful to the anonymous referees for valuable suggestions and comments which helped us improve the paper.

References

[1]

[2] L. Babel. A fast algorithm for the maximum weight clique problem. *Computing*, 52: 31–38, 1994.

[3] D. Ballard, C. Brown. Computer vision. Prentice-Hall, Englewood Cliffs, 1982.

[4] E. Balus, C. Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal of Computing*, 15(4): 1054–1068, 1986.

[5] R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. *Algorithmica*, 29(4): 610–637, 2001.

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

[7] U. Benlic and J.K. Hao. A Multilevel Memetic Approach for Improving Graph k-Partitions. *IEEE Transactions on Evolutionary Computation*, 15(5): 624–642, 2011.

[8] K.D. Boese. Cost versus Distance in the Traveling Salesman Problem. Technical Report TR-950018, UCLA CS Department, 1995.

[9] I. Bomze, M. Budinich, M. Pelillo, and C. Rossi. Annealed replication: a new heuristic for the maximum clique problem. *Discrete Applied Mathematics*, 121:27–49, 2002.

[10] C. Friden, A. Hertz, and D. de Werra. Stabulus: A technique for finding stable sets in large graphs with tabu search. *Computing*, 42:3–44, 1989.

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

[12] P. Hansen, N. Mladenovic, and D. Urosevic. Variable neighborhood search for the maximum clique. *Discrete Applied Mathematics*, 145(1):117–125, 2004.

[13] T. Jones and S. Forrest. Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms. *Proceedings of the 6th International Conference on Genetic Algorithms*, Morgan Kaufmann, 184–192, 1995.

[14] R.M. Karp. Reducibility among combinatorial problems. *In R.E. Miller and J.W. Thatcher editors*, Complexity of Computer Computations, 85–103, Plenum Press, New York, 1972.

[15] K. Katayama, A. Hamamoto and H. Narihisa. An effective local search for the maximum clique problem. *Information Processing Letters*, 95: 503–511, 2005.

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

[17] E. Macambira and C. de Souza. The edge-weighted clique problem: valid inequalities, facets and polyhedral computations. *European Journal of Operations Research*, 123(2): 346–371, 2000.

[18] E. Macambira. An application of tabu search heuristic for the maximum edge-weighted subgraph problem. *Annals of Operations Research*, 117: 175–190, 2000.

[19] P. Merz and B. Freisleben. Fitness Landscapes, Memetic Algorithms and Greedy Operators for Graph Bi-Partitioning. *Evolutionary Computation*, 8(1): 61–91, 2000.

[20] P.R. Östergärd. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(13): 197–207, 2002.

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

[22] W. Pullan and H.H. Hoos. Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 25: 159–185, 2006.

[23] W. Pullan. Phased local search for the maximum clique problem. *Journal of Combinatorial Optimization*, 12: 303–323, 2006.

[24] W. Pullan. Approximating the maximum vertex/edge weighted clique using local search. *Journal of Heuristics*, 14: 117–134, 2008.

[25] C.R. Reeves. Landscapes, Operators and Heuristic Search. *Annals of Operations Research*, 86: 473–490, 1997.

[26] P.S. Segundo, D. Rodríguez-Losada, A. Jiménez. An exact bit-parallel algorithm for the maximum clique problem. *Computers & Operations Research*, 38(2):571–581, 2011.

[27] A. Singh and A.K. Gupta. A hybrid heuristic for the maximum clique problem. *Journal of Heuristics*, 12: 5–22, 2008.

[28] C. Solnon and S. Fenet. A study of ACO capabilities for solving the maximum clique problem. *Journal of Heuristics*, 12(3): 155–180, 2006.

[29] T. Stützle. Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174(3): 1519–1539, 2006.

[30] Q. Wu and J.K. Hao. An adaptive multistart tabu search approach to solve the maximum clique problem. To appear in *Journal of Combinatorial Optimization*, `http://dx.doi.org/10.1007/s10878-011-9437-8.`

[31] Q. Zhang, J. Sun and E. Tsang. EDA+GA: Evolutionary Algorithm with Guided Mutation for the Maximum Clique Problem. *IEEE Trans. on Evolutionary Computation*, 9(2):192–200, 2005.