Engineering Applications of Artificial Intelligence 37 (2015) 20–33

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

General swap-based multiple neighborhood tabu search for the maximum independent set problem

Yan Jin, Jin-Kao Hao *

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

ARTICLE INFO

Article history:
Received 11 April 2014
Received in revised form
2 July 2014
Accepted 12 August 2014
Available online 15 September 2014

Keywords:
Maximum independent set
Maximum clique
Multiple neighborhoods
Local search
Tabu search

ABSTRACT

Given a graph $G=(V,E)$, the Maximum Independent Set problem (MIS) aims to determine a subset $S \subseteq V$ of maximum cardinality such that no two vertices of S are adjacent. This paper presents a general Swap-Based Tabu Search (SBTS) for solving the MIS. SBTS integrates distinguished features including a general and unified $(k,1)$-swap operator, four constrained neighborhoods and specific rules for neighborhood exploration. Extensive evaluations on two popular benchmarks (DIMACS and BHOSLIB) of 120 instances show that SBTS attains the best-known results for *all* the instances. To our knowledge, such a performance was not reported in the literature for a single heuristic. The best-known results on 11 additional instances from code theory are also attained.

1. Introduction

Given a simple undirected graph $G=(V,E)$ with vertex set $V=\{v_1,\ldots,v_n\}$ and edge set $E \subset V \times V$. An independent set S is a subset of V such that no two vertices are adjacent, i.e., $\forall v_i, v_j \in S, \{v_i, v_j\} \notin E$. An independent set is said maximum if it has the largest cardinality among all the independent sets of G. The maximum independent set problem (MIS) is to determine a maximum independent set of an arbitrary graph. As one of Karp's 21 NP-complete problems (Karp, 1972), MIS is among the most popular problems in combinatorial optimization (Garey and Johnson, 1979; Johnson and Trick, 1996).

In graph theory, there are two tightly related problems: the maximum clique problem (MC) and minimum vertex cover problem (MVC). A clique C of G is a subset of V such that all vertices in C are pairwise adjacent, i.e., $\forall v_i, v_j \in C, \{v_i, v_j\} \in E$. MC is to find a clique C of maximum cardinality. A vertex cover VC of G is a subset of V such that each edge of E is incident to at least one vertex of VC, i.e., $\forall \{v_i, v_j\} \in E, v_i \in VC \vee v_j \in VC$. MVC is to determine a vertex cover of minimum cardinality.

Let $\overline{G}=(V,\overline{E})$ be the complementary graph of $G=(V,E)$ such that $\overline{E} \subset V \times V$ and $\forall v_i, v_j \in V, \{v_i, v_j\} \in \overline{E}$ if and only if $\{v_i, v_j\} \notin E$. Then given a subset S of V, the following three statements are equivalent (Wu and Hao, 2014): S is an independent set in G, $V \setminus S$ is a vertex cover in G and S is a clique in $\overline{G}$. As a consequence, MIS, MC and MVC are three equivalent problems such that any algorithm designed for one of these problems can be directly applied to solve the other two problems. These problems are relevant to a wide variety of applications such as code theory, information retrieval, signal transmission, classification theory, experimental design and many more others (Bomze et al., 1999; Johnson and Trick, 1996; Wu and Hao, 2014). In this work, we focus on studying the MIS problem.

During the past decades, a large number of solution procedures for solving MIS, MC and MVC have been reported in the literature. Among them are several exact algorithms based on the general branch-and-bound framework (Carraghan and Pardalos, 1990; Li and Quan, 2010; Östergård, 2002; San Segundo et al., 2011; Tomita and Kameda, 2007). These exact methods are applicable to problem instances of limited sizes. For larger cases, various heuristics have been proposed to obtain near-optimal solutions. The most representative heuristics include tabu search (Battiti and Protasi, 2001; Friden et al., 1989; Wu et al., 2012; Wu and Hao, 2013), stochastic local search (Andrade et al., 2012; Grosso et al., 2008; Katayama et al., 2005; Pullan, 2006, 2008), parallel hyper-heuristics mixing several low-level heuristics (Pullan et al., 2011), simulated annealing (Geng et al., 2007), variable neighborhood search (Hansen et al., 2004), breakout local search (Benlic and Hao, 2013), local search with edge weighting (Cai et al., 2013; Richter et al., 2007) and evolutionary algorithms (Brunato and Battiti, 2011; Zhang et al., 2005).). According to the reported results on benchmark instances, in particular those of the well-known Second DIMACS Implementation Challenge on Cliques, Coloring, and Satisfiability (Johnson and Trick, 1996), it seems that ILS and GLP

* Corresponding author.
E-mail addresses: jin@info.univ-angers.fr (Y. Jin), hao@info.univ-angers.fr (J.-K. Hao).

http://dx.doi.org/10.1016/j.engappai.2014.08.007
0952-1976/

(Andrade et al., 2012), BLS (Benlic and Hao, 2013), NuMVC (Cai et al., 2013), PLS (Pullan, 2006, 2008), CLS (Pullan et al., 2011), COVER (Richter et al., 2007), MN/TS (Wu et al., 2012) and AMTS (Wu and Hao, 2013) are among the top performing heuristics in the literature. Nevertheless, due to the large variety of structures of these instances (they are random graphs or transformed from different real problems), no single approach can attain the best-known results for all the DIMACS instances.

In this work, we introduce a general Swap-Based Tabu Search (SBTS) heuristic for the maximum independent set problem. SBTS inspects the search space by a dynamic alternation between intensification and diversification steps (Glover and Laguna, 1997; Lourenço et al., 2003; Schrimpf et al., 2000). The search process is driven by a general and unified (*k*,1)-swap ($k \geq 0$) operator combined with specific rules to explore four constrained neighborhoods. Given an independent set S, (*k*,1)-swap exchanges one vertex (which is strategically selected) in $V \setminus S$ against its k adjacent vertices in S. For the purpose of intensification, SBTS uses (0,1)-swap to improve the solution and (1,1)-swap to make sidewalks with the help of specific selection rules. To overcome local optima, SBTS adopts an adaptive perturbation strategy which applies either a (2,1)-swap for a weak perturbation or a (k,1)-swap ($k > 2$) for a strong perturbation. A tabu mechanism is also employed to prevent the search from short-term cycles. Compared with existing local search algorithms, SBTS distinguishes itself by its unified $(k,1)-$swap operator, its specific neighborhoods and its dedicated rules for neighborhood exploration.

The proposed SBTS algorithm attains the best-known results for all 120 instances of the well-known DIMACS and BHOSLIB benchmarks with very different structures and topologies. This is the first time a single heuristic reaches such a performance. The best-known results are also attained on an additional set of 11 real instances from code theory.

The rest of the paper is structured as follows. Section 2 describes the SBTS approach. Section 3 shows computational results and comparisons with the state-of-the-art algorithms in the literature. Before concluding, Section 4 investigates and analyzes some important issue of the proposed algorithm.

2. A swap-based tabu search for MIS

Our Swap-Based Tabu Search (SBTS) algorithm for MIS follows the iterated local search framework (Lourenço et al., 2003) and shares similarities with other methods like variable neighborhood search (Hansen et al., 2004) and the ruin-and-recreate search (Schrimpf et al., 2000). However, as we explain in this section, SBTS possesses some particular features like four constrained neighborhoods and the specific rules for an effective exploration of these neighborhoods.

2.1. General procedure

The general SBTS procedure is summarized in Algorithm 1. SBTS uses a fast randomized construction procedure (Section 2.3) to obtain a first feasible independent set S (i.e., no two vertices of S are adjacent, S is also called *a feasible solution* or simply *a solution* in the paper). From this initial solution, SBTS tries to find improved solutions (i.e., larger independent sets) by a series of intensification and diversification steps (Sections 2.7 and 2.8). Both intensification and diversification steps are based on the general $(k,1)-$swap operator (Section 2.5).

Specifically, each intensification step makes a $(k,1)-$swap move ($k = 0, 1$) to increase the cardinality of the independent set or search new solutions while keeping the cardinality unchanged. Inversely, a diversification step applies a $(k,1)-$swap move ($k \geq 2$) to decrease temporarily the quality of the current solution (the current solution loses $k-1$ vertices). Whenever there exist intensification moves, they are always preferred over diversification moves. Diversification moves are only applied to escape from a local optimum (i.e., when no eligible $(k,1)-$swap move ($k = 0, 1$) is available). As we explain in Sections 2.7 and 2.8, both intensification and diversification are subject to dedicated rules which govern the way $(k,1)-$swap moves are executed.

SBTS uses a global variable S_* to record the best solution ever discovered during the search and a tabu list to prevent short-term cycles (see Section 2.6). The algorithm stops when a fixed number of iterations are realized.

Algorithm 1. General procedure of the SBTS algorithm for MIS.

```
1:  Input: A graph G, Iters_max (maximum allowed iterations
        per run)
2:  Output: The largest independent set S_* found.
3:  S ← Initialization()  /* Generate a feasible independent set
    S, Sect. 2.3 */
4:  S_* ← S  /* S_* records the largest independent set found so
    far */
5:  f_* ← f(S)  /* f_* records the cardinality of S_* */
6:  Initialize tabu_list  /* Initialize the tabu list, Section 2.6 */
7:  for iters ← 1 to Iters_max do
8:    if there exists an eligible intensification move then
9:      S ← IntensificationStep(S)  /* Apply (k,1)−swap
        (k ≤ 1) to improve solution S, Section 2.7 */
10:     if f(S) > f_* then
11:       S_* ← S, f_* ← f(S)
12:     end if
13:   else
14:     S ← DiversificationStep(S)  /* Apply (k,1)−swap
      (k > 1) to perturb solution S, Section 2.8*/
15:   end if
16:   Update tabu_list /* Section 2.6 */
17: end for
18: return S_*
```

2.2. Search space and evaluation function

Before presenting the components of the SBTS algorithm, we define first the search space Ω explored by the algorithm as well as its evaluation function f to measure the quality of a candidate solution.

For a given graph $G = (V, E)$, the search space Ω explored by SBTS is the set of all the independent sets of G, i.e., $\Omega = \{S \subseteq V : v_i, v_j \in S, \{v_i, v_j\} \notin E\}$. For any feasible solution $S \in \Omega$, its quality is directly assessed by the cardinality of S, i.e., $f(S) = |S|$. Given two independent sets S and S', S is better than S' if and only if $f(S) > f(S')$.

2.3. Initial solution

The initial solution used by the SBTS algorithm is generated by the following sequential randomized heuristic (V is the vertex set of graph G).

1. Set S to empty.
2. Select randomly a vertex $u \in V$ and add u into S.
3. Remove from V vertex u and all its adjacent vertices $v \in V$ ($\{u, v\} \in E$).
4. Repeat steps (2)–(3) until V becomes empty and return S.

It is easy to observe that the resulting solution S is a feasible (and maximal) independent set. Due to the random choices at

step (2), each run of this construction procedure may lead to a different solution. Given the stochastic nature of SBTS, this feature is useful for multiple runs of SBTS.

2.4. Preliminary definitions

To explain the intensification and diversification mechanisms of SBTS, we introduce some key concepts (measures) which are particularly useful to define the different neighborhoods and the application rules of the general $(k,1)-$swap operator.

Definition 1 (*Mapping Degree* K^M). Given a graph $G=(V,E)$ and an independent set S, the Mapping Degree of a vertex v_i in $V \setminus S$ is the number of its adjacent vertices v_j in S, i.e., $\forall v_i \in V \setminus S$, $K^M(v_i) = |\{v_j \in S : \{v_i, v_j\} \in E\}|$. A similar definition of Mapping Degree can be found in Andrade et al. (2012).

Fig. 1 shows a graph G with 10 vertices and an independent set $S=\{1,4,6,8\}$ and $V \setminus S=\{2,3,5,7,9,10\}$. According to the definition, vertex 2 in $V \setminus S$ has one adjacent vertex (1) in S, hence the Mapping Degree $K^M(2)=1$. Similarly, the Mapping Degrees of the other vertices in $V \setminus S$ are shown in Table 1. The Mapping Degree is used to partition the vertices of $V \setminus S$ into four subsets which define the neighborhoods used by SBTS (see Section 2.5).

Definition 2. (Expanding Degree K^E) The Expanding Degree of a vertex v_i in S is the number of its adjacent vertices v_j in $V \setminus S$ whose Mapping Degree K^M equals to 1, i.e., $\forall v_i \in S, K^E = |\{v_j \in V \setminus S : \{v_i, v_j\} \in E, K^M(v_j)=1\}|$.

In Fig. 1, among the 4 vertices of S, only vertices 1 and 4 have adjacent neighbors in $V \setminus S$ with a Mapping Degree of 1, thus their Expanding Degree is $K^E(1)=1$ and $K^E(4)=2$ while vertices 6 and 8 have zero Expanding Degree (see Table 1). The Expanding Degree is used to define the rule to exploit the neighborhood induced by (1,1)-swap (see Section 2.7).

Definition 3 (*Diversifying Degree* K^D). Given a graph $G=(V,E)$ and an independent set S, the Diversifying Degree of a vertex v_i in $V \setminus S$ is the number of adjacent vertices v_j in $V \setminus S$, i.e. $\forall v_i \in V \setminus S$, $K^D(v_i) = |\{v_j \in V \setminus S : \{v_i, v_j\} \in E\}|$.

The Diversifying Degree is used to differentiate vertices with the same Expanding Degrees when the neighborhood induced by (1,1)-swap is exploited (see Section 2.7). It is also employed to define the rule to select degrading, i.e., $(k,1)-$swap $(k>1)$ moves to escape from local optima (see Section 2.8).

For the details of our example shown in Fig. 1, see Table 1. As shown in the Appendix, these measures will be dynamically updated after each iteration of the algorithm and this can be achieved efficiently in an incremental way.

2.5. $(k,1)-$swap, *neighborhoods and exploration of neighborhoods*

The search process of the SBTS algorithm is basically driven by the general $(k,1)-$swap $(k=0,1,2,\ldots)$ operator. In this section, we provide a detailed presentation of this operator, the different neighborhoods induced by the operator and the way these neighborhoods are explored.

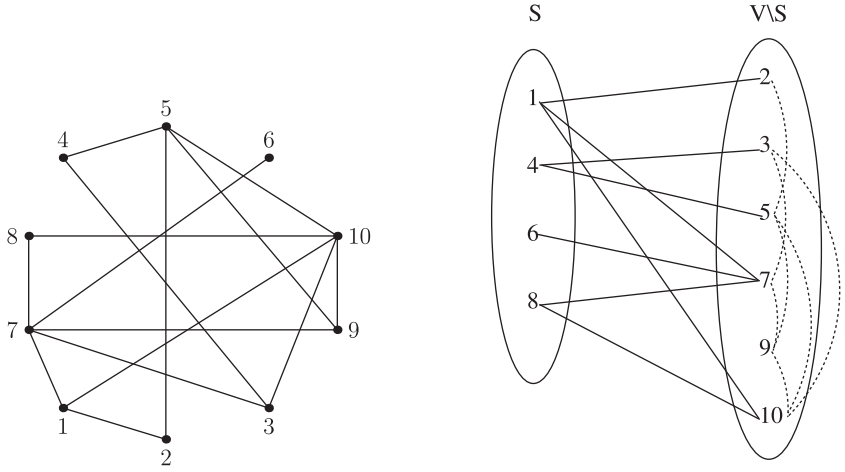

Fig. 1. An illustrative example of graph G.

Table 1
Mapping degree, expanding degree and diversifying degree on the illustrative graph.

Vertex in $\lvert V \setminus S \rvert$	Neighbors in S	Mapping Degree K^M	Neighbors in $V \setminus S$	Diversifying Degree K^D
2	1	1	5	1
3	4	1	7, 10	2
5	4	1	2, 9, 10	3
7	1, 6, 8	3	3, 9	2
9		0	5, 7, 10	3
10	1, 8	2	3, 5, 9	3

Vertex in $\lvert S \rvert$	Neighbors in $V \setminus S$ whose Mapping Degree $K^M=1$	Expanding Degree K^E
1	2	1
4	3, 5	2

Let S be an independent set and $V \setminus S$ its complementary set. Let "$S \oplus (k,1)$−swap" denote the application of $(k,1)$−swap ($k \geq 0$) to S. Then the resulting solution S' is given by $S' \leftarrow S \oplus (k,1)$−swap.

According to the value of k, $(k,1)$−swap changes differently the cardinality of the current solution S. The resulting solution has a larger or equal cardinality when $k \leq 1$ (i.e., $k = 0, 1$). Otherwise (i.e., $k \geq 2$), the resulting solution is deteriorated by $k-1$ units. Moreover, whatever the value k takes, applying a $(k,1)$−swap move to an independent set always leads to a feasible solution.

To define the rules to apply this general $(k,1)$−swap operator, we introduce four different neighborhoods. Precisely, given an independent set S (the current solution), we partition its complementary set $V \setminus S$ into four subsets according to the Mapping Degree of each vertex (see Section 2.4).

1. NS_k: the set of vertices v_i in $V \setminus S$ whose Mapping Degree K^M equals to k, i.e., $NS_k = \{v_i \in V \setminus S : K^M(v_i) = k\}, (k = 0, 1, 2)$.
2. $NS_{>2}$: the set of vertices v_i in $V \setminus S$ whose Mapping Degree K^M is larger than 2, i.e., $NS_{>2} = \{v_i \in V \setminus S : K^M(v_i) > 2\}$.

For the example of Fig. 1 where $S = \{1, 4, 6, 8\}$ and $V \setminus S = \{2, 3, 5, 7, 9, 10\}$, we have $NS_0 = \{9\}, NS_1 = \{2, 3, 5\}, NS_2 = \{10\}$ and $NS_{>2} = \{7\}$.

Clearly, each NS_k $(k = 0, 1, 2, >2)$ set defines unambiguously a different (and constrained) neighborhood when it is employed by the $(k,1)$−swap operator. Precisely, for a given NS_k, the associated neighborhood is composed of all the solutions obtained by swapping a vertex of NS_k with its k adjacent vertices in S. For this reason, we will also use NS_k $(k = 0, 1, 2, >2)$ to denote the associated neighborhoods interchangeably.

To explore the search space, the SBTS algorithm selects at each iteration a particular vertex from one NS_k $(k = 0, 1, 2, >2)$ as follows. SBTS first examines NS_0 to see whether an improving (0,1)-swap move is applicable. If NS_0 is not empty, a (0,1)-swap move is applied with a vertex randomly chosen from NS_0. Otherwise, if NS_1 offers eligible vertices, a side-walk (1,1)-swap move is applied to a vertex of NS_1 which is selected according to the specific rule presented in Section 2.7. If NS_1 is empty or all the vertices of NS1 are forbidden by the tabu list, SBTS makes a degrading $(k,1)$−swap move with a vertex from NS_2 or $NS_{>2}$ following the rule defined in Section 2.8. Hence, at each iteration, SBTS only checks a smaller number (i.e., $|NS_k|$ instead of $|V \setminus S|$) of neighboring solutions to explore the search space. According to the value of k, each iteration corresponds to either an intensification step (k=0,1) or a diversification step ($k \geq 2$). After each iteration, K^M, K^E, K^D, along with the neighborhoods NS_k and their sizes are updated accordingly (see Appendix).

One understands intuitively that during the search process, NS_0 will be exhausted very rapidly. Among the remaining NS_k $(k = 1, 2, >2)$ sets, NS_k with $k \geq 2$ can only deteriorate the solution and should not be used frequently. The only set that could lead to a hopeful improvement of the solution is NS_1. In Sections 2.7 and 2.8, we introduce dedicated rules for the exploration of the neighborhoods NS_k with $k \geq 1$.

2.6. Tabu list and aspiration rule

The SBTS algorithm uses a tabu list to avoid short-term cycles (Glover and Laguna, 1997). Precisely, each time a $(k,1)$−swap move is executed, the k vertices which are swapped out from the independent set S are classified tabu in order to prevent these vertices from moving back to S for the next tt iterations (tt is called tabu tenure). On the other hand, the vertex which joins S is not subject to tabu prohibition. Thus the tabu list is updated only after a $(k,1)$−swap with $k \geq 1$.

Suppose $(k,1)$−swap exchanges vertex $v_i \in NS_k$ $(k \geq 1)$ and its k adjacent vertices $(v_{j_1}, v_{j_2} \cdots v_{j_k})$ in S. For each vertex v_{j_p} $(p = 1 \cdots k)$, its tabu tenure tt is adaptively set as follows.

- k=1: If $|NS_1| < |NS_2| + |NS_{>2}|$, $tt = 10 + Random(|NS1|)$ where $Random(A)$ returns a random value from the domain $\{0 \cdots A-1\}$; Otherwise, $tt = |NS_1|$.
- $k > 1$: tt is set to 7.

These tabu tenure rules are purely empirical. However, for the case of k=1, the first part corresponds to situations which occur usually and the adopted tabu tenure ($tt = 10 + Random(|NS1|)$) is inspired from the literature (Dorne and Hao, 1998; Galinier and Hao, 1999; Wu and Hao, 2013). The second part (which occurs occasionally) is based on the consideration that when there are many side-walk moves (i.e., $|NS_1|$ is very high relative to $|NS_2|$ and $|NS_{>2}|$), the vertex that just left the solution will not be considered before having tried a number of side-walk moves as high as $|NS_1|$. For the case of $k > 1$, since there are several vertices (at least two) leaving the independent set and these k vertices are not chosen according to specific objectives, there is no reason to prevent them from joining the solutions for a long period of time. For this reason, the tabu tenure for them can be set to a relatively small value. In fact, we observe that as long as the tabu tenure remains in the range of 4 to 10, it does not really impact the performance of the algorithm. So we set the tabu tenure to the middle value 7 which proves to be robust in our experiments. In Section 4.2, we provide more information about the tabu tenure.

Notice that vertices in NS_0 are never forbidden by the tabu list given that any vertex in NS_0 can increase the current solution by one unit. This can be considered as an aspiration condition (Glover and Laguna, 1997) that revokes the tabu status of any vertex if it belongs to NS_0.

Finally, given an independent set S and the associated sets NS_k $(k = 1, 2, >2)$, a vertex from any NS_k is said *eligible* if it is not forbidden by the tabu list.

2.7. Intensification

Intensification of the SBTS algorithm aims to find better solutions or to reach new solutions without deteriorating the current solution. For this purpose, whenever NS_0 is not empty, SBTS applies $(0,1)$−swap to improve the solution. For instance, in Fig. 1, given the current solution $S = \{1, 4, 6, 8\}$, $NS_0 = \{9\}$ and $NS_1 = \{2, 3, 5\}$, SBTS will select vertex 9 to apply $(0,1)$−swap to generate a better solution $S = \{1, 4, 6, 8, 9\}$. When NS_0 becomes empty, SBTS checks then the NS_1 neighborhood for a possible (1,1)-swap (side-walk) move.

If NS_1 offers multiple choices for a (1,1)-swap, one must decide which vertex of NS_1 is selected for the (1,1)-swap. One trivial strategy is to make this decision at random. However, as we illustrate below, the order of examining the vertices in NS_1 for (1,1)-swap may impact the solution quality. To make this decision as fruitful as possible, we devise a selection rule which takes into account problem specific information relative to the Expanding Degree and Diversifying Degree (see Section 2.4). The proposed selection rule favors the (1,1)-swap moves that tend to create new promising (e.g., improving) moves for future iterations.

Selection Rule for the NS_1 neighborhood examination:

1. Collect in set NS_1^- any vertex $v_i \in NS_1$ such that its adjacent neighbor v_j in S has the largest Expanding Degree.
2. If NS_1^- is composed of a single vertex, select this vertex; otherwise, select the vertex $v_i \in NS_1^-$ with the largest Diversifying Degree (ties are broken at random).

The first part of this *Selection Rule* is based on the following consideration. When swapping $v_i \in NS_1$ with $v_j \in S$ such that v_j has the largest Expanding Degree, we encourage the emergence of improving (0,1)-swap moves. For instance, in Fig. 1, given $NS_1 = \{2, 3, 5\}$ and suppose that all the vertices in NS_1 are eligible for a (1,1)-swap move (the notion of eligibility under the tabu rule is explained in Section 2.6). Since vertices 3 and 5 of NS_1 have the adjacent vertex 4 in S with an Expanding Degree of 2 while vertex 2 of NS_1 has the adjacent vertex 1 in S with an Expanding Degree of 1, we have $NS_1^- = \{3, 5\}$ (i.e., vertices 3 and 5 are preferred than vertex 2). At this point, one notices a (1,1)-swap using any vertex 3 or 5 of NS_1^- (say 3) will change the Mapping Degree of the other vertex (vertex 5) to 0. This makes the other vertex to become a member of the updated NS_0 and could be added to the independent set at the next iteration. In comparison, since vertex $2 \in NS_1$ has an Expanding Degree of 1, swapping 2 into S cannot create any improving moves.

The second part of this *Selection Rule* is based on the consideration that the vertices of NS_1 with a larger Diversifying Degree could make the search more diversified after a (1,1)-swap move. Indeed, after swapping $v_i \in NS_1$ and $v_j \in S$, we need to update the Mapping Degree, the Expanding Degree and Diversifying Degree concerned by v_i and v_j (see Appendix), leading to modifications of the neighborhoods NS_k ($k = 0, 1, 2, >2$). By definition, a vertex $v_i \in NS_1$ with a larger Diversifying Degree has more adjacent vertices in $V \setminus S$. Selecting such a vertex for a (1,1)-swap move leads to more changes in $V \setminus S$, thus more changes in the neighborhoods NS_k ($k = 0, 1, 2, >2$). In this sense, this helps to diversify the choices of the next iteration of the search procedure. For our example in Fig. 1, vertices 3 and 5 have respectively a Diversifying Degree of 2 and 3. According to the *Selection Rule*, vertex 5 (instead of vertex 3) is selected to take part in the swap move with vertex 4 in S. After this move, three vertices (2, 9 and 10 which are adjacent to 5 in $V \setminus S$) take part in neighborhood updating. In comparison, vertex 3 in NS_1 (with a small Diversifying Degree) will induce fewer changes in the neighborhoods.

One notices that this heuristic selection rule has no theoretical guarantee of being able to always lead to the best choice. However, the rule is designed to favor a good choice when such a choice is available. The computational results shown in the paper confirm its usefulness in practice.

Further reducing NS_1 neighborhood examination:

As explained above, among the vertices of NS_1, those v_i whose adjacent neighbor v_j in S has an Expanding Degree of 1 are less promising than the other vertices since using these v_i in (1,1)-swap can only lead to new side-walk (or degrading) moves and can never create new improving moves for the next iteration. In order to prevent the search from making uselessly too many side-walk moves, we define an additional rule to reduce NS_1 as follows. If there are more (1,1)-swap moves than $(k,1)-$swap ($k > 1$) moves (i.e., $|NS_1| > |NS_2| + |NS_{>2}|$), we exclude from NS_1 any v_i such that its adjacent neighbor v_j has an Expanding Degree of 1. For instance, in Fig. 1, $NS_1 = \{2, 3, 5\}$. If we apply this reduction rule, vertex 2 will be excluded from NS_1 since the Expanding Degree of its neighbor in S (vertex 1) equals to 1. Experiments show that this reduction rule could improve the search efficiency for a number of situations where a large number of side-walk moves frequently appear during the search process.

Algorithm 2. The Intensification Step for MIS.

```
1:  Input: A feasible independent set S
2:  Output: The independent set S'.
3:  /* Explore neighborhood NS_0 with an improving (0,1)-swap
    move */
4:  if NS_0 is not empty then
5:     Choose randomly a vertex v_i from NS_0;
6:     S' ← S ⊕ (0,1)−swap;
7:  else
8:     /* Explore neighborhood NS_1 with a side-walk (1,1)-
       swap move */
9:     if |NS_1| > |NS_2| + |NS_{>2}| then
10:       Exclude vertices v_i from NS_1 whose neighbor v_j in S
          satisfies K^E(v_j) = 1;
11:    end if
12:    Determine a vertex v_i from NS_1 according to Selection
       Rule;
13:    if v_i is obtained then
14:       S' ← S ⊕ (1,1)−swap;
15:    else
16:       S' ← S;
17:    end if
18: end if
19: Perform the updating procedure; /* see Appendix */
20: Return S';
```

The pseudo-code of one intensification iteration is given in Algorithm 2 where S is the current independent set and NS_k ($k = 0, 1, 2, >2$) are the associated neighborhoods.

Notice that after each $(1,1)-$swap, the neighborhoods NS_k are updated accordingly (see Appendix). Additionally, the vertex that is swapped out from S is added to the tabu list to prevent it from being moved back to S for a number of next iterations (see Section 2.6).

If NS_0 is empty and NS_1 does not offer any eligible (1,1)-swap move (i.e., NS_1 is empty or all the vertices of NS_1 are forbidden by the tabu list), the search continues with a diversification step which is explained in the next section.

2.8. Diversification

When the current solution cannot be further improved by a (0,1)-swap or changed by a (1,1)-swap, the search procedure is trapped in a local optimum. To escape from this local optimum, the SBTS algorithm resorts to $(k,1)-$swap ($k \geq 2$) moves to perturb the current solution in order to displace the search to a new search zone. These swap moves are carried out according to some dedicated rules which once again depend on problem specific information.

One observes first that a $(k,1)-$swap ($k \geq 2$) move applied to a solution S deteriorates the cardinality of S by exactly $k-1$ units. Consequently, a smaller k (e.g., $k=2$) perturbs more weakly a solution while a larger k (e.g., $k > 2$) changes more strongly the solution. To control the perturbation strength, SBTS adopts an adaptive strategy according to a relation between the number of possible $(1,1)-$swap moves (i.e., $|NS_1|$) and the number of $(k,1)-$swap ($k > 1$) moves (i.e., $|NS_2| + |NS_{>2}|$). Precisely, the adaptive perturbation strategy is defined as follows.

1. If $|NS_1| > |NS_2| + |NS_{>2}|$, SBTS uses $NS_{>2}$ to perform a strong perturbation by a $(k,1)-$swap ($k > 2$) move as follows: Select an eligible vertex v_i of $NS_{>2}$ with the largest Diversifying Degree (ties are broken at random) and swap the chosen vertex v_i with its k neighbors in S.
2. Otherwise, SBTS applies with equal probability either NS_2 or $NS_{>2}$ to perform either a weak or strong perturbation.
   - $k = 2$: Select an eligible vertex v_i of NS_2 with the largest Diversifying Degree (ties are broken at random) and then swap v_i with its two neighbors in S.

– $k > 2$: Determine an eligible v_i of $NS_{>2}$ at random without considering the tabu list and then swap the chosen vertex v_i with its k neighbors in S.

The underlying rationale for point (1) is that when a local optimum is reached, all the vertices of NS_1 are prohibited by the tabu list (i.e., they have been removed recently from the independent set S, see Section 2.6). A large NS_1 indicates thus that in the recent past, the search has gone through a high number of side-walk moves. This situation corresponds to a kind of deep local optimum which is difficult to escape. To displace the search into a new search zone, we need to apply a strong perturbation which is achieved by employing a $(k,1)-$swap move with $k > 2$.

The second case corresponds to the situation where the search has made a relative small number of side-walk moves. In this case, we alternate probabilistically the perturbation strength to try to find better solutions in a zone around the current local optimum (with a weak perturbation) or far from current local optimum (with a strong perturbation).

Like for an intensification step, after a $(k,1)$-swap ($k \geq 2$), the neighborhood sets NS_k ($k = 0, 1, 2, >2$) are updated accordingly (see Appendix). The k vertices that are swapped out from S are added to the tabu list (Section 2.6).

3. Experimental results

3.1. Benchmark instances

To evaluate the efficiency of our proposed SBTS algorithm, we carry out experiments on three different data sets: DIMACS, BHOSLIB and CODE.

- *DIMACS benchmark*: This set was established for the Second DIMACS Implementation Challenge (Johnson and Trick, 1996). It contains 12 varieties of instances with multiple topologies and densities. It is composed of 80 graphs with size ranging from less than 50 vertices and 1000 edges up to more than 4000 vertices and 5 000 000 edges. These instances are the most popular and frequently used for evaluating algorithms for MIS, MC and MVC. Among these 80 DIMACS instances, the maximum clique is now known for 76 of them except 4 graphs: 3 (large) random graphs with at least 500 vertices (C500.9, C1000.9, C2000.9) and 1 structured graphs (johnson32_2_4) (McCreesh and Prosser, 2013; Wu and Hao, 2014). These instances are available from http://www.cs.hbg.psu.edu/txn131/clique.html.
- *BHOSLIB benchmark*: This set arose from the SAT'04 Competition. The BHOSLIB instances were translated from hard random SAT instances (Xu et al., 2005). Each of the 40 instances has a known, but hidden optimal solution. These instances have a size of ranging from less than 500 vertices and 100 000 edges up to more than 1500 vertices and 10 000 000 edges. The set is more and more used in the literature for performance evaluation. These instances are available from http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm.
- **CODE benchmark**: This set is composed of 11 large graphs arising from code theory with size ranging from 1024 vertices and 7936 edges up to 4096 vertices and more than 184 320 edges. These instances have unknown optima and are the least frequently used in the literature. They are available from http://neilsloane.com/doc/graphs.html.

Since the original DIMACS graphs are proposed for MC, we use their complement graphs to test our SBTS algorithm. For BHOSLIB and CODE benchmarks, the original graphs are used.

3.2. Experimental protocol

Our SBTS algorithm is coded in C++[1] and compiled using g++ with the '-O2' option on a Cluster running Linux with 2.83 GHz and 8 GB. When we run the DIMACS machine benchmark program[2] on our machine, we obtain the following results: 0.20 CPU seconds for graph r300.5, 1.23 CPU seconds for r400.5 and 4.68 CPU seconds for r500.5.

Given its stochastic nature, we run SBTS independently 100 times to solve each instance with initial solutions generated by the procedure of Section 2.3. The stop condition of each run is a maximum of 10^8 iterations which are divided into 10^4 restarts, each restart being limited to 10^4 iterations (i.e., $Iters_{\max} = 10^4$, Algorithm 1). This experimental protocol is typically used in the literature (see next section). SBTS runs with the self-tuned tabu tenure tt given in Section 2.6. Though fine-tuning tabu tenure would lead to improved results for some graphs, for our experiments, we used the above tabu tenure except as otherwise stated. No other parameter is required by SBTS.

3.3. Computational results of SBTS on DIMACS, BHOSLIB and CODE instances

Tables 2–4 show respectively the computational statistics of the SBTS algorithm on the three sets of benchmark instances with respect to f_{bk} which designates the optimal value or the best lower bound (i.e., the largest independent set ever reported in the literature). Notice that for the popular DIMACS and BHOSLIB benchmarks, recent heuristics can attain the f_{bk} value for many cases, as it is shown in Table 5 of Section 3.4.

Table 2 shows the computational statistics of the SBTS algorithm on the set of 80 DIMACS instances. Columns 1–4 give the characteristics of each graph: the name, the number of vertices and edges, and the optimal or best-known result f_{bk} (optimal values are marked with '*'). The columns under heading "SBTS" list our best result f_*, the average result f_{avg}, the successful runs *Success* for reaching f_* over the 100 independent runs, the average iterations *AvgIters* and the average CPU time $t(s)$ (seconds) over the successful runs.

Table 2 demonstrates that SBTS obtains quite competitive results on the set of DIMACS instances. Specifically, SBTS can consistently reach the previous best-known solutions for 75 out of the 80 instances with a perfect success rate. Furthermore, SBTS can reach the best-known results for *all* the instances with various topologies and densities, including the most difficult graphs (brock 800_x($x = 1, 2, 3, 4$), C2000.9, MANN_a 45 and MANN_a 81). To the best of our knowledge, the top-performing heuristics in the literature miss at least one best-known result of these difficult graphs. On the other hand, one observes that SBTS has a low success rate (less than 50%) for 3 graphs. Notice that for the 4 open instances (C500.9, C1000.9, C2000.9, johnson32_2_4), SBTS hits the best lower bounds for each run except for C2000.9. One can speculate that these lower bounds (except for C2000.9) would be close to or would be optimal solutions and thus are difficult to improve, even though this observation does not constitute a proof. As for the computing time, SBTS requires on average less than 1000 seconds except for C2000.9 and C4000.5.

Table 3 reports the computational results of SBTS on the set of 40 BHOSLIB instances. The columns 1–4 give the characteristic of the graphs and columns 5–9 present the detailed results of the proposed SBTS algorithm. From this table, one finds that SBTS also

[1] The source code of our SBTS algorithm will be available at: http://www.info.univ-angers.fr/pub/hao/mis.html.

[2] ftp://dimacs.rutgers.edu/pub/dsj/clique/

Table 2
Detailed computational results of SBTS on the set of 80 DIMACS instances. Each instance is solved 100 times and each run is limited to a maximum of 10^8 iterations.

Characteristics of the graphs				SBTS				
Name	$\|V\|$	$\|E\|$	f_{bk}	f_*	$f_{avg}(Std.)$	*Success*	*AvgIters*	t (s)
brock200_1	200	14 834	21*	21	21.00	100/100	329	0.0005
brock200_2	200	9876	12*	12	12.00	100/100	12 859	0.0450
brock200_3	200	12 048	15*	15	15.00	100/100	11 606	0.0296
brock200_4	200	13 089	17*	17	17.00	100/100	33 277	0.0768
brock400_1	400	59 723	27*	27	27.00	100/100	15 572 682	66.3977
brock400_2	400	59 786	29*	29	29.00	100/100	4 159 016	20.2432
brock400_3	400	59 681	31*	31	31.00	100/100	340 265	1.7676
brock400_4	400	59 765	33*	33	33.00	100/100	160 679	0.8298
brock800_1	800	207 505	23*	23	21.52 (0.88)	26/100	56 901 393	963.6288
brock800_2	800	208 166	24*	24	22.29 (1.49)	43/100	46 449 467	784.3647
brock800_3	800	207 333	25*	25	24.16 (1.35)	72/100	40 752 010	651.0726
brock800_4	800	207 643	26*	26	25.90 (0.70)	98/100	25 784 326	421.6047
C125.9	125	6963	34*	34	34.00	100/100	85	0.0001
C250.9	250	27 984	44*	44	44.00	100/100	492	0.0005
C500.9	500	112 332	57	57	57.00	100/100	23 184	0.0540
C1000.9	1000	450 079	68	68	68.00	100/100	1 438 740	8.3696
C2000.5	2000	999 836	16*	16	16.00	100/100	7628	0.9211
C2000.9	2000	1 799 532	80	80	77.29 (0.64)	2/100	79 591 805	1515.5700
C4000.5	4000	4 000 268	18*	18	18.00	100/100	4 289 342	1553.2406
DSJC500.5	500	125 248	13*	13	13.00	100/100	524	0.0074
DSJC1000.5	1000	499 652	15*	15	15.00	100/100	26 455	2.2891
keller4	171	9435	11*	11	11.00	100/100	27	0.0001
keller5	776	225 990	27*	27	27.00	100/100	2101	0.0194
keller6	3361	4 619 898	59	59	59.00	100/100	12 311 511	754.8754
MANN_a9	45	918	16*	16	16.00	100/100	3	0.0000
MANN_a27	378	70 551	126*	126	126.00	100/100	635	0.0008
MANN_a45	1035	533 115	345*	345	345.00	100/100	4 763 131	27.5632
MANN_a81	3321	5 506 380	1100*	1100	1100.00	100/100	716 340	22.6991
hamming6-2	64	1824	32*	32	32.00	100/100	18	0.0000
hamming6-4	64	704	4*	4	4.00	100/100	2	0.0000
hamming8-2	256	31 616	128*	128	128.00	100/100	217	0.0001
hamming8-4	256	20 864	16*	16	16.00	100/100	10	0.0000
hamming10-2	1024	518 656	512*	512	512.00	100/100	365	0.0004
hamming10-4	1024	434 176	40	40	40.00	100/100	255	0.0037
gen200-p0.9-44	200	17 910	44*	44	44.00	100/100	947	0.0005
gen200-p0.9-55	200	17 910	55*	55	55.00	100/100	576	0.0004
gen400-p0.9-55	400	71 820	55*	55	55.00	100/100	1504	0.0027
gen400-p0.9-65	400	71 820	65*	65	65.00	100/100	186	0.0006
gen400-p0.9-75	400	71 820	75*	75	75.00	100/100	432	0.0010
c-fat200-1	200	1534	12*	12	12.00	100/100	34	0.0001
c-fat200-2	200	3235	24*	24	24.00	100/100	119	0.0009
c-fat200-5	200	8473	58*	58	58.00	100/100	89	0.0005
c-fat500-1	500	4459	14*	14	14.00	100/100	63	0.0013
c-fat500-2	500	9139	26*	26	26.00	100/100	76	0.0020
c-fat500-5	500	23 191	64*	64	64.00	100/100	102	0.0031
c-fat500-10	500	46 627	126*	126	126.00	100/100	157	0.0045
johnson8-2-4	28	210	4*	4	4.00	100/100	1	0.0000
johnson8-4-4	70	1855	14*	14	14.00	100/100	3	0.0000
johnson16-2-4	120	5460	8*	8	8.00	100/100	1	0.0000
johnson32-2-4	496	107 880	16	16	16.00	100/100	1	0.0000
p_hat300-1	300	10 933	8*	8	8.00	100/100	35	0.0002
p_hat300-2	300	21 928	25*	25	25.00	100/100	22	0.0001
p_hat300-3	300	33 390	36*	36	36.00	100/100	32	0.0001
p_hat500-1	500	31 569	9*	9	9.00	100/100	105	0.0028
p_hat500-2	500	62 946	36*	36	36.00	100/100	157	0.0011
p_hat500-3	500	93 800	50*	50	50.00	100/100	607	0.0028
p_hat700-1	700	60 999	11*	11	11.00	100/100	1620	0.0476
p_hat700-2	700	121 728	44*	44	44.00	100/100	57	0.0014
p_hat700-3	700	183 010	62*	62	62.00	100/100	153	0.0021
p_hat1000-1	1000	122 253	10*	10	10.00	100/100	48	0.0069
p_hat1000-2	1000	244 799	46*	46	46.00	100/100	343	0.0150
p_hat1000-3	1000	371 746	68*	68	68.00	100/100	700	0.0127
p_hat1500-1	1500	284 923	12*	12	12.00	100/100	85 232	13.0180
p_hat1500-2	1500	568 960	65*	65	65.00	100/100	365	0.0202
p_hat1500-3	1500	847 244	94*	94	94.00	100/100	683	0.0169
san200_0.7_1	200	13 930	30*	30	30.00	100/100	6682	0.0139
san200_0.7_2	200	13 930	18*	18	18.00	100/100	765	0.0015
san200_0.9_1	200	17 910	70*	70	70.00	100/100	890	0.0008
san200_0.9_2	200	17 910	60*	60	60.00	100/100	37	0.0001
san200_0.9_3	200	17 910	44*	44	44.00	100/100	3192	0.0025
san400_0.5_1	400	39 900	13*	13	13.00	100/100	2586	0.0248
san400_0.7_1	400	55 860	40*	40	40.00	100/100	137 032	0.8490
san400_0.7_2	400	55 860	30*	30	30.00	100/100	11 758	0.0691

Table 2 (*continued*)

Characteristics of the graphs				SBTS				
Name	$\|V\|$	$\|E\|$	f_{bk}	f_*	$f_{avg}(Std.)$	*Success*	*AvgIters*	t (s)
san400_0.7_3	400	55 860	22*	22	22.00	100/100	14 543	0.0818
san400_0.9_1	400	71 820	100*	100	100.00	100/100	3823	0.0075
san1000	1000	250 500	15*	15	15.00	100/100	1 410 471	46.4781
sanr200_0.7	200	13 868	18*	18	18.00	100/100	435	0.0008
sanr200_0.9	200	17 863	42*	42	42.00	100/100	665	0.0004
sanr400_0.5	400	39 984	13*	13	13.00	100/100	552	0.0041
sanr400_0.7	400	55 869	21*	21	21.00	100/100	559	0.0026

Table 3
Detailed computational results of SBTS on the set of 40 BHOSLIB instances. Each instance is solved 100 times and each run is limited to a maximum of 10^8 iterations.

Characteristics of the graphs				SBTS				
Name	$\|V\|$	$\|E\|$	f_{bk}	f_*	$f_{avg}(Std.)$	*Success*	*AvgIters*	$t(s)$
frb30-15-1	450	83 198	30*	30	30.00	100/100	8182	0.0264
frb30-15-2	450	83 151	30*	30	30.00	100/100	15 420	0.0562
frb30-15-3	450	83 216	30*	30	30.00	100/100	14 226	0.0498
frb30-15-4	450	83 194	30*	30	30.00	100/100	15 057	0.0564
frb30-15-5	450	83 231	30*	30	30.00	100/100	44 397	0.1611
frb35-17-1	595	148 859	35*	35	35.00	100/100	171 245	0.6895
frb35-17-2	595	148 868	35*	35	35.00	100/100	96 263	0.4014
frb35-17-3	595	148 784	35*	35	35.00	100/100	26 607	0.1027
frb35-17-4	595	148 873	35*	35	35.00	100/100	232 484	0.8445
frb35-17-5	595	148 572	35*	35	35.00	100/100	61 897	0.2411
frb40-19-1	760	247 106	40*	40	40.00	100/100	46 853	0.2384
frb40-19-2	760	247 157	40*	40	40.00	100/100	4 316 200	20.6966
frb40-19-3	760	247 325	40*	40	40.00	100/100	550 507	2.7152
frb40-19-4	760	246 815	40*	40	40.00	100/100	2 187 043	11.6519
frb40-19-5	760	246 801	40*	40	40.00	100/100	9 738 592	55.9365
frb45-21-1	945	3 86 854	45*	45	45.00	100/100	2 800 152	23.0019
frb45-21-2	945	387 416	45*	45	45.00	100/100	6 617 043	53.1387
frb45-21-3	945	387 795	45*	45	45.00	100/100	13 044 028	98.7293
frb45-21-4	945	38 7491	45*	45	45.00	100/100	5 276 955	42.9933
frb45-21-5	945	387 461	45*	45	45.00	100/100	10 366 230	80.3013
frb50-23-1	1150	580 603	50*	50	49.75 (0.43)	75/100	37 020 418	368.6880
frb50-23-2	1150	579 824	50*	50	49.49 (0.50)	49/100	40 728 061	302.2116
frb50-23-3	1150	579 607	50*	50	49.13 (0.34)	13/100	62 285 352	517.4469
frb50-23-4	1150	580 417	50*	50	50.00	100/100	10 398 673	94.6139
frb50-23-5	1150	580 640	50*	50	50.00	100/100	17 773 506	119.3523
frb53-24-1	1272	714 129	53*	53	52.03 (0.17)	3/100	55 616 513	498.3600
frb53-24-2	1272	714 067	53*	53	52.30 (0.46)	30/100	34 698 236	321.4827
frb53-24-3	1272	714 229	53*	53	52.66 (0.47)	66/100	43 238 016	359.6429
frb53-24-4	1272	714 048	53*	53	52.22 (0.41)	22/100	42 301 012	340.4545
frb53-24-5	1272	714 130	53*	53	52.91 (0.29)	91/100	27 705 528	273.8870
frb56-25-1	1400	869 624	56*	56	55.07 (0.26)	7/100	54 577 332	551.6357
frb56-25-2	1400	869 899	56*	56	55.06 (0.31)	8/100	51 319 979	470.2750
frb56-25-3	1400	869 921	56*	56	55.30 (0.46)	30/100	41 946 192	383.5283
frb56-25-4	1400	869 262	56*	56	55.86 (0.35)	86/100	34 350 025	335.2178
frb56-25-5	1400	869 699	56*	56	55.79 (0.41)	79/100	38 964 414	568.9961
frb59-26-1	1534	1 049 256	59*	59	58.02 (0.20)	2/100	25 836 232	261.1367
frb59-26-2	1534	1 049 648	59*	59	57.96 (0.24)	1/100	78 884 128	762.2700
frb59-26-3	1534	1 049 729	59*	59	57.94 (0.37)	4/100	42 283 734	388.9075
frb59-26-4	1534	1 048 800	59*	59	58.00 (0.32)	5/100	74 758 342	969.9380
frb59-26-5	1534	1 049 829	59*	59	58.81 (0.46)	84/100	40 168 986	394.9430

performs well for this benchmark set. Specifically, SBTS reaches the optimal results with a perfect success rate for the instances with up to 1000 vertices. The BHOSLIB set is known to be more difficult compared to most of the DIMACS benchmark. Yet, SBTS can still attain the optimal results for all the 40 instances. On the other hand, SBTS has a low or very low success rate (less than 50%) for 11 graphs and requires a large computing time for the largest instances.

Table 4 reports the computational statistics of our SBTS algorithm on the set of 11 CODE instances where the best-known results f_{bk} are from (Andrade et al., 2012; Sloane, 2000). The results of SBTS are obtained with the default tabu tenure tt except those of 1et.2048, 1tc2048 and 1zc.4096 for which $tt = 40 + Randdom(|NS1|)$. From the table, one finds that SBTS attains the best-known results for all the CODE instances in a short time. SBTS reaches the best-known results with a perfect success rate for 9 out of 11 instances. However, for one case (1zc.4096), its success rate is very low (1%).

From the results on DIMACS, BHOSLIB and CODE benchmarks, one observes that there is no clear correlation between the problem size and the necessary time to solve it since the difficulty of an instance also depends on its structure.

Table 4
Detailed computational results of SBTS on the set of 11 CODE instances. Each instance is solved 100 times and each run is limited to a maximum of 10^8 iterations.

Characteristics of the graphs				SBTS				
Name	$\|V\|$	$\|E\|$	f_{bk}	f_*	$f_{avg}(Std.)$	*Success*	*AvgIters*	$t(s)$
1dc.1024	1024	24 063	94	94	94.00	100/100	10 764	0.0289
1dc.2048	2048	58 367	172	172	172.00	100/100	33 895	0.1678
1et.1024	1024	9600	171	171	171.00	100/100	38 018	0.0669
1et.2048	2048	22 528	316	316	316.00	100/100	2 016 754	6.5911
1tc.1024	1024	7936	196	196	196.00	100/100	10 798	0.0157
1tc.2048	2048	18 944	352	352	352.00	100/100	1 622 215	6.5195
1zc.1024	1024	33 280	112	112	111.99 (0.01)	99/100	9 984 898	29.7898
1zc.2048	2048	78 848	198	198	198.00	100/100	18 931 972	81.6927
1zc.4096	4096	184 320	379	379	373.00 (3.57)	1/100	96 720 177	1187.4100
2dc.1024	1024	169 162	16	16	16.00	100/100	1371	0.0327
2dc.2048	2048	504 451	24	24	24.00	100/100	154 941	6.9823

Table 5
Comparisons of SBTS with five reference algorithms on 45 most difficult DIMACS and BHOSLIB instances. Each instance is solved 100 times and each run is limited to a maximum of 10^8 iterations for PLS (Pullan, 2006, 2008), COVER (Richter et al., 2007), MN/LS (Wu et al., 2012), $1.6*10^8$ iterations for BLS (Benlic and Hao, 2013) and 2000 seconds for NuMVC (Cai et al., 2013).

Graph		MN/TS		BLS		PLS		COVER		NuMVC		SBTS	
Name	f_{bk}	f_*	$t(s)$	f_*	$t(s)$	f_*	$t(s)$	f_*	$t(s)$	f_*	$t(s)$	f_*	$t(s)$
brock200_1	21*	21	0.01	21	0.01	21	0.00	21	0.01	–	–	21	0.00
brock200_2	12*	12	0.06	12	0.18	12	0.03	12	0.43	12	0.13	12	0.05
brock200_3	15*	15	0.07	15	0.57	15	0.03	15	7.62	–	–	15	0.03
brock200_4	17*	17	0.09	17	0.43	17	0.08	17	7.90	17	1.26	17	0.08
brock400_1	27*	27	10.27	27	121.40	27	1.08	*25* (*25.00*)	–	–	–	27	66.40
brock400_2	29*	29	1.34	29	17.40	29	0.38	*28* (*27.01*)	–	29 (28.84)	572.39	29	20.24
brock400_3	31*	31	0.63	31	5.08	31	0.18	31 (30.50)	135.26	–	–	31	1.77
brock400_4	33*	33	0.28	33	3.17	33	0.10	33 (32.70)	112.98	33	4.98	33	0.83
brock800_1	23*	23 (22.72)	188.14	23 (22.40)	1568.24	23	30.09	*21* (*21.00*)	–	–	–	23 (21.52)	963.63
brock800_2	24*	24(23.88)	156.47	24 (23.04)	1078.13	24	24.41	*22* (*22.00*)	–	*21* (*21.00*)	–	24 (22.29)	784.36
brock800_3	25*	25	118.57	25 (24.52)	1020.11	25	15.08	*23* (*23.00*)	–	–	–	25 (24.16)	651.07
brock800_4	26*	26	62.38	26	601.74	26	6.54	*24* (*24.00*)	–	*21* (*21.00*)	–	26 (25.90)	421.60
C125.9	34*	34	0.01	34	0.00	34	0.00	34	0.01	34	0.00	34	0.00
C250.9	44*	44	0.01	44	0.00	44	0.00	44	0.01	44	0.00	44	0.00
C500.9	57	57	0.06	57	0.00	57	0.19	57	0.31	57	0.13	57	0.05
C1000.9	68	68	0.63	68	35.70	68	1.88	68	5.82	68	2.02	68	8.37
C2000.5	16*	16	0.07	16	2.90	16	0.73	16	3.78	16	2.93	16	0.92
C2000.9	80	80 (78.37)	563.70	80 (78.60)	4811.17	*78* (*78.00*)	–	*78* (*77.84*)	–	80 (78.71)	1393.30	80 (77.29)	1515.57
C4000.5	18*	18	144.37	18	654.60	18	149.65	18	689.74	18	252.81	18	1553.24
keller4	11*	11	0.01	11	0.00	11	0.00	11	0.01	11	0.00	11	0.00
keller5	27*	27	0.05	27	0.09	27	0.05	27	0.07	27	0.04	27	0.02
keller6	59	59	97.87	59	24.80	59 (57.75)	550.95	59	15.63	59	2.51	59	754.88
MANN_a27	126*	126	3.42	126	35.20	126	0.03	126	0.01	126	0.00	126	0.00
MANN_a45	345*	*340* (*340.00*)	90.58	*342* (*340.82*)	–	*344* (*344.00*)	28.76	345 (344.41)	–	345	86.36	345	27.56
MANN_a81	1100*	*1090* (*1090.00*)	632.24	*1094* (*1092.17*)	–	*1098* (*1098.00*)	269.66	1100 (1098.11)	–	1100 (1099.06)	732.90	1100	22.70
frb50-23-1	50*	50 (49.84)	116.92	50 (49.96)	882.22	50 (49.72)	1045.59	50 (49.89)	171.92	50	38.14	50 (49.75)	368.69
frb50-23-2	50*	50 (49.47)	161.77	50 (49.56)	1074.17	50 (49.45)	1171.15	50 (49.30)	1.72	50	176.59	50 (49.49)	302.21
frb50-23-3	50*	50 (49.15)	214.58	50 (49.08)	1037.68	50 (49.16)	1041.40	50 (49.24)	2.61	50 (49.95)	532.81	50 (49.13)	517.45
frb50-23-4	50*	50	11.91	50	55.51	50	126.44	50	16.94	50	7.89	50	94.61
frb50-23-5	50*	50	50.90	50	142.93	50 (49.99)	436.29	50 (49.98)	88.94	50	19.53	50	119.35
frb53-24-1	53*	53 (52.03)	240.36	53 (52.04)	2306.74	53 (52.06)	1707.39	53 (52.09)	11.31	53 (52.86)	715.12	53 (52.03)	498.36
frb53-24-2	53*	53 (52.30)	209.89	53 (52.16)	2015.13	53 (52.23)	1548.89	53 (52.34)	4.24	53	205.35	53 (52.30)	321.48
frb53-24-3	53*	53 (52.91)	253.96	53 (52.88)	1199.95	53 (52.66)	1185.68	53 (52.91)	157.80	53	51.23	53 (52.66)	359.64
frb53-24-4	53*	53 (52.45)	178.01	53 (52.54)	1361.23	53 (52.46)	1423.27	53 (52.24)	10.74	53	266.87	53 (52.22)	340.45
frb53-24-5	53*	53 (52.90)	278.31	53 (52.90)	1100.00	53 (52.85)	979.85	53 (52.84)	253.05	53	39.89	53 (52.91)	273.89
frb56-25-1	56*	56 (55.22)	174.02	56 (55.20)	2304.83	56 (55.10)	1240.19	56 (55.15)	20.73	56	470.68	56 (55.07)	551.64
frb56-25-2	56*	56 (55.12)	127.16	56 (55.06)	1500.44	56 (54.93)	1702.39	56 (55.12)	30.33	56 (55.97)	617.49	56 (55.06)	470.28
frb56-25-3	56*	56 (55.25)	209.48	56 (55.20)	1409.09	56 (55.08)	1476.61	56 (55.76)	435.30	56	121.30	56 (55.30)	383.53
frb56-25-4	56*	56 (55.85)	158.14	56 (55.86)	999.51	56 (55.66)	1304.03	56 (55.84)	291.11	56	49.45	56 (55.86)	335.22
frb56-25-5	56*	56	85.57	56	591.49	56 (55.81)	1089.08	56 (55.98)	89.58	56	26.76	56 (55.79)	569.00
frb59-26-1	59*	59 (58.05)	242.75	59 (57.96)	3298.21	*58* (*57.85*)	–	59 (58.11)	30.76	59 (58.88)	687.85	59 (58.02)	261.14
frb59-26-2	59*	59 (58.01)	396.38	59 (58.00)	2399.92	*58* (*57.63*)	–	59 (58.06)	40.86	59 (58.38)	1160.02	59 (57.96)	762.27
frb59-26-3	59*	59 (58.23)	197.36	59 (58.31)	2338.59	59 (57.77)	1929.05	59 (58.12)	65.04	59 (58.96)	580.03	59 (57.94)	388.91
frb59-26-4	59*	59 (58.10)	192.45	59 (58.20)	1823.63	59 (57.71)	2044.91	59 (58.01)	73.92	59 (58.79)	741.21	59 (58.00)	969.94
frb59-26-5	59*	59 (58.99)	96.09	59	403.30	59 (58.77)	1193.22	59 (58.89)	292.60	59	61.91	59 (58.81)	394.94
Best #(*Avg.*)		−2 (74.02)		−2 (74.05)		−5 (74.18)		−7 (74.01)		−(≥2)(≤74.36)		0 (74.21)	

3.4. Comparisons with seven state-of-the-art algorithms

To assess the performance of the proposed SBTS algorithm relative to the state-of-the-art methods, we compare in this section SBTS with some best-performing algorithms for MIS, MC and MVC in the literature. We present two comparisons which concern the DIMACS and BHOSLIB sets on the one hand and the CODE set on the other hand.

3.4.1. Comparisons with five references algorithms on DIMACS and BHOSLIB benchmarks

For this comparison, we focus on 45 most difficult instances from DIMACS and BHOSLIB sets and ignore the other instances since they can be easily solved with a 100% success rate by all the compared algorithms. First we summarize below the experimental conditions used by 5 reference algorithms which are implemented on sequential architectures and report state-of-the-art computational results on both DIMACS and BHOSLIB benchmarks.

- MN/TS (Wu et al., 2012): This is a multi-neighborhood tabu search algorithm which is designed for the equivalent maximum clique (and its weighted generalization). It is run on a PC with 2.83 GHz CPU and 8 GB RAM, and the stop condition is a maximum of 10^8 iterations.
- BLS (Benlic and Hao, 2013): This is an iterated local search algorithm which combines a descent procedure with a dedicated and adaptive perturbation strategy. BLS is run on a Xeon E5440 with 2.83 GHz and 2 GB, and the stop condition is a maximum of 1.6×10^8 iterations.
- PLS (Pullan, 2006, 2008): This is a highly effective phased local search algorithm which relies on three sub-algorithms using different vertex selection rules. It is run on a Pentium IV machine with 512KB L2 cache and 512 MB RAM, and the stop condition is a maximum of 10^8 iterations for all instances except for MANN_a45 and MANN_a81 where 10^9 iterations are allowed.
- COVER (Richter et al., 2007): This is a local search algorithm designed for the equivalent minimum vertex cover problem which uses edge weighting techniques. It is run on a machine with 2.13 GHz and 2 GB RAM, and the stop condition is a maximum of 10^8 iterations.
- NuMVC (Cai et al., 2013): This is a very recent local search algorithm for MVC using edge weighting techniques. It is run on a machine with 3 GHz CPU and 4 GB RAM, and the stop condition is a cutoff time which is set to 2000 s.

Table 5 summarizes the results of the competing algorithms. All the results are based on 100 independent runs for each graph. The reported results of the reference algorithms are extracted from the corresponding papers while the results of SBTS are from Tables 2 and 3.

For each compared algorithm, we show its best result f_*, followed by the average result f_{avg} given in parenthesis over 100 runs if the success rate is lower than 100%, and the average time in seconds $t(s)$ over the successful runs. For COVER, as stated in Richter et al. (2007), $t(s)$ is the median run time which is indicative of a typical run of the algorithm.

"Best #" in the last row of Table 5 shows the number of instances for which an algorithm cannot reach the best-known results in the literature and "Avg." indicates the average value of f_{avg} for the 45 instances for each algorithm. Note that, "-" in Table 5 means that the result is unavailable.

One observes from Table 5 that except SBTS, each reference algorithm fails to find the best-known results for at least two instances (entries in italic). Indeed, given that these instances have very different characteristics and structures, it is known that it is very difficult for a single heuristic to perform well on all the instances (Pullan et al., 2011). Besides, SBTS has a slightly better average result of 74.21 against 74.18 of PLS which is the best among the reference algorithms (except NuMVC whose average is an optimistic upper bound since its results are missing for six instances). When we examine the results of Table 5 in detail, we can make the following observations.

For the DIMACS instances, MN/TS, BLS and PLS (which are maximum clique or maximum independent set algorithms) reach the best reported results for the groups "brock", "C", "keller" with a high success rate except for C2000.9 which is among the most difficult instance. For this instance, MN/TS and BLS achieve the best-known result (80) with an average of 78.37 and 78.60 respectively while PLS fails to find solutions larger than 78. For the group "MANN", MN/TS, BLS and PLS cannot to reach the best-known results of MANN_a45 (345) and MANN_a81 (1100). The largest solutions they find have a size of 340, 342, 344 for MANN_a45, and a size of 1090, 1094, 1098 for MANN_a81 respectively. Generally, it seems that the typical MC or MIS algorithms (e.g., MN/TS, BLS, PLS) have serious difficulties to solve these two "MANN" instances.

By contrast, the typical MVC algorithms COVER and NuMVC perform well on the group "MANN" with a high success rate while they clearly encounter difficulties for the group "brock". Indeed, COVER fails to reach the best-known result for 6 out of the 12 brock instances. For the 6 brock instances tested by NuMVC, two results do not match the best-known values. Besides, for C2000.9, NuMVC can achieve the best-known result of 80 while COVER can only achieve a solution of size of 78.

Our SBTS algorithm achieves the best-known results for all 25 DIMACS instances including the two 'problematic' groups "brock" and "MANN". In particular, SBTS can attain the best results for MANN_a45 and MANN_a81 with a perfect success rate, which is better than the typical MC or MIS algorithms.

The average results given in parenthesis show that MN/TS, BLS, PLS, COVER and NuMVC can attain the reported best results in every single run for 20, 19, 21, 14 and 20 cases out of 25 DIMACS instances respectively, while SBTS has a perfect success rate for 20 cases, which is more than BLS and COVER, equal to MN/TS and NuMVC and one less than PLS. However, for C2000.9, the average result of SBTS is slightly worse than the reference algorithms.

For the BHOSLIB instances, the reference algorithms can attain the best-known results except PLS which fails to reach the optimal solutions for frb59-26-1 and frb59-26-2. One observes from the average results that MN/TS, BLS, PLS, COVER and NuMVC can attain the optimal solutions in every single run for 3, 4, 1, 1 and 13 cases respectively. Our SBTS algorithm is able to reach the best-known results with a perfect success rate for 2 cases, which is more than PLS and COVER but less than MN/TS, BLS and NuMVC.

Finally, it is more delicate to make a fully fair comparison of the computing time given that the compared algorithms are coded in different languages with different data structures, run on different platforms and more importantly lead to results of different quality for a number of graphs. As an indicative, we observe that to reach a result of equal quality, SBTS is more time consuming than MN/TS, COVER and NuMVC, but remains competitive with BLS and PLS.

3.4.2. Comparisons with two reference algorithms on CODE benchmark

The CODE benchmark is less popular than the DIMACS and BHOSLIB sets and few papers report results on the 11 CODE instances including (Andrade et al., 2012; Butenko et al., 2009; Etzion and Ostergard, 1998). However, we think the CODE instances are of interest since they come from real problems (code theory) and known to be relatively difficult. For this study, we

Table 6
Comparisons of SBTS with two typical MIS algorithms ILS and GLP (Andrade et al., 2012) on 32 representative instances. Each instance is solved 15 times and each run is allowed to a maximum of 10^8 iterations for SBTS and average arc (edge) scans limited to 2^{17} for ILS and GLP.

Graph		ILS		GLP		SBTS	
Name	f_{bk}	f_*	t (s)	f_*	t(s)	f_*	t (s)
brock400_1	27*	*25* (*25.0*)	11.00	27 (25.1)	22.00	27	59.21
brock400_2	29*	*25* (*25.0*)	11.00	29 (27.7)	20.00	29	25.39
brock400_3	31*	31 (27.0)	11.00	31	19.00	31	1.01
brock400_4	33*	33 (30.3)	11.00	33	16.00	33	0.86
brock800_1	23*	*21* (*21.0*)	60.00	23 (21.1)	112.00	23 (21.3)	1127.50
brock800_2	24*	*21* (*21.0*)	60.00	*21* (*21.0*)	111.00	24 (22.0)	787.93
brock800_3	25*	*22* (*22.0*)	60.00	25 (22.2)	111.00	25 (24.4)	763.61
brock800_4	26*	26 (21.3)	60.00	26 (21.7)	112.00	26 (25.7)	400.82
C2000.9	80	*77* (*76.9*)	103.00	*79* (*77.5*)	182.00	*78* (*77.1*)	1558.14
C4000.5	18*	18 (17.1)	1897.00	18	3708.00	18	1553.24
MANN_a45	345*	345 (344.5)	3.00	*344* (*343.8*)	5.00	345	17.99
MANN_a81	1100*	1100	10.00	*1098* (*1097.6*)	17.00	1100	22.29
frb30-15-1	30*	30	9.00	30	22.00	30	0.03
frb35-17-1	35*	35 (34.9)	13.00	35	32.00	35	1.93
frb40-19-1	40*	40	19.00	40	43.00	40	0.87
frb45-21-1	45*	45 (44.7)	27.00	45 (44.9)	62.00	45	34.50
frb50-23-1	50*	50 (48.9)	36.00	*49* (*48.6*)	82.00	50 (49.7)	250.32
frb53-24-1	53*	53 (51.5)	42.00	*52* (*51.3*)	93.00	*52* (*52.0*)	52.36
frb56-25-1	56*	*55* (*54.2*)	49.00	*55* (*54.1*)	111.00	*55* (*55.0*)	123.34
frb59-26-1	59*	*58* (*57.3*)	57.00	*57* (*57.0*)	126.00	59 (58.0)	436.48
frb100-40	100*	*96* (*95.3*)	249.00	*95* (*94.1*)	495.00	*96* (*95.4*)	862.18
1dc.1024	94	94 (93.1)	14.00	94 (93.1)	31.00	94	0.00
1dc.2048	172	172 (171.1)	32.00	172 (171.5)	74.00	172	0.06
1et.1024	171	171	8.00	171 (170.9)	16.00	171	0.16
1et.2048	316	316	16.00	316	40.00	316	4.13
1tc.1024	196	196	8.00	196	18.00	196	0.03
1tc.2048	352	352	15.00	352	37.00	352	19.10
1zc.1024	112	112 (111.1)	10.00	112	28.00	112	43.44
1zc.2048	198	198 (197.3)	22.00	198 (197.8)	65.00	198	56.94
1zc.4096	379	379 (367.7)	51.00	379 (374.4)	160.00	379 (372.6)	99.83
2dc.1024	16	16	50.00	16	198.00	16	0.01
2dc.2048	24	24 (23.8)	165.00	24	527.00	24	3.07

adopt as our reference two most recent algorithms that use the CODE benchmark: ILS and GLP (Andrade et al., 2012).[3] Both ILS and GLP are run on a computer equipped with a 3.16 GHz Intel Core 2 Duo CPU and 4 GB of RAM. Unlike the reference studies of the last section which make 100 independent runs, the results of ILS and GLP reported in Andrade et al. (2012) are based on 15 runs. The stop condition for each run is the average arc (edge) scans limited to 2^{17} (Andrade et al., 2012).

In addition to the 11 CODE instances, the authors of Andrade et al. (2012) also report results on a subset of 33 DIMACS and 9 BHOSLIB instances (8 instances as they are introduced in Section 3.1 plus one additional challenging instance frb100-40). To make a fair comparison, we re-run SBTS 15 times (like ILS and GLP) on these 11 CODE instances and the 42 DIMACS/BHOSLIB instances. Since there is no evident way to relate the number of average arc (edge) scans used by ILS and GLP to the number of iterations used by SBTS, SBTS is run under the stop condition given in Section 3.4. To report the results, we only retain the 12 (out of 33) most difficult graphs for the DIMACS set while keeping the 11 CODE instances and the 9 BHOSLIB instances since for the remaining instances, all three compared algorithms reach the same results.

Table 6 shows the comparison of ILS (columns 3 and 4), GLP (columns 5 and 6) and SBTS (columns 7 and 8): the best result f_*, followed by the average results f_{avg} given in parenthesis over 15 runs, and the average time in seconds $t(s)$ over the successful runs.

From Table 6, one observes that ILS and GLP cannot reach the best-known results for 9 (entries in italic) out of 21 difficult DIMACS and BHOSLIB instances while SBTS fails to reach the best-known result for 4 instances with its 15 runs (corresponding to the cases where its success rate is lower than 15%, see Tables 2 and 3). Furthermore, the average results of SBTS on the instances which cannot be solved with a 100% success rate are all better than ILS and GLP except for C2000.9 where the average of SBTS (77.1) is worse than GLP (77.5) but better than ILS (76.9). We do not emphasize the computing time since the compared algorithms give several results of different quality (f_*).

For the CODE set, the three compared algorithms can achieve the best-known result for the 11 instances. Furthermore, ILS and GLP reach the best results with a 100% success rate for 5 and 6 cases respectively against 10 cases for our SBTS algorithm (i.e., except 1zc.4096).

To conclude, the comparative results indicate that the proposed SBTS algorithm is quite competitive with the reference algorithms not only for the best obtained solutions but also for the average solutions. SBTS seems to be the most comprehensive approach to solve the DIMACS, BHOSLIB and CODE instances with multiple topologies and densities.

4. Analysis of SBTS

Now we turn our attention to an analysis of the important feature of the proposed SBTS algorithm: the selection rule for intensification (Section 2.7) and the analysis of tabu tenure (Section 2.6).

[3] The study of Andrade et al. (2012) uses two other test sets (MESH and ROAD) that includes very large sparse graphs. Unfortunately, these instances cannot be loaded into our computer due to the matrix representation used by SBTS.

4.1. Influence of the selection rule for intensification

As described in Section 2.7, SBTS uses a Selection Rule for the NS_1 neighborhood for (1,1)-swap moves. In this section, we carry out an experiment to verify the importance of this dedicated Selection Rule compared to a random selection rule. For this purpose, we create a variant of SBTS (denoted by $SBTS_{random}$) by replacing its Selection Rule with a random selection rule. With $SBTS_{random}$, when NS_1 offers multiple eligible vertices, one of them is picked at random and used by the (1,1)-swap move.

For this experiment, we run SBTS and $SBTS_{random}$ 100 times on each of the 32 instances (DIMACS, BHOSLIB, CODE) of Section 3.4.2 under the same condition as before. The results are given in Table 7 which shows for each algorithm the best result f_*, the average result f_{avg} (in parenthesis) and the average time in second $t(s)$ to reach the best result f_*. From Table 7, one notices that SBTS performs better than $SBTS_{random}$ both in terms of the best result f_* and the average result f_{avg}. Precisely, SBTS achieves the best-known result for all the instances except frb100-40 while $SBTS_{random}$ attains the best-known result for only 25 cases out of 32 instances. Besides, SBTS has a perfect success rate for 20 cases against 13 cases for $SBTS_{random}$. This experiment demonstrates the usefulness of using the proposed Selection Rule to explore the NS_1 neighborhood.

4.2. Analysis of the tabu tenure technique

Recall that the tabu tenure tt is set differently according to $k=1$ or $k>1$ (see Section 2.6). In this section, we carry out an experiment to show the usefulness of this tuning technique. For this purpose, we adopt for the case $k>1$ the same tabu tenure as for the case $k=1$ and denote the resulting variant by $SBTS_{unique}$. We use $SBTS_{unique}$ to solve 100 times each of the 32 instances under the same condition as before.

Table 8 shows the comparative results of $SBTS_{unique}$ (columns 3 and 4) and SBTS (columns 5 and 6). For each algorithm, we show the best result f_* followed by the average result f_{avg} (in parenthesis) and the average time in second $t(s)$. We observe that contrary to SBTS which finds all the best-known results except frb100-40 instance, $SBTS_{unique}$ fails to do so for 3 instances. SBTS has a perfect success rate of reaching the best-known result for 20 cases against 17 cases for $SBTS_{unique}$. This study shows the interest of the adopted tabu tenue technique and confirms the importance of tuning the tabu tenure carefully.

5. Conclusion

In this paper, we have presented SBTS, a general and unified swap-based tabu search algorithm for solving the maximum independent set problem. The proposed algorithm explores the search space by a dynamic alternation between intensification and diversification steps. The search process is driven by the $(k,1)-$swap operator combined with specific rules to examine four different neighborhoods. For the purpose of intensification, SBTS uses (0,1)-swap to improve the solution and (1,1)-swap to make side-walks with specific selection rules. To overcome local optima, SBTS adopts an adaptive perturbation strategy which applies either a (2,1)-swap for a weak perturbation or a $(k,1)$-swap $(k>2)$ for a strong perturbation. A tabu mechanism is also employed to prevent the search from short-term cycles.

We have tested the proposed algorithm on two sets of 120 well-known instances (DIMACS and BHOSLIB) with multiple

Table 7
Comparisons of $SBTS_{random}$ with SBTS.

Graph		$SBTS_{random}$		SBTS	
Name	f_{bk}	f_* (f_{avg})	t (s)	$f_*(f_{avg})$	t (s)
brock400_1	27*	27	118.95	27	66.40
brock400_2	29*	29	18.81	29	20.24
brock400_3	31*	31	4.63	31	1.77
brock400_4	33*	33	0.81	33	0.83
brock800_1	23*	23 (21.40)	874.88	23 (21.52)	963.63
brock800_2	24*	24 (22.29)	823.12	24 (22.29)	784.36
brock800_3	25*	25 (23.89)	937.43	25 (24.16)	651.07
brock800_4	26*	26 (25.65)	671.61	26 (25.90)	421.60
C2000.9	80	*78 (76.30)*	2117.21	80 (77.29)	1515.57
C4000.5	18*	18 (17.99)	5482.05	18	1553.24
MANN_a45	345*	345 (344.06)	302.25	345	27.56
MANN_a81	1100*	*1098 (1098.00)*	452.82	1100	22.70
frb30-15-1	30*	30	0.16	30	0.03
frb35-17-1	35*	35	48.40	35	0.69
frb40-19-1	40*	40	36.51	40	0.24
frb45-21-1	45*	45 (44.97)	327.35	45	23.00
frb50-23-1	50*	50 (48.92)	394.40	50 (49.75)	368.69
frb53-24-1	53*	*52 (51.25)*	609.09	53 (52.03)	498.36
frb56-25-1	56*	*55 (54.11)*	594.24	56 (55.07)	551.64
frb59-26-1	59*	*58 (57.04)*	890.92	59 (58.02)	261.14
frb100-40	100*	*95 (94.22)*	2094.28	*97 (95.48)*	862.18
1dc.1024	94	94	0.09	94	0.03
1dc.2048	172	172	0.20	172	0.17
1et.1024	171	171	0.02	171	0.07
1et.2048	316	316 (315.68)	116.37	316	6.59
1tc.1024	196	196	0.02	196	0.02
1tc.2048	352	352 (351.96)	110.99	352	6.52
1zc.1024	112	112 (111.99)	44.57	112 (111.99)	29.79
1zc.2048	198	198 (197.40)	172.03	198	81.69
1zc.4096	379	*377 (356.02)*	727.18	379 (372.66)	99.83
2dc.1024	16	16	0.35	16	0.03
2dc.2048	24	24	188.08	24	6.98

Table 8
Comparisons of $SBTS_{unique}$ with SBTS.

Graph		$SBTS_{unique}$		SBTS	
Name	f_{bk}	f_*	t (s)	f_*	t (s)
brock400_1	27*	27 (26.98)	59.04	27	66.40
brock400_2	29*	29	6.07	29	20.24
brock400_3	31*	31	0.71	31	1.77
brock400_4	33*	33	0.23	33	0.83
brock800_1	23*	23 (21.46)	452.06	23 (21.52)	963.63
brock800_2	24*	24 (22.44)	653.53	24 (22.29)	784.36
brock800_3	25*	25 (24.10)	739.92	25 (24.16)	651.07
brock800_4	26*	26 (25.95)	458.71	26 (25.90)	421.60
C2000.9	80	*79 (76.85)*	605.79	80 (77.29)	1515.57
C4000.5	18*	18	1981.74	18	1553.24
MANN_a45	345*	345 (344.25)	318.49	345	27.56
MANN_a81	1100*	1100 (1098.41)	1680.60	1100	22.70
frb30-15-1	30*	30	0.03	30	0.03
frb35-17-1	35*	35	0.47	35	0.69
frb40-19-1	40*	40	0.74	40	0.24
frb45-21-1	45*	45	68.51	45	23.00
frb50-23-1	50*	50 (49.55)	356.14	50 (49.75)	368.69
frb53-24-1	53*	53 (52.03)	438.46	53 (52.03)	498.36
frb56-25-1	56*	56 (55.09)	453.20	56 (55.07)	551.64
frb59-26-1	59*	59 (57.86)	777.11	59 (58.02)	261.14
frb100-40	100*	*97 (95.28)*	849.67	*97 (95.48)*	862.18
1dc.1024	94	94	0.03	94	0.03
1dc.2048	172	172	0.07	172	0.17
1et.1024	171	171	0.22	171	0.07
1et.2048	316	316	27.23	316	6.59
1tc.1024	196	196	0.01	196	0.02
1tc.2048	352	352	13.73	352	6.52
1zc.1024	112	112	52.26	112 (111.99)	29.79
1zc.2048	198	198 (197.89)	178.46	198	81.69
1zc.4096	379	*377 (365.26)*	377.07	379 (372.66)	99.83
2dc.1024	16	16	0.01	16	0.03
2dc.2048	24	24	2.78	24	6.98

topologies and densities. Computational results show that SBTS competes favorably with 5 state-of-the-art algorithms in the literature. In particular, SBTS can achieve the best-known results for all the 120 instances. An additional test of SBTS on a set of 11 instances from code theory has confirmed its competitiveness relative to two other reference methods.

Even though the proposed approach achieves competitive results on the three benchmarks, one observes that some best results can only be reached occasionally. More studies are needed to improve the stability and search capacity of the approach. One possibility would be to introduce multiple search strategies and apply them dynamically and adaptively according to learned guiding information. Another possibility would be to combine SBTS with the memetic search framework where a meaningful solution recombination mechanism must be sought.

Acknowledgment

We are grateful to the anonymous referees for valuable suggestions and comments which helped us to improve the paper and to Dr. R.F. Werneck for kindly sending us the CODE benchmarks. The work is partially supported by the following projects: RaDaPop and LigeRO (2009–2013, Pays de la Loire Region) and PGMO project (2014–2015, Jacques Hadamard Mathematical Foundation). Support for Yan Jin from the China Scholarship Council is also acknowledged.

Appendix A. Information updating procedure

After each $(k,1)$-swap, SBTS updates the Mapping degree, Expanding degrees and Diversifying degree of some vertices as well as the associated neighborhoods NS_k $(k=0,1,2,>2)$. We explain below the updating procedure.

Suppose that $v_i \in NS_k$ $(k=0,1,2,>2)$ is swapped with its k adjacent vertices $v_j \in S$. Let $v_{ia} \in V \setminus S$ be any adjacent vertex of v_i (i.e., $v_{ia} \in V \setminus S$, $\{v_{ia}, v_i\} \in E$). For each v_j, let $v_{ja} \in V \setminus S$ be any adjacent vertex of v_j (i.e., $v_{ja} \in V \setminus S$, $\{v_{ja}, v_j\} \in E$). Then the updating procedure realizes the following operations.

1. First, for v_i and each v_j: Since v_i moves from NS_k to S, its Expanding Degree is initially set to 0. Since v_j is removed from S, its Mapping Degree and Diversifying Degree is initially set to 0.
2. Then, for each vertex v_{ia}: its Diversifying Degree decreases by 1 and its Mapping Degree increases by 1. Meanwhile, the Mapping Degree of v_j increases by 1.
   The Expanding Degree of v_i increases by 1 if the Mapping Degree of v_{ia} increases from 0 to 1 (including v_j) while the Expanding Degree of v_i decreases by 1 if the Mapping Degree of v_{ia} increases from 1 to 2.
   When the Mapping Degree of v_{ia} changes from k to $k+1$ $(k=0,1,2)$, v_{ia} moves from NS_k to NS_{k+1} for $k=0,1$ and to $NS_{>2}$ for $k=2$. If $k>2$, v_{ia} stays in $NS_{>2}$. Notice that v_j belongs now to NS_1.
3. Finally, for each v_j and its adjacent vertices v_{ja} in $V \setminus S$: The Diversifying Degree of v_{ja} increases by 1 while the Mapping Degree of v_{ja} decreases by 1. Meanwhile, the Diversifying Degree of v_j increases by 1 for each v_{ja}.
   For any $v'_j \in S$ adjacent to v_{ja} $(\{v'_j, v_{ja}\} \in E, v'_j \neq v_j)$, its Expanding Degree increases by 1 if the Mapping Degree of v_{ja} decreases from 2 to 1.
   According to the decrease of the Mapping Degree of v_{ja} from $k+1$ to k $(k=0,1,2)$, v_{ja} displaces from NS_{k+1} ($NS_{>2}$ if $k>2$) to NS_k. If $k>2$, v_{ja} stays in $NS_{>2}$.

Notice that no v_j is swapped out from S if a (0,1)-swap is applied. In this case, only v_i and its adjacent vertices in $V \setminus S$ need to be updated.

This procedure can be efficiently performed in $O(k+|\{v_{ia}\}|+k|\{v_{ja}\}|)$. For Fig. 1, if vertex 9 is added into the solution $S=\{1, 4, 6, 8\}$ after a (0,1)-swap, the Mapping Degree of its neighbors $\{5, 7, 10\}$ becomes $K^M(5)=2, K^M(7)=4$ and $K^M(10)=3$. The Expanding Degree of vertex 4 becomes $K^E(4)=1$. The new neighborhoods become: $NS_0=\emptyset$, $NS_1=\{2,3\}$, $NS_2=\{5\}$ and $NS_{>2}=\{7,10\}$.

References

Andrade, D.V., Resende, M.G.C., Werneck, R.F., 2012. Fast local search for the maximum independent set problem. J. Heurist. 18 (4), 525–547.
Battiti, R., Protasi, M., 2001. Reactive local search for the maximum clique problem. Algorithmica 29 (4), 610–637.
Benlic, U., Hao, J.K., 2013. Breakout local search for maximum clique problems. Comput. Oper. Res. 40 (1), 192–206.
Bomze, I.M., Budinich, M., Pardalos, P.M., Pelillo, M., 1999. The maximum clique problem. In: Du, D.Z., Pardalos, P.M. (Eds.), Handbook of Combinatorial Optimization, suppl. vol. A. Kluwer Academic Publishers, Boston, pp. 1–74.
Brunato, M., Battiti, R., 2011. R-EVO: a reactive evolutionary algorithm for the maximum clique problem. IEEE Trans. Evol. Comput. 15 (6), 770–782.
Butenko, S., Pardalos, P., Sergienko, I., Shylo, V.P., Stetsyuk, P., 2009. Estimating the size of correcting codes using extremal graph problems. In: Pearce, C., Hunt, E. (Eds.), Structure and Applications, Optimization and its Application, vol. 32, pp. 227–243 (Chapter 12).
Cai, S., Su, K., Luo, C., Sattar, A., 2013. NuMVC: an efficient local search algorithm for minimum vertex cover. J. Artif. Intell. Res. 46, 687–716.
Carraghan, R., Pardalos, P.M., 1990. An exact algorithm for the maximum clique problem. Oper. Res. Lett. 9 (6), 375–382.
Dorne, R., Hao, J.K., 1998. Tabu search for graph coloring, T-colorings and set T-colorings. In: Voss, S., et al. (Eds.), Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization. Kluwer, Boston, pp. 77–92 (Chapter 6).
Etzion, T., Ostergard, P.R.J., 1998. Greedy and heuristic algorithms for codes and colorings. IEEE Trans. Inf. Theory 44 (1), 382–388.
Friden, C., Hertz, A., de Werra, D., 1989. Stabulus: a technique for finding stable sets in large graphs with tabu search. Computing 42 (1), 35–44.
Galinier, P., Hao, J.K., 1999. Hybrid evolutionary algorithms for graph coloring. J. Comb. Optim. 3 (4), 379–397.
Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, San Francisco.
Geng, X., Xu, J., Xiao, J., Pan, L., 2007. A simple simulated annealing algorithm for the maximum clique problem. Inf. Sci. 177 (22), 5064–5071.
Glover, F., Laguna, M., 1997. Tabu Search. Kluwer Academic Publishers, Dordrecht, The Netherlands.
Grosso, A., Locatelli, M., Pullan, W., 2008. Simple ingredients leading to very efficient heuristics for the maximum clique problem. J. Heurist. 14 (6), 587–612.
Hansen, P., Mladenović, N., Urošević, D., 2004. Variable neighborhood search for the maximum clique. Discret. Appl. Math. 145 (1), 117–125.
Johnson, D.S., Trick, M.A., 1996. Cliques, coloring and satisfiability: second DIMACS implementation challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol 26, AMS, Providence, RI.
Katayama, K., Hamamoto, A., Narihisa, H., 2005. An effective local search for the maximum clique problem. Inf. Process. Lett. 95 (5), 503–511.
Karp, R.M., 1972. Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (Eds.), Complexity of Computer Computations. Plenum Press, New York, pp. 85–103.
Li, C., Quan, Z., 2010. Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In: Proceedings of ICTAI-10. pp. 344–351.
Lourenço, H.R., Martin, O.C., Stützle, T., 2003. Iterated local search. In: Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol. 57, Kluwer Academic Publishers, Dordrecht, pp. 321–353.
McCreesh, C., Prosser, P., 2013. Multi-threading a state-of-the-art maximum clique algorithm. Algorithms 6, 618–635.
Östergård, P.R.J., 2002. A fast algorithm for the maximum clique problem. Discret. Appl. Math. 120 (1), 97–207.
Pullan, W., 2006. Phased local search for the maximum clique problem. J. Comb. Optim. 12 (3), 303–323.
Pullan, W., 2008. Approximating the maximum vertex/edge weighted clique using local search. J. Heurist. 14, 117–134.
Pullan, W., Mascia, F., Brunato, M., 2011. Cooperating local search for the maximum clique problem. J. Heurist. 17, 181–199.
Richter, S., Helmert, M., Gretton, C., 2007. A stochastic local search approach to vertex cover. In: Proceedings of KI-07, pp. 412–426.
San Segundo, P., Rodríguez-Losada, D., Jiménez, A., 2011. An exact bit-parallel algorithm for the maximum clique problem. Comput. Oper. Res. 38 (2), 571–581.
Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., Dueck, G., 2000. Record breaking optimization results using the ruin and recreate principle. J. Comput. Phys. 159, 139–171.

Sloane, N.J.A., 2000. Challenge problems: independent sets in graphs. ⟨http://neilsloane.com/doc/graphs.html⟩.
Tomita, E., Kameda, T., 2007. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. J. Glob. Optim. 37 (1), 95–111.
Wu, Q., Hao, J.K., Glover, F., 2012. Multi-neighborhood tabu search for the maximum weight clique problem. Ann. Oper. Res. 196 (1), 611–634.
Wu, Q., Hao, J.K., 2013. An adaptive multistart tabu search approach to solve the maximum clique problem. J. Comb. Optim. 26 (1), 86–108.
Wu, Q., Hao, J.K., 2014. A review on algorithms for maximum clique problems. Eur. J. Oper. Res.. Submitted and revised manuscript. ⟨http://www.info.univ-angers.fr/pub/hao/papers/WuHaoClique2014.pdf⟩.
Xu, K., Boussemart, F., Hemery, F., Lecoutre, C., 2005. A simple model to generate hard satisfiable instances. In: Proceedings of IJCAI-05, pp. 337–342.
Zhang, Q., Sun, J., Tsang, E., 2005. An evolutionary algorithm with guided mutation for the maximum clique problem. IEEE Trans. Evol. Comput. 9 (2), 192–200.