European Journal of Operational Research 257 (2017) 41–54

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

PUSH: A generalized operator for the Maximum Vertex Weight Clique Problem

Yi Zhou[a], Jin-Kao Hao[a,b,*], Adrien Goëffon[a]

[a] LERIA, Université d'Angers, 2 Boulevard Lavoisier, Angers 49045, France
[b] Institut Universitaire de France, Paris, France

ARTICLE INFO

Article history:
Received 24 March 2016
Accepted 26 July 2016
Available online 2 August 2016

Keywords:
Local search
Cliques
Tabu search
Heuristics

ABSTRACT

The Maximum Vertex Weight Clique Problem (MVWCP) is an important generalization of the well-known NP-hard Maximum Clique Problem. In this paper, we introduce a generalized move operator called *PUSH*, which generalizes the conventional *ADD* and *SWAP* operators commonly used in the literature and can be integrated in a local search algorithm for MVWCP. The *PUSH* operator also offers opportunities to define new search operators by considering dedicated candidate push sets. To demonstrate the usefulness of the proposed operator, we implement two simple tabu search algorithms which use *PUSH* to explore different candidate push sets. The computational results on 142 benchmark instances from different sources (DIMACS, BHOSLIB, and Winner Determination Problem) indicate that these algorithms compete favorably with the leading MVWCP algorithms.

1. Introduction

Given an undirected graph $G = (V, E, w)$ with vertex set V and edge set $E \subseteq V \times V$, let $w : V \rightarrow \mathbb{R}^+$ be a weighting function that assigns to each vertex $v \in V$ a positive value w_v. A clique $C \subseteq V$ of G is a subset of vertices such that its induced subgraph is complete, i.e., every two vertices in C are pairwise adjacent in G $(\forall u, v \in C, (u, v) \in E)$. For a clique C of G, its weight is given by $W(C) = \sum_{v \in C} w_v$. The Maximum Vertex Weight Clique Problem (MVWCP) is to determine a clique C^* of maximum vertex weight.

MVWCP is an important generalization of the classical Maximum Clique Problem (MCP) (Wu & Hao, 2015a). Indeed, when the vertices of V are assigned the unit weight of 1, MVWCP is equivalent to MCP which is to find a clique C^* of maximum cardinality. Given that the decision version of MCP is NP-complete (Karp, 1972), the generalized MVWCP problem is at least as difficult as MCP. Consequently solving MVWCP represents an imposing computational challenge in the general case. Note that MVWCP is different from another MCP variant – the Maximum Edge Weight Clique Problem (Alidaee, Glover, Kochenberger, & Wang, 2007; van Dijkhuizen & Faigle, 1993) where a clique C^* of maximum *edge* weight is sought.

Like MCP which has many practical applications, MVWCP can be used to formulate and solve some relevant problems in diverse domains. For example, in computer vision, MVWCP can be used to solve image matching problems (Ballard & Brown, 1982). In combinatorial auctions, the Winner Determination Problem can be recast as MVWCP and solved by MVWCP algorithms (Wu & Hao, 2015b; 2016).

Given the significance of MVWCP, much effort has been devoted to design various algorithms for solving the problem over the past decades. On the one hand, there are a variety of exact algorithms which aim to find optimal solutions. For instance, in 2001, Östergård (2001) presented a branch-and-bound (B&B) algorithm where the vertices are processed according to the order provided by a vertex coloring of the given graph. This MVWCP algorithm is in fact an adaptation of an existing B&B algorithm designed for MCP (Östergård, 2002). Kumlander (2004) introduced a backtrack tree search algorithm which also relies on a heuristic coloring-based vertex order. Warren and Hicks (2006) exposed three B&B algorithms which use weighted clique covers to generate upper bounds and branching rules. Wu and Hao (2016) developed an algorithm which introduces new bounding and branching techniques using specific vertex coloring and sorting. Fang, Li, and Xu (2016) presented an algorithm which uses Maximum Satisfiability Reasoning as a bounding technique. On the other hand, local search heuristics constitute another popular approach to find high-quality sub-optimal or optimal solutions in a reasonable computing time. Mannino and Stefanutti (1999) proposed a tabu search method based on edge projection and augmenting

* Corresponding author at: LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France. Fax: +33241735073.

E-mail addresses: zhou@info.univ-angers.fr (Y. Zhou), hao@info.univ-angers.fr (J.-K. Hao), goeffon@info.univ-angers.fr (A. Goëffon).

http://dx.doi.org/10.1016/j.ejor.2016.07.056
0377-2217/

sequence. Bomze, Pelillo, and Stix (2000) formulated MVWCP as a continuous problem which is solved by a parallel algorithm using a distributed computational network model. Busygin (2006) exposed a trust region algorithm. The same year, Singh and Gupta (2006) introduced a hybrid method combining genetic algorithm, a greedy search and the exact algorithm of Carraghan and Pardalos (1990). Pullan (2008) adapted the Phase Local Search for the classical MCP to MVWCP. Wu, Hao, and Glover (2012) introduced a tabu search algorithm integrating multiple neighborhoods. Benlic and Hao (2013) presented the Breakout Local Search algorithm which also explores multiple neighborhoods and applies both directed and random perturbations. Recently, Wang, Hao, Glover, Lü, and Wu (2016) reformulated MVWCP as a Binary Quadratic Program (BQP) which was solved by a probabilistic tabu search algorithm designed for BQP.

As shown in the literature, local search represents the most popular and the dominating approach for solving MVWCP heuristically. Typically, local search heuristics explore the search space by iteratively transforming the incumbent solution into another solution by means of some move (or transformation) operators. Existing heuristic algorithms for MVWCP (Benlic & Hao, 2013; Pullan, 2008; Wang et al., 2016; Wu & Hao, 2015b; Wu et al., 2012) are usually based on two popular move operators for search intensification: (1) *ADD* which inserts a vertex to the incumbent solution (a feasible clique), and (2) *SWAP* which exchanges a vertex in the clique against a vertex out of the clique. In studies like Benlic and Hao (2013); Wu et al. (2012), another operator called *DROP* was also used, which simply removes a vertex from the current clique. The algorithms using these operators have reported remarkable results on a large range of benchmark problems. Still as we show in this work, the performance of local search algorithms could be further improved by employing more powerful search operators.

This work introduces a generalized move operator called *PUSH*, which inserts one vertex into the clique and removes $k \geq 0$ vertices from the clique to maintain the feasibility of the transformed clique. The proposed *PUSH* operator shares similarities with some restart and perturbation operators used in MCP and MVWCP algorithms like Benlic and Hao (2013); Grosso, Locatelli, and Pullan (2008) and finds its origin in these previous studies. Meanwhile, as we show in this work, the *PUSH* operator not only generalizes the existing *ADD* and *SWAP* operators, but also offers the possibility of defining additional clique transformation operators. Indeed, dedicated local search operators can be obtained by customizing the set of candidate vertices considered by *PUSH*. Such alternative operators can then be employed in a search algorithm as a means of intensification or diversification.

To assess the usefulness of the *PUSH* operator, we experiment two restart tabu search algorithms (ReTS-I and ReTS-II), which explore different candidate push sets for push operations. ReTS-I operates on the largest possible candidate push set while ReTS-II works with three customized candidate push sets. Both algorithms share a probabilistic restart mechanism. The proposed approach is assessed on three sets of well-known benchmarks (DIMACS, BHOSLIB, and Winner Determination Problem) of a total of 142 instances. The computational results indicate that both ReTS-I and ReTS-II compete favorably with the leading MVWCP algorithms of the literature. Moreover, the generality of the *PUSH* operator could allow it to be integrated within any local search algorithm to obtain enhanced performances.

The paper is organized as follows. Section 2 formally introduces the *PUSH* operator. Section 3 presents the two push-based tabu search algorithms. Section 4 reports our experimental results and comparisons with respect to state-of-the-art algorithms. Section 5 is dedicated to an experimental analysis of the restart strategy while conclusions and perspectives are given in Section 6.

2. *PUSH*: a generalized operator for MVWCP

2.1. Preliminary definitions

Let $G = (V, E, w)$ be an input graph as defined in the introduction, $C \subseteq V$ a feasible solution (i.e., a clique) such that any two vertices in C are linked by an edge in E (throughout the paper, C is used to designate a clique), and $v \in V$ an arbitrary vertex. We introduce the following notations:

- $N(v)$ and $\bar{N}(v)$ denote respectively the set of adjacent and non-adjacent vertices of a vertex v in V, i.e., $N(v) = \{u : (v, u) \in E\}$ and $\bar{N}(v) = \{u : (v, u) \notin E\}$.
- $N_C(v)$ and $\bar{N}_C(v)$ denote respectively the set of adjacent and non-adjacent vertices of a vertex v in C, i.e., $N_C(v) = C \cap N(v)$ and $\bar{N}_C(v) = C \cap \bar{N}(v)$.
- Ω is the search space including all the cliques of G, i.e., $\Omega = \{C \subseteq V : \forall v, u \in C, v \neq u, (v, u) \in E\}$.
- m is a move operator which transforms a clique to another one. We use $C \oplus m$ to designate the clique $C' = C \oplus m$ obtained by applying the move operator m to C. C' is called a neighbor solution (or neighbor clique) of C.
- N_m is the set of neighbor solutions that can be obtained by applying m to an incumbent solution C.

The weight $W(C) = \sum_{v \in C} w_v$ of a solution (clique) $C \in \Omega$ is used to measure its quality (fitness). For two solutions C and C' in Ω, C' is said to be better than C if $W(C') > W(C)$. The weight $W(C^*)$ of the best solution ever found by a search procedure is abbreviated as W^*.

2.2. Motivations for the PUSH operator

As shown in the literature, local search is the dominating approach for tackling MVWCP (see for example, Benlic and Hao (2013); Pullan (2008); Wu and Hao (2015b); Wu et al. (2012)). Local search typically explores the search space by iteratively transforming an incumbent solution C to a neighbor solution (often of better quality) by means of the following move operators:

- *ADD* extends C with a vertex $v \in V \setminus C$ which is necessarily adjacent to all the vertices in C. Each application of *ADD* always increases the weight of C and leads to a better solution.
- *SWAP* exchanges a vertex $v \in V \setminus C$ with another vertex $v' \in C$, v being necessarily adjacent to all vertices in C except v'. Each application of *SWAP* can increase the weight of C (if $w_v > w_{v'}$), keep its weight unchanged (if $w_v = w_{v'}$) or decrease the quality of C (if $w_v < w_{v'}$).

In some cases like Benlic and Hao (2013); Wu et al. (2012), a third move operator (*DROP*) was also employed which simply removes a vertex from C (thus always leading to a worse neighbor solution).

Generally, local search for MVWCP aims to reach solutions of increasing quality by iteratively moving from the incumbent solution to a neighbor solution. This is typically achieved by applying *ADD* whenever it is possible to increase the weight of the clique, applying *SWAP* when no vertex can be added to the clique and occasionally calling for *DROP* to escape local optima.

However, as both *ADD* and *SWAP* have a prerequisite on the operating vertex in $V \setminus C$, these operators may miss improving solutions in some cases. To illustrate this point, we consider the example of Fig. 1 where vertex weights are indicated in brackets next to the vertex labels. As shown in Fig. 1(a), $C = \{a, b, c\}$ is a clique with a total weight of 12. Since vertex d is neither adjacent to a nor b, d cannot join clique C by means of the *ADD* and *SWAP* operators. Meanwhile, one observes that if we insert vertex d into the

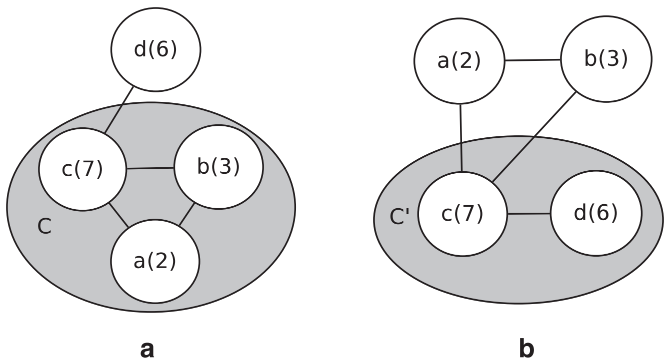

Fig. 1. An example which shows that a better solution can be reached by the *PUSH* operator, but cannot be attained by the traditional *ADD* and *SWAP* operators.

clique and remove both vertices a and b, we obtain a new clique C' (Fig. 1(b)) of weight of 13, which is better than C.

Inspired by this observation, the *PUSH* operator proposed in this work basically transforms C by pushing a vertex v taken from a *dedicated* subset of $V\backslash C$ into C and removing, if needed, one or more vertices from C to re-establish solution feasibility. Indeed, when the added vertex v is not adjacent to all the vertices in C, the vertices of C which are not adjacent to v (i.e., the vertices in the set $\bar{N}_C(v)$ as defined in Section 2.1) need to be removed from C to maintain the feasibility of the new solution. In the above example, C can be transformed to a better solution by pushing vertex d into the clique and then expelling both a and b.

2.3. Definition of the PUSH operator

Let C be a clique, and v an arbitrary vertex which does not belong to C ($v \in V \setminus C$). $PUSH(v, C)$ (or $PUSH(v)$ for short if the current clique does not need to be explicitly emphasized) generates a new clique by first inserting v into C and then removing any vertex $u \in C$ such that $(u, v) \notin E$ (i.e., $u \in \bar{N}_C(v)$, see Section 2.1).

Formally, the neighbor clique C' after applying $PUSH(v)$ ($v \in V \setminus C$) to C is given by:

$$C' = C \oplus PUSH(v) = C \setminus \bar{N}_C(v) \cup \{v\}$$

Consequently, the set of neighbor cliques induced by $PUSH(v)$ in the general case, denoted by N_{PUSH} is given by:

$$N_{PUSH} = \bigcup_{v \in V \setminus C} \{C \oplus PUSH(v)\} \tag{1}$$

For each neighbor solution $C' = C \oplus PUSH(v)$ generated by a $PUSH(v, C)$ move, we define the move gain (denoted by δ_v) as the variation in the objective function value between C' and C:

$$\delta_v = W(C') - W(C) = w_v - \sum_{u \in \bar{N}_C(v)} w_u \tag{2}$$

Thus, a positive (negative) move gain indicates a better (worse) neighbor solution C' compared to C while the zero move gain corresponds to a neighbor solution of equal quality.

Typically, a local search algorithm makes its decision of moving from the incumbent solution to a neighbor solution based on the move gain information at each iteration. In order to be able to efficiently compute the move gains of neighbor solutions, we present in Section 3.5 fast streamlining evaluation techniques with the help of dedicated data structures.

One notices that *PUSH* shares similarities with some customized restart or perturbation operators used in Benlic and Hao (2013); Grosso et al. (2008) and finds its origin in these previous studies. In the iterated local search algorithm designed for MCP by Grosso et al. (2008), the clique delivered at the end of each local optimization stage is perturbed by insertion of a random vertex and serves then as a new starting point for the next stage of local optimization. In the BLS algorithm for MCP and MVWCP presented by Benlic and Hao (2013), each random perturbation adds a vertex such that the resulting clique must satisfy a quality threshold. In these two previous studies, clique feasibility is established by a repair process which removes some vertices after each vertex insertion.

2.4. Special cases of PUSH

From the general definition of *PUSH* given in the last section, we can customize the move operator by identifying a dedicated vertex subset of $V\backslash C$ called *candidate push set* (CPS) that provides the candidate vertices for *PUSH*. We first discuss two special cases by considering non-adjacency information convoyed by $\bar{N}_C(v)$ (see the notations introduced in Section 2.1).

- If CPS is given by $A = \{v : |\bar{N}_C(v)| = 0,\ v \in V \setminus C\}$, then *PUSH* is equivalent to *ADD*.
- If CPS is given by $B = \{v : |\bar{N}_C(v)| = 1,\ v \in V \setminus C\}$, then *PUSH* is equivalent to *SWAP*.

We can also use other information like move gain to constrain the candidate push set, as illustrated by the following examples.

(1) If CPS is given by $M_1 = \{v : \delta_v > 0, v \in V \setminus C\}$, then *PUSH* always leads to a neighbor solution better than C.
(2) If CPS is given by $M_2 = \{v : \delta_v \le 0, |\bar{N}_C(v)| = 1, v \in V\backslash C\}$, then *PUSH* exchanges one vertex in $V\backslash C$ with one vertex in C, leading to a solution of equal or worse quality relative to C.
(3) If CPS is given by $M_3 = \{v : \delta_v \le 0, |\bar{N}_C(v)| > 1, v \in V\backslash C\}$, then *PUSH* inserts one vertex into C and removes at least two vertices from C, leading to a solution of equal or worse quality relative to C.
(4) If CPS is given by $V\backslash C$, then the candidate push set is not constrained. One notices that $V \setminus C = M_1 \cup M_2 \cup M_3$.

An example of these special cases is provided in Fig. 2.

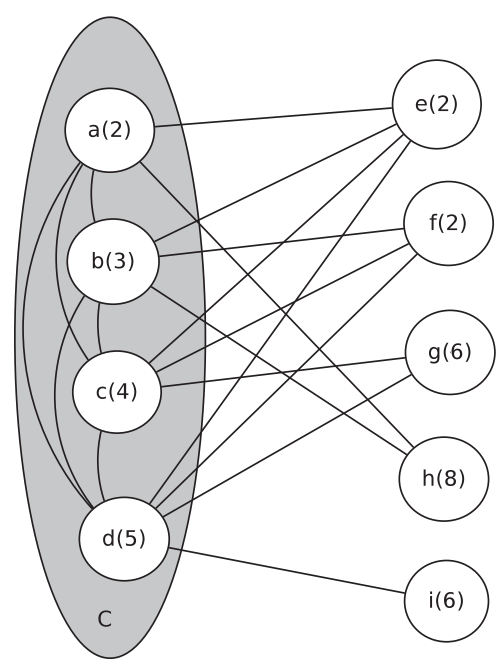

Fig. 2. A simple graph labeled with vertex weights in brackets. The current clique is $C = \{a, b, c, d\}$, $W(C) = 2 + 3 + 4 + 5 = 14$, $\bar{N}_C(e) = \emptyset$, $\bar{N}_C(f) = \{a\}$, $\bar{N}_C(g) = \{a, b\}$, $\bar{N}_C(h) = \{c, d\}$, $\bar{N}_C(i) = \{a, b, c\}$, thus, $\delta_e = 2$, $\delta_f = 0$, $\delta_g = 1$, $\delta_h = -1$, $\delta_i = -3$. According to the definitions, $A = \{e\}$, $B = \{f\}$, $M_1 = \{e, g\}$, $M_2 = \{f\}$, $M_3 = \{h, i\}$.

Algorithm 1: Framework of the restart tabu search algorithms for MVWCP.

Input: $G = (V, E, w)$ - MVWCP instance, ρ - restart probability parameter, L - maximum number of consecutive non-improving iterations.
Output: C^* - maximum vertex weight clique.
1 **begin**
2 $C^* \leftarrow \emptyset$; /* C^* maintains the best solution found so far */
3 $C \leftarrow Random\_Solution(G)$; /* Section 3.1. C is the current solution */
4 **while** stopping condition is not met **do**
5 $(C, C^*) \leftarrow Tabu\_Search(G, C, C^*, L)$; /* Sections 3.3 and 3.4 */
6 **if** random number $\in [0, 1] < \rho$ **then**
7 $C \leftarrow Reconstruct\_Solution(G, C)$; /* Section 3.2 */
8 **else**
9 $C \leftarrow Random\_Solution(G)$; /* Section 3.1 */
10 **end**
11 **return** C^*

In addition to the *ADD* and *SWAP* operators, several restart rules of local search algorithms for MCP can also be recast with the *PUSH* operator. In particular, the restart Rule 1 in Grosso et al. (2008) (previously used in Pullan & Hoos (2006)) states that $C := [C \cap N(v)] \cup \{v\}$, v picked at random in $V \backslash C$ (i.e., add a random vertex v in the clique while keeping in the clique the adjacent vertices of v). This rule is equivalent to push a vertex from candidate push set $V \backslash C$ into the current solution. As to the restart Rule 2 in Grosso et al. (2008), let us define the candidate push set $S_q = \{v : \delta_v \leq 1 - q, v \in V \setminus C\}$ ($q > 0$ is a fixed parameter). Then the Rule 2 is to push a random vertex from S_q into C if S_q is not empty; otherwise, push a random vertex from $V \backslash C$. Moreover, in the BLS algorithm for MCP and MVWCP (Benlic & Hao, 2013), the so-called random perturbation modifies the incumbent clique by adding vertices such that the quality of the resulting clique is not deteriorated more than a quality threshold. This random perturbation strategy can simply be considered as applying the *PUSH* operator to vertices from the candidate push set $M_4 = \{v : \delta_v > (\alpha - 1) * W(C), v \in V \setminus C\}$ (where $0 < \alpha < 1$ is a predefined parameter).

Finally, by considering other candidate push sets subject to specific conditions, it is possible to obtain multiple customized search operators that can be employed by any local optimization procedure to effectively explore the search space. In the next section, we present two local search algorithms based on the above M_1, M_2, M_3 and $V \backslash C$ candidate push sets.

3. PUSH-based tabu search

In this section, we introduce two simple Restart Tabu Search (Glover & Laguna, 2013) algorithms (denoted by ReTS-I and ReTS-II). Both algorithms rely on the *PUSH* operator, but explore different candidate push sets. In ReTS-I, the candidate push set considered includes all the vertices out of the clique (i.e., CPS $= V \setminus C$) while in ReTS-II, the algorithm jointly considers the candidate push sets M_1, M_2 and M_3 introduced in Section 2.3.

Both ReTS-I and ReTS-II share the same restart local search framework as shown in Algorithm 1, but implement different local optimization procedures with different CPS (line 5, see Sections 3.3 and 3.4). The general framework starts from an initial solution C (or initial clique) generated by means of *Random_Solution* (Section 3.1). The solution is then improved by one of the dedicated tabu search procedures described respectively in Sections 3.3 and 3.4. When the search stagnates in a deep local optimum, the search restarts from a new solution, which is constructed either by *Reconstruct_Solution* (Section 3.2) with probability $\rho \in [0.0, 1.0]$ (a parameter), or by *Random_Solution* (Section 3.1) with probability $1 - \rho$. It is noted that *Reconstruct_Solution* reconstructs a new solution from C, while *Random_Solution* randomly generates a new solution from scratch. The whole search process repeats the above procedure until a prefixed stopping condition is met. The details of the tabu search optimization procedures and restart procedures are described in the following sections.

3.1. Random initial solution

The $Random\_Solution(G)$ procedure (Algorithm 1, lines 3 and 9) starts from an initial clique C composed by an unique random vertex. Then iteratively, a vertex v in candidate push set $A = \{v : |\bar{N}_C(v)| = 0, v \in V \setminus C\}$ (Sections 2.3 and 2.4) is randomly selected and added into C. A is then updated by $A \leftarrow A \setminus (\{v\} \cup \bar{N}_A(v))$. The procedure continues until the candidate push set A becomes empty. A maximal clique (i.e., $\forall v \in V \backslash C, |\bar{N}_C(v)| > 0$) is then reached and returned as the initial solution of the search procedure. This initialization procedure ignores the solution quality (the clique weight), but ensures a good randomness of the initial solutions generated. Such a feature represents a simple and useful diversification technique which helps the search algorithm to start the search in a different region of each repeated run. The initialization procedure can be efficiently implemented with a time complexity of $O(|V||E|)$. This process is similar to the initial constructive phase preceding the first *SWAP* move in the MCP algorithm of Grosso et al. (2008) and also applied in the MVWCP algorithms of Benlic and Hao (2013); Wu et al. (2012).

3.2. Solution reconstruction

The reconstruction procedure (Algorithm 1, line 7) generates a new solution by iteratively replacing vertices of a given solution. At the beginning, considering a clique C, all the vertices of $V \backslash C$ are marked available to join C by means of the *PUSH* operator. Then, at each iteration, the available vertex belonging to candidate push set M_1 (see Section 2.4) with the maximum δ value (ties are broken randomly) is selected and pushed into C. Vertices which are removed from C during the *PUSH* operation are then marked unavailable. As a consequence, they cannot rejoin the solution during the remaining iterations. The reconstruction procedure stops after $|C|$ iterations or when no available vertex may be found from M_1. The current clique C is then returned as the reconstructed solution. Such a reconstruction procedure perturbs the given solution C but, in most cases, does not decrease the quality heavily. The time complexity of each iteration is bounded by $O(|V| + (\max_{v \in V}\{|\bar{N}(v)|\})^2)$ as it scans the M_1 set and calls the *PUSH* operator (The time complexity of the push operator is discussed in Section 3.5). This reconstruction procedure can also be viewed as an objective-guided

Algorithm 2: ReTS-I: Tabu search with the largest candidate push set.

Input: C - current solution, C^* - best solution ever found, L - maximum number of consecutive non-improving iterations.
Output: C - renewed current solution, C^* - maximum vertex weight clique.

```
1  begin
2      Iter ← 0 ;                                              /* Counter of iterations */
3      for each v ∈ V do
4          tabu_v ← 0 ;            /* tabu_v is the earliest iteration vertex v is allowed to join C */
5      l ← 0 ;                     /* Counter of consecutive iterations where C* is not improved */
6      while l < L do
7          M ← {v ∈ V\C, tabu_v ≤ Iter or W(C) + δ_v > W(C*)} ;   /* M is the set of eligible vertices for PUSH */
8          v ← argmax_{v∈M} δ_v;
9          for u ∈ C \ N_C(v) do
10             tabu_u ← Iter + tt(v)
11         C ← C ⊕ PUSH(v);
12         if W(C) > W(C*) then
13             C* ← C;
14             l ← 0;
15         else
16             l ← l + 1;
17         Iter ← Iter + 1;
18     end
19 return C, C*
```

strong perturbation procedure since the vertices in the original solution are totally replaced and vertex insertions are subject to the stipulation of the maximum δ value.

3.3. ReTS-I: Tabu search with the largest candidate push set

The first tabu search procedure denoted by ReTS-I uses a greedy rule which gives preference, at each step of the search, to neighbor solutions having the best objective value. ReTS-I implements this heuristic with the largest possible candidate push set $V\backslash C$. To prevent the search from falling into cycles, a tabu mechanism (Glover & Laguna, 2013) is incorporated.

The general process of ReTS-I is shown in Algorithm 2, where each element $tabu_v$ of vector *tabu* (called the tabu list) records the earliest iteration number that vertex v is allowed to move inside C. At each iteration, one vertex is allowed to join the current solution only when it is not forbidden by the tabu list. Nevertheless, a move leading to a solution better than the best solution found so far is always accepted (this is the so-called aspiration criterion, line 7, Algorithm 2). If v is the vertex to be pushed into C, then all the vertices moving out of C (i.e., those of $\bar{N}_C(v)$) are forbidden to rejoin the solution for the next $tt(v)$ (tabu tenure) iterations (lines 9–10, Algorithm 2). The TS procedure ends when the best solution cannot be improved for L (a parameter) consecutive iterations. Note that similar strategies which temporarily forbid the removed vertices to rejoin the solution have been used in Benlic and Hao (2013); Grosso et al. (2008); Pullan (2008); Wu et al. (2012). Also note that the added vertex is free to leave the clique. This rule is based on the fact that due to the objective of maximizing the clique weight, an added vertex has little chance to be removed anyway.

The tabu tenure $tt(v)$ for a vertex v is empirically fixed as follows:

$$tt(v) = 7 + random(0, \eta(v)) \tag{3}$$

where $\eta(v) = |\{u \in V \setminus C : \bar{N}_C(u) = \bar{N}_C(v)\}|$ is the number of vertices which have as many non-adjacent vertices in C as v, and *random*(0, n) returns a random integer in range [0, n).

Since ReTS-I needs to scan $V\backslash C$ at each iteration (line 6, Algorithm 2), the time complexity of each iteration of ReST-I is bounded by $O(|V|)$. The ReTS-I algorithm is quite simple, but performs well as shown by the experimental outcomes presented in Section 4.

3.4. ReTS-II: Tabu search with three decomposed candidate push sets

Contrary to ReTS-I which explores the whole and unique candidate push set $V\backslash C$, ReTS-II, as shown in Algorithm 3 , considers more features of the candidate vertices for *PUSH*. For this algorithm, we decompose $V\backslash C$ into three candidate push sets M_1, M_2 and M_3 as defined in Section 2.4. At each iteration, the three CPS are evaluated in a fixed order: $M_1 \rightarrow M_2 \rightarrow M_3$ (lines 7–15) and a vertex with the largest δ value is chosen by *PUSH* to perform the move. Note that M_1 contains preferable vertices as they necessarily increase the weight of the incumbent clique. If no candidate vertex is available in M_1, selecting a vertex from M_2 or M_3 will degrade the solution (or keep the solution unchanged). Pushing these vertices may be useful to help the search to leave the current local optimum. M_2 is evaluated before M_3 since pushing a vertex from M_2 will generally lead to less vertices to be removed from C than pushing a vertex from M_3. The motivation of using these three sets with a preference order is thus to keep the improvement possibilities as much as possible and proceed to more important perturbations only when no other alternative is possible.

Moreover, when M_3 is used, only a random sample (of a predetermined size r) of vertices in M_3 are evaluated for each *PUSH* operation if no appropriate vertex is found in M_1 and M_2. Also, let us precise that *PUSH* selects the vertex with the best δ value in the sample set. This sampling strategy and its variants were previously used in several studies (ID-Walk Neveu, Trombettoni, and Glover, 2004, Candidate List Glover and Laguna, 2013, Best from Multiple Choices Cai, 2015). This strategy is obviously more cost effective than evaluating an entire candidate set.

ReTS-II uses the same tabu mechanism as ReTS-I. Note that the aspiration criterion does not need to be considered for pushing a vertex from M_2 and M_3 as better solutions cannot be reached in

Algorithm 3: ReTS-II: Tabu search with three candidate push sets.

Input: C - current solution, C^* - best solution ever found, L - maximum number of consecutive non-improving iterations, r - maximum sample size of M_3.
Output: C - renewed current solution, C^* - maximum vertex weight clique found.

1 **begin**
2   $Iter \leftarrow 0$;
3   **for each** $v \in V$ **do**
4     $tabu_v \leftarrow 0$;
5   $l \leftarrow 0$;
6   **while** $l < L$ **do**
7     $M \leftarrow \{v \in V \backslash C, W(C) + \delta_v > W(C^*)\}$;
8   **if** $M \neq \emptyset$ **then**
9       $l \leftarrow 0$;   /* New best solution */
10   **else**
11       $M \leftarrow \{v \in V \backslash C,\ \delta_v > 0$ and $tabu_v \leq Iter\}$;   /* Restricted M_1 */
12   **if** $M = \emptyset$ **then**
13         $M \leftarrow \{v \in V \backslash C,\ \delta_v \leq 0$ and $|\bar{N}_C(v)| = 1$ and $tabu_v \leq Iter\}$;   /* Restricted M_2 */
14   **if** $M = \emptyset$ **then**
15         $M \leftarrow$ Randomly sample r vertices from $\{v \in V \backslash C,\ tabu_v \leq Iter\}$;   /* Restricted M_3 */
16       $l \leftarrow l + 1$;
17   Randomly select $v \in \text{argmax}_{v \in M} \delta_v$;
18   **for each** $u \in C \setminus N_C(v)$ **do**
19       $tabu_u \leftarrow tt(u)$
20     $C \leftarrow C \oplus PUSH(v)$;
21   **if** $l = 0$ **then**
22       $C^* \leftarrow C$;
23     $Iter \leftarrow Iter + 1$;
24   **end**
25 **return** C, C^*

these cases. Vertices dropped from C by applying *PUSH* to M_2 or M_3 are forbidden to rejoin C for consecutive $tt(v)$ iterations (lines 18–19, Algorithm 3). The tabu tenure $tt(v)$ is tuned in the same way as in ReTS-I (Section 3.3).

Finally, it is interesting to contrast ReTS-I and ReTS-II. In fact, like ReTS-I, ReTS-II also gives priority to candidate vertices leading to a solution of better quality. Meanwhile, when no such kind of vertex exists, ReTS-II may choose a different vertex for the *PUSH* operation. For example, suppose that the same candidate push set $M = \{a, b\}$ is applied in Algorithms 2 and 3 with $\delta_a = -1,\ \delta_b = -3,\ |\bar{N}_C(a)| = 2,\ |\bar{N}_C(b)| = 1$. Then ReTS-I chooses vertex a while ReTS-II selects vertex b for the *PUSH* operation. Therefore, by using different candidate push sets, ReTS-I and ReTS-II visit different search trajectories to explore the search space. The computational experiments shown in Section 4 will allow us to observe the relative performances of both algorithms.

3.5. Fast evaluation of move gains

As presented in Section 2.3, each neighbor solution relative to a current clique leads to a move gain δ, which can be positive, null or negative. Since move gain evaluations are frequent in the TS procedures, we elaborate a fast streamlining technique which enables a direct access to all possible δ values (i.e., corresponding to the insertion in the current clique of each candidate vertex), as well as a fast update of the impacted move gains at each iteration. In this section, we present this incremental evaluation mechanism.

Let us consider a vector $\Delta = (\delta_v)_{v \in V}$ such that δ_v represents the move gain $W(C \oplus Push(v)) - W(C)$. According to the definition in Section 2.3, a *PUSH* operation is composed of two basic operations: adding a vertex to the current clique C (*ADD*), and possibly removing one or several vertices from C (*DROP*). Pushing a vertex v into a clique C can be viewed as adding v in C before removing from C every vertex which is not adjacent to v. Nevertheless this decomposition implies to consider infeasible solutions between vertex insertion and removals. We propose then to update incrementally Δ after each basic operation (*ADD*, *DROP*) by first removing from C the vertices which are not adjacent to the pushed vertex v, and finally adding v to C.

If a vertex v' is removed (dropped) from the current solution C, then the move gain δ_u of any vertex $u \in V$ is updated as follows:

$$\forall u \in V,\ \delta_u \leftarrow \begin{cases} \delta_u + w_{v'}, & \text{if } \ u \in \bar{N}(v') \\ w_u, & \text{if } \ u = v' \\ \delta_u, & \text{otherwise} \end{cases} \tag{4}$$

To speed up the update process, we use the complementary graph $\bar{G}$ of the input graph G, so that $\bar{N}(v)$ sets can be explicitly defined. Since only the move gains associated to vertices of $\bar{N}(v') \cup \{v'\}$ need to be updated, the time complexity of updating Δ after a *DROP* operation is bounded by $O(\max_{v \in V}\{|\bar{N}(v)|\})$.

Similarly, when a vertex v is added into the current solution C, then Δ is updated as follows:

$$\delta_u \leftarrow \begin{cases} \delta_u - w_v, & \text{if } \ u \in \bar{N}(v) \\ 0, & \text{if } \ u = v \\ \delta_u, & \text{otherwise} \end{cases} \tag{5}$$

This operation is obviously also bounded in time by $O(\max_{v \in V}\{|\bar{N}(v)|\})$. Thus updating the move gains after a *PUSH* operation can be performed in $O((\max_{v \in V}\{|\bar{N}(v)|\})^2)$, since operation $C \oplus Push(v)$ involves one *ADD* operation and $|\bar{N}_C(v)|$ *DROP* operations, with $|\bar{N}_C(v)|$ being bounded by $\max_{v \in V}\{|\bar{N}(v)|\}$.

4. Computational experiments

This section is dedicated to an experimental assessment of the two tabu search algorithms using the generalized *PUSH* operator. The assessment was based on three sets of 142 well-known benchmark instances and comparisons with state-of-the-art MVWCP algorithms.

4.1. Benchmarks

The three benchmark sets include the following instances: 80 DIMACS instances, 40 BHOSLIB instances, and 22 instances from the Winner Determination Problem in combinatorial auctions.

- **DIMACS benchmarks**. This set of 80 instances originated from the second DIMACS implementation challenge for the maximum clique problem[1]. These instances cover both real world problems (coding theory, fault diagnosis, Steiner Triple Problem...) and random graphs. They include small graphs (50 vertices and 1000 edges) to large graphs (4000 vertices and 5,000,000 edges). Though DIMACS graphs were originally collected for benchmarking MCP algorithms, these graphs are still very popular and widely used as a testbed for evaluating MVWCP algorithms (Benlic & Hao, 2013; Fang et al., 2016; Mannino & Stefanutti, 1999; Pullan, 2008; Wang et al., 2016; Wu et al., 2012). Considering that vertices are unweighted in these instances, we assign to each vertex i (i is an index number) the weight $i \mod 200 + 1$, following the rule in Pullan (2008).
- **BHOSLIB benchmarks**. The BHOSLIB (Benchmarks with Hidden Optimum Solutions) instances were generated randomly in the SAT phase transition area according to the model RB[2]. The 40 instances included in this set were widely used to test MCP and MVWCP algorithms. The sizes of these instances range from 450 vertices and 17,794 edges, to 1534 vertices and 127,011 edges. The weight of each vertex is assigned following the aforementioned rule, i.e., $i \mod 200 + 1$ for vertex i.
- **Winner Determination Problem (WDP) benchmarks**. The Winner Determination Problem can be reformulated as a MVWCP and thus solved by MVWCP solvers (Wu & Hao, 2015b; 2016). Therefore, benchmark instances for WDP can also be used to test the performance of MVWCP solvers. Three sets of a total number of 530 instances were reported in Wu and Hao (2015b); 2016), from which we selected 22 representative instances[3]. Six *in* instances, whose number of vertices varies among 1000, 1500 and 2000, come from Lau and Goh (2002); Ten *Decay*, *Random*, *Uniform* and *Wrandom* instances are obtained from a generator (Sandholm, 2002); *Paths* and *Regions* instances are generated using the Combinatorial Auction Test Suite (CATS, Leyton-Brown, Pearson, & Shoham, 2000). Contrary to the DIMACS and BHOSLIB instances which are defined using integer weights, WDP weights are fractional.

4.2. Experimental protocol

As shown in Algorithms 1–3, ReTS-I and ReTS-II share two common parameters: the probability parameter ρ which controls the two types of restart, and the maximum number L of consecutive non-improving iterations before a restart. Besides, ReTS-II has one additional parameter which is the sample size r. For our experiments, we used the following default values: $L = 4000$, $r = 50$, and $\rho = 0.7$. We provide an analysis of ρ in Section 5. In general, we observed that varying the parameter values around the default values did not alter much the computational outcomes for most of the tested instances even if some results can be further improved by fine-tuning the parameters.

ReTS-I and ReTS-II were coded in C++[4] and compiled with g++ 4.4.7 with optimization flag -o3. Our experiments were performed on a computer with an AMD Opteron 4184 processor (2.8GHz and 2GB RAM) running Linux 2.6.32. When solving the DIMACS machine benchmarks[5] without compilation optimization flag, the run time on our machine is 0.40, 2.50 and 9.55 seconds respectively for instances r300.5, r400.5 and r500.5.

Following the literature (Benlic & Hao, 2013; Wang et al., 2016; Wu et al., 2012), both algorithms were run 100 times to solve each benchmark instance. For the DIMACS and BHOSLIB instances, a maximum of 10^8 iterations were allowed per run while for the WDP instances, the stopping condition was set to be a cutoff time limit of 10 minutes per run. As discussed in Section 4.4, these settings correspond to the computational effort used by the state-of-the-art MVWCP algorithms in the literature.

4.3. Computational results

Tables 1 –3 report the computational results obtained by ReTS-I and ReTS-II on the DIMACS, BHOSLIB and WDP instances respectively. In these tables, column BKV reports the best-known objective values (BKV) ever found by the previous algorithms (Benlic & Hao, 2013; Fang et al., 2016; Pullan, 2008; Wang et al., 2016; Wu et al., 2012) (proven optima are indicated with the star symbol '$*$' with the BKV values). For each algorithm, column *Best*(*hit*) indicates the best objective value W^* found by ReTS-I and ReTS-II among 100 trials as well as the number of trails hitting the best value (success rate); column $ave(std)$ denotes the average value and the standard deviation of the 100 W^* values; column *Time* gives the average seconds of the trails hitting the W^* value. The value of 0.00 in columns *Time* indicates that the corresponding average time in seconds is inferior to 0.005.

Table 1 discloses that ReTS-I and ReTS-II reach all the best-known results of DIMACS instances except MANN_a45 and MANN_a81 (indicated in italic), which are believed to be quite challenging for heuristic algorithms (Fang et al., 2016). Moreover, for 73 out of 80 instances (>91 percent), both algorithms attain the best-known results in every single trial. For the remaining 7 instances except MANN_a45 and MANN_a81, each algorithm still hits the best-known results in more than 30 trials. In terms of computational time, most of these instances are solved in less than 1 second. For the 3 hard MANN instances (MANN_a27, MANN_a45, MANN_a81), results are attained in 1–17 minutes.

From Table 2 on the BHOSLIB instances, one finds that ReTS-I and ReTS-II improve the best-known result of the literature on frb53-24-3 (from 5640 to 5655). Although both algorithms attain the best-known solutions on 38 out of 40 instances, each algorithm fails to do so in 2 cases (frb50-23-4 and frb56-25-5 for ReTS-I, frb56-23-3 and frb 56-23-4 for ReTS-II, indicated in italic). Interestingly, both algorithms achieve a success rate of at least 93 percent on the first 20 instances while the success rate drops to less than 50 percent on most of the last 20 instances. Concerning the computational time, both algorithms require more time to find the best-known solutions when the sizes of the graphs increase, but each average time is inferior to 12 minutes. In general, BHOSLIB instances are more difficult than DIMACS ones for ReTS-I and ReTS-II, but both algorithms still perform quite well by

[1] http://www.cs.hbg.psu.edu/txn131/clique.html
[2] http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm
[3] www.info.univ-angers.fr/pub/hao/wdp.html
[4] Our source code will be made available at: www.info.univ-angers.fr/pub/hao/ReTS.html.
[5] dfmax: ftp://dimacs.rutgers.edu/pub/dsj/clique/.

Table 1
Computational results of ReTS-I and ReTS-II on 80 DIMACS instances.

Instance	BKV	ReTS-I			ReTS-II		
		Best(*hit*)	*ave*(*std*)	*Time*	*Best*(*hit*)	*ave*(*std*)	*Time*
C1000.9	9254	9254(100)	9254.00(0.00)	2.50	9254(100)	9254.00(0.00)	1.73
C125.9	2529	2529(100)	2529.00(0.00)	0.00	2529(100)	2529.00(0.00)	0.00
C2000.5	2466	2466(100)	2466.00(0.00)	2.34	2466(100)	2466.00(0.00)	7.39
C2000.9	10,999	10,999(92)	10,996.44(8.72)	417.56	10,999(82)	10,993.08(12.76)	474.23
C250.9	5092*	5092(100)	5092.00(0.00)	0.01	5092(100)	5092.00(0.00)	0.01
C4000.5	2792	2792(100)	2792.00(0.00)	116.05	2792(100)	2792.00(0.00)	298.05
C500.9	6955	6955(100)	6955.00(0.00)	0.06	6955(100)	6955.00(0.00)	0.08
DSJC1000_5	2186*	2186(100)	2186.00(0.00)	0.38	2186(100)	2186.00(0.00)	0.37
DSJC500_5	1725*	1725(100)	1725.00(0.00)	0.13	1725(100)	1725.00(0.00)	0.10
MANN_a27	12,283*	12,283(78)	12,282.78(0.41)	82.77	12,283(99)	12,282.99(0.10)	60.03
MANN_a45	34,265*	*34,259*(1)	34,253.60(1.11)	157.98	*34,254*(58)	34,253.43(0.74)	357.19
MANN_a81	111,386	*111,370*(1)	111,351.19(6.63)	990.02	*111,277*(1)	111,233.47(26.42)	477.75
MANN_a9	372	372(100)	372.00(0.00)	0.00	372(100)	372.00(0.00)	0.00
brock200_1	2821	2821(100)	2821.00(0.00)	0.00	2821(100)	2821.00(0.00)	0.00
brock200_2	1428	1428(100)	1428.00(0.00)	0.00	1428(100)	1428.00(0.00)	0.00
brock200_3	2062	2062(100)	2062.00(0.00)	0.00	2062(100)	2062.00(0.00)	0.00
brock200_4	2107	2107(100)	2107.00(0.00)	0.00	2107(100)	2107.00(0.00)	0.00
brock400_1	3422*	3422(100)	3422.00(0.00)	0.04	3422(100)	3422.00(0.00)	0.05
brock400_2	3350*	3350(100)	3350.00(0.00)	0.04	3350(100)	3350.00(0.00)	0.07
brock400_3	3471*	3471(100)	3471.00(0.00)	0.07	3471(100)	3471.00(0.00)	0.06
brock400_4	3626*	3626(100)	3626.00(0.00)	2.04	3626(100)	3626.00(0.00)	1.43
brock800_1	3121*	3121(100)	3121.00(0.00)	0.14	3121(100)	3121.00(0.00)	0.20
brock800_2	3043*	3043(100)	3043.00(0.00)	0.39	3043(100)	3043.00(0.00)	0.61
brock800_3	3076*	3076(100)	3076.00(0.00)	0.39	3076(100)	3076.00(0.00)	0.51
brock800_4	2971*	2971(31)	2970.31(0.46)	835.03	2971(93)	2970.93(0.26)	506.41
c-fat200-1	1284	1284(100)	1284.00(0.00)	0.00	1284(100)	1284.00(0.00)	0.00
c-fat200-2	2411	2411(100)	2411.00(0.00)	0.00	2411(100)	2411.00(0.00)	0.00
c-fat200-5	5887	5887(100)	5887.00(0.00)	0.00	5887(100)	5887.00(0.00)	0.00
c-fat500-1	1354	1354(100)	1354.00(0.00)	0.01	1354(100)	1354.00(0.00)	0.01
c-fat500-10	11,586	11,586(100)	11,586.00(0.00)	0.11	11,586(100)	11,586.00(0.00)	0.03
c-fat500-2	2628	2628(100)	2628.00(0.00)	0.02	2628(100)	2628.00(0.00)	0.01
c-fat500-5	5841	5841(100)	5841.00(0.00)	0.09	5841(100)	5841.00(0.00)	0.03
gen200_p0.9_44	5043*	5043(100)	5043.00(0.00)	0.00	5043(100)	5043.00(0.00)	0.00
gen200_p0.9_55	5416*	5416(100)	5416.00(0.00)	0.12	5416(100)	5416.00(0.00)	0.00
gen400_p0.9_55	6718	6718(100)	6718.00(0.00)	0.18	6718(100)	6718.00(0.00)	0.12
gen400_p0.9_65	6940	6940(100)	6940.00(0.00)	0.05	6940(100)	6940.00(0.00)	0.04
gen400_p0.9_75	8006*	8006(100)	8006.00(0.00)	0.03	8006(100)	8006.00(0.00)	0.02
hamming10-2	50,512*	50,512(100)	50,512.00(0.00)	0.20	50,512(100)	50,512.00(0.00)	0.20
hamming10-4	5129	5129(100)	5129.00(0.00)	26.25	5129(100)	5129.00(0.00)	15.74
hamming6-2	1072	1072(100)	1072.00(0.00)	0.00	1072(100)	1072.00(0.00)	0.00
hamming6-4	134	134(100)	134.00(0.00)	0.00	134(100)	134.00(0.00)	0.00
hamming8-2	10,976	10,976(100)	10,976.00(0.00)	0.01	10,976(100)	10,976.00(0.00)	0.01
hamming8-4	1472	1472(100)	1472.00(0.00)	0.00	1472(100)	1472.00(0.00)	0.00
johnson16-2-4	548	548(100)	548.00(0.00)	0.00	548(100)	548.00(0.00)	0.00
johnson32-2-4	2033*	2033(100)	2033.00(0.00)	0.04	2033(100)	2033.00(0.00)	0.04
johnson8-2-4	66	66(100)	66.00(0.00)	0.00	66(100)	66.00(0.00)	0.00
johnson8-4-4	511	511(100)	511.00(0.00)	0.00	511(100)	511.00(0.00)	0.00
keller4	1153	1153(100)	1153.00(0.00T	0.00	1153(100)	1153.00(0.00)	0.00
keller5	3317	3317(100)	3317.00(0.00)	1.12	3317(100)	3317.00(0.00)	0.33
keller6	8062	8062(100)	8062.00(0.00)	532.74	8062(96)	8059.91(10.78)	929.74
p_hat1000-1	1514*	1514(100)	1514.00(0.00)	0.14	1514(100)	1514.00(0.00)	0.28
p_hat1000-2	5777*	5777(100)	5777.00(0.00)	0.11	5777(100)	5777.00(0.00)	0.11
p_hat1000-3	8111	8111(100)	8111.00(0.00)	0.19	8111(100)	8111.00(0.00)	0.21
p_hat1500-1	1619*	1619(100)	1619.00(0.00)	0.32	1619(100)	1619.00(0.00)	0.39
p_hat1500-2	7360	7360(100)	7360.00(0.00)	0.35	7360(100)	7360.00(0.00)	0.44
p_hat1500-3	10,321	10,321(100)	10,321.00(0.00)	2.06	10,321(100)	10,321.00(0.00)	0.50
p_hat300-1	1057	1057(100)	1057.00(0.00)	0.00	1057(100)	1057.00(0.00)	0.00
p_hat300-2	2487	2487(100)	2487.00(0.00)	0.01	2487(100)	2487.00(0.00)	0.01
p_hat300-3	3774	3774(100)	3774.00(0.00)	0.01	3774(100)	3774.00(0.00)	0.01
p_hat500-1	1231*	1231(100)	1231.00(0.00)	0.03	1231(100)	1231.00(0.00)	0.04
p_hat500-2	3920*	3920(100)	3920.00(0.00)	0.02	3920(100)	3920.00(0.00)	0.03
p_hat500-3	5375*	5375(100)	5375.00(0.00)	0.04	5375(100)	5375.00(0.00)	0.05
p_hat700-1	1441*	1441(100)	1441.00(0.00)	0.04	1441(100)	1441.00(0.00)	0.05
p_hat700-2	5290*	5290(100)	5290.00(0.00)	0.06	5290(100)	5290.00(0.00)	0.05
p_hat700-3	7565	7565(100)	7565.00(0.00)	0.10	7565(100)	7565.00(0.00)	0.08
san1000	1716*	1716(100)	1716.00(0.00)	71.07	1716(100)	1716.00(0.00)	12.08
san200_0.7_1	3370	3370(100)	3370.00(0.00)	0.21	3370(100)	3370.00(0.00)	0.13
san200_0.7_2	2422*	2422(100)	2422.00(0.00)	0.04	2422(100)	2422.00(0.00)	0.01
san200_0.9_1	6825*	6825(100)	6825.00(0.00)	0.04	6825(100)	6825.00(0.00)	0.01
san200_0.9_2	6082*	6082(100)	6082.00(0.00)	0.00	6082(100)	6082.00(0.00)	0.00
san200_0.9_3	4748*	4748(100)	4748.00(0.00)	0.01	4748(100)	4748.00(0.00)	0.01
san400_0.5_1	1455	1455(100)	1455.00(0.00)	0.19	1455(100)	1455.00(0.00)	0.11

(*continued on next page*)

Table 1 (*continued*)

Instance	BKV	ReTS-I			ReTS-II		
		Best(*hit*)	*ave*(*std*)	*Time*	*Best*(*hit*)	*ave*(*std*)	*Time*
san400_0.7_1	3941*	3941(97)	3932.00(51.18)	172.04	3941(100)	3941.00(0.00)	74.66
san400_0.7_2	3110*	3110(97)	3105.26(26.95)	234.69	3110(100)	3110.00(0.00)	40.11
san400_0.7_3	2771*	2771(100)	2771.00(0.00)	0.41	2771(100)	2771.00(0.00)	0.07
san400_0.9_1	9776*	9776(100)	9776.00(0.00)	2.38	9776(100)	9776.00(0.00)	0.44
sanr200_0.7	2325*	2325(100)	2325.00(0.00)	0.00	2325(100)	2325.00(0.00)	0.00
sanr200_0.9	5126*	5126(100)	5126.00(0.00)	0.00	5126(100)	5126.00(0.00)	0.00
sanr400_0.5	1835*	1835(100)	1835.00(0.00)	0.02	1835(100)	1835.00(0.00)	0.02
sanr400_0.7	2992*	2992(100)	2992.00(0.00)	0.03	2990(100)	2990.00(0.00)	1.54

Table 2
Computational results of ReTS-I and ReTS-II on 40 BHOSLIB instances.

Instance	BKV	ReTS-I			ReTS-II		
		Best(*hit*)	*ave*(*std*)	*Time*	*Best*(*hit*)	*ave*(*std*)	*Time*
frb30-15-1	2990*	2990(100)	2990.00(0.00)	1.43	2990(100)	2990.00(0.00)	1.54
frb30-15-2	3006*	3006(100)	3006.00(0.00)	2.09	3006(100)	3006.00(0.00)	0.28
frb30-15-3	2995*	2995(100)	2995.00(0.00)	1.84	2995(100)	2995.00(0.00)	1.31
frb30-15-4	3032*	3032(100)	3032.00(0.00)	0.31	3032(100)	3032.00(0.00)	0.17
frb30-15-5	3011*	3011(100)	3011.00(0.00)	0.80	3011(100)	3011.00(0.00)	1.16
frb35-17-1	3650	3650(100)	3650.00(0.00)	5.10	3650(100)	3650.00(0.00)	3.19
frb35-17-2	3738	3738(100)	3738.00(0.00)	87.05	3738(100)	3738.00(0.00)	65.51
frb35-17-3	3716	3716(100)	3716.00(0.00)	22.26	3716(100)	3716.00(0.00)	10.17
frb35-17-4	3683	3683(100)	3683.00(0.00)	14.80	3683(100)	3683.00(0.00)	1.89
frb35-17-5	3686	3686(100)	3686.00(0.00)	2.70	3686(100)	3686.00(0.00)	6.40
frb40-19-1	4063	4063(100)	4063.00(0.00)	51.68	4063(100)	4063.00(0.00)	61.32
frb40-19-2	4112	4112(100)	4112.00(0.00)	71.72	4112(100)	4112.00(0.00)	73.87
frb40-19-3	4115	4115(99)	4114.94(0.60)	127.00	4115(100)	4115.00(0.00)	79.66
frb40-19-4	4136	4136(98)	4135.92(0.56)	160.48	4136(100)	4136.00(0.00)	44.45
frb40-19-5	4118	4118(100)	4118.00(0.00)	34.72	4118(98)	4117.96(0.28)	208.18
frb45-21-1	4760	4760(98)	4759.76(1.68)	161.39	4760(93)	4759.10(3.29)	231.63
frb45-21-2	4784	4784(100)	4784.00(0.00)	68.11	4784(100)	4784.00(0.00)	35.50
frb45-21-3	4765	4765(90)	4764.80(0.60)	253.27	4765(100)	4765.00(0.00)	53.80
frb45-21-4	4799	4799(100)	4799.00(0.00)	105.52	4799(100)	4799.00(0.00)	31.81
frb45-21-5	4779	4779(100)	4779.00(0.00)	11.23	4779(100)	4779.00(0.00)	10.56
frb50-23-1	5494	5494(4)	5485.18(4.01)	154.05	5494(4)	5482.58(5.64)	590.72
frb50-23-2	5462	5462(9)	5451.94(3.20)	393.53	5462(44)	5455.27(6.08)	458.97
frb50-23-3	5486	5486(57)	5485.24(1.59)	358.92	5486(87)	5485.82(0.50)	292.35
frb50-23-4	5454	*5453*(91)	5452.54(1.48)	243.84	5454(6)	5450.70(4.36)	548.32
frb50-23-5	5498	5498(100)	5498.00(0.00)	118.21	5498(94)	5497.31(2.80)	277.51
frb53-24-1	5670	5670(33)	5661.37(8.67)	349.95	5670(58)	5664.29(8.05)	325.66
frb53-24-2	5707	5707(1)	5685.28(8.73)	880.86	5707(5)	5689.22(10.82)	415.40
frb53-24-3	5640	**5655**(3)	5636.51(6.54)	417.69	**5655**(3)	5632.08(8.50)	457.64
frb53-24-4	5714	5714(4)	5696.85(17.10)	421.52	5714(7)	5693.46(17.38)	402.50
frb53-24-5	5659	5659(1)	5651.39(2.96)	777.93	5659(5)	5649.69(6.01)	381.24
frb56-25-1	5916	5916(59)	5906.48(14.25)	344.18	5916(51)	5900.01(18.76)	428.24
frb56-25-2	5886	5886(9)	5873.03(8.70)	516.06	5886(37)	5878.44(8.03)	470.10
frb56-25-3	5859	5859(1)	5832.31(13.27)	450.99	*5854*(1)	5821.85(14.57)	30.19
frb56-25-4	5892	5892(2)	5866.11(13.48)	477.79	*5885*(2)	5856.77(13.99)	449.13
frb56-25-5	5853	*5841*(1)	5812.23(9.32)	354.28	5853(2)	5816.87(13.86)	514.91
frb59-26-1	6591	6591(20)	6578.65(7.24)	521.08	6591(32)	6576.59(11.57)	432.31
frb59-26-2	6645	6645(13)	6589.14(25.69)	505.28	6645(25)	6599.84(30.81)	660.05
frb59-26-3	6608	6608(1)	6579.05(13.05)	973.94	6608(25)	6593.63(12.32)	455.84
frb59-26-4	6592	6592(71)	6585.08(12.12)	377.92	6592(18)	6565.23(20.67)	541.34
frb59-26-5	6584	6584(3)	6558.48(10.04)	320.54	6584(9)	6563.39(10.95)	399.46

attaining together all the best-known results and finding even an improved best-known result (new best lower bound).

Table 3 (WDP instances) shows that ReTS-I and ReTS-II attain the best-known solutions on all WDP instances considered except the 4 *Decay* instances and Paths2000_100 (in italic); for Paths2000_100, the algorithms have a success rate of 88 percent and 76 percent respectively, while this rate drops to 10 percent or less for the 4 Decay instances, confirming that these Decay instances are particularly hard (Sandholm, 2002). Actually, as shown in Section 4.4, these instances also represent a challenge for one of the best performing reference heuristics MN/TS (Wu et al., 2012). Finally, one observes that ReTS-II attains the best-known results every run on the *in* instances in less than 4 seconds, while ReTS-I fails to consistently hit the best results and requires longer computing times for these instances. Nevertheless we cannot claim that ReTS-II always dominates ReTS-I since the latter performs better on Uniform2000_400_10 in terms of successful trials and computing time.

4.4. Comparisons with state-of-the-art algorithms

As indicated in the introduction, a large number of heuristic algorithms for the Maximum Vertex Weight Clique Problem have been reported in the literature, including particularly AugSearch

Table 3
Computational results of ReTS-I and ReTS-II on 22 selected WDP instances.

Instance	BKV	ReTS-I			ReTS-II		
		Best(hit)	*ave(std)*	*Time*	*Best(hit)*	*ave(std)*	*Time*
in101	72,724.61	72,724.62(63)	72,243.68(731.24)	216.92	72,724.62(100)	72,724.62(0.00)	0.31
in108	75,813.21	75,813.21(100)	75,813.21(0.00)	5.02	75,813.21(100)	75,813.21(0.00)	1.87
in115	70,221.56	70,221.56(76)	70,149.21(128.75)	234.30	70,221.56(100)	70,221.56(0.00)	1.76
in201	81,557.74	81,557.74(100)	81,557.74(0.00)	2.35	81,557.74(100)	81,557.74(0.00)	0.41
in207	93,129.25	93,129.25(100)	93,129.25(0.00)	11.57	93,129.25(100)	93,129.25(0.00)	0.65
in209	87,268.96	87,268.96(11)	86,812.25(160.57)	254.65	87,268.96(100)	87,268.96(0.00)	0.98
in401	77,417.48∗	77,417.48(100)	77,417.48(0.00)	0.69	77,417.48(100)	77,417.48(0.00)	0.04
in403	74,843.96∗	74,843.96(100)	74,843.96(0.00)	0.05	74,843.96(100)	74,843.96(0.00)	0.04
in404	78,761.69∗	78,761.69(100)	78,761.69(0.00)	0.48	78,761.69(100)	78,761.69(0.00)	0.14
in601	108,800.45	108,800.45(100)	108,800.45(0.00)	76.26	108,800.45(100)	108,800.45(0.00)	3.34
in604	107,733.80	107,733.80(40)	107,062.40(548.20)	257.76	107,733.80(100)	107,733.80(0.00)	2.90
in614	108,364.58	108,364.58(100)	108,364.58(0.00)	24.66	108,364.58(100)	108,364.58(0.00)	0.84
Decay2000_200	159.67∗	*156.97*(10)	154.63(0.91)	496.68	*157.34*(4)	155.99(0.52)	215.57
Decay2000_300	226.82∗	*221.17*(1)	218.46(1.24)	116.72	*221.92*(9)	219.74(0.77)	201.19
Decay2000_400	277.01∗	*270.90*(5)	267.79(1.36)	127.14	*271.16*(5)	269.85(0.65)	495.26
Decay2000_500	340.81∗	*334.36*(1)	330.12(1.45)	114.02	*333.80*(5)	330.77(1.09)	552.07
Random2000_500	12.63∗	12.63(100)	12.63(0.00)	0.19	12.63(100)	12.63(0.00)	0.18
Uniform2000_400_10	22.02	22.02(100)	22.02(0.00)	83.76	22.02(90)	22.00(0.06)	197.04
Uniform2000_500_10	26.56	26.56(100)	26.55(0.03)	244.54	26.56(100)	26.50(0.07)	294.15
Wrandom2000_500	37.69∗	37.69(100)	37.69(0.00)	0.20	37.69(100)	37.69(0.00)	0.19
Paths2000_100	36.77∗	*35.56*(88)	35.13(0.13)	45.90	*36.32*(72)	36.05(0.09)	54.53
Regions2000_40	4558.90∗	4558.90(19)	4503.94(35.25)	223.79	4558.90(54)	4540.09(22.69)	220.75

(Mannino & Stefanutti, 1999), HSSGA (Singh & Gupta, 2006), PLS (Pullan, 2008), MN/TS (Wu et al., 2012), BLS (Benlic & Hao, 2013), and BQP-PTS (Wang et al., 2016). To further assess the performance of the proposed approach, we compared ReTS-I and ReTS-II with three state-of-the-art algorithms (MN/TS, BLS and BQP-TS). Besides, these reference algorithms have been run on computing platforms which are the same as or very similar to our computer (2.8 GigaHertz and 2 GigaBytes RAM running Linux 2.6.32). For the WDP instances, we used CPLEX as an additional reference as it achieves more competitive results than heuristic algorithms on some specific instances (Wu & Hao, 2015b). Since CPLEX did not perform well on DIMACS and BHOSLIB instances (Fang et al., 2016), it was not used for our comparisons for these two benchmarks.

- **MN/TS** is a tabu search algorithm with multiple move operators, designed for solving both MCP and MVWCP (Wu et al., 2012). The results reported for MN/TS on the DIMACS and BHOSLIB instances have been obtained using a maximum of 10^8 iterations per run (on a computer cadenced at 2.83 GigaHertz and 8 GigaBytes RAM). Besides, the results of MN/TS on the WDP instances within 300 seconds per run were reported in Wu and Hao (2015b). Each instance was solved for 100 independent trials in all these experiments. Thus, both the stopping condition and the computing platform are almost the same as those used in our experiments.
- **BLS** (Breakout Local Search) incorporates an adaptive perturbation strategy for the resolution of MCP and MVWCP (Benlic & Hao, 2013). BLS reported computational results on the sets of DIMACS and BHOSLIB benchmarks, by running the algorithm 100 times on each instance on the same computing platform as our algorithms (2.83 GigaHertz Xeon E5440 CPU and 2 GigaBytes RAM). The stopping condition for each of the 100 runs was set to 1.6×10^8 iterations, which was superior to the computational limit used by MN/TS and our algorithms.
- **BQP-PTS** is a probabilistic tabu search algorithm designed for solving unconstrained Binary Quadratic Programs (BQP) (Wang et al., 2016). To solve the MVWCP instances, each instance is first recast into a BQP which is then solved by the probabilistic tabu search algorithm. The DIMACS and BHOSLIB instances were tested by this method on a PC with a Pentium 2.83 GigaHertz CPU and 2 GigaBytes RAM. Each benchmark instance was solved by 100 independent trails, each trail being limited to 3600 seconds, but extended to 36,000 seconds for large instances C4000.5, MABNN_a27, MANN_a45, MANN_a81.
- **CPLEX**. For the WDP instances, we include the results reported in Wu and Hao (2015b), which were obtained by the exact solver, CPLEX 12.4, within a maximum of 3600 seconds on a PC cadenced at 2.83 GigaHertz with 8 GigaBytes of RAM.

Considering that MN/TS, BLS and BQP-PTS have a 100 percent success rate on most of the DIMACS instances in less than 1 seconds, we selected 7 hard and representative instances from this set in order to summarize the performances of the 5 compared algorithms. Moreover, as indicated in Section 4.3, the last 20 instances of BHOSLIB are more challenging for ReTS-I and ReTS-II than the first 20 instances, we only highlight the comparative results on these last 20 instances. The results of this comparison are summarized in Table 4. Column *Gap* represents the gap between the objective value of the best solution found by an algorithm and the best-known value in the literature (BKV). A positive (negative) gap value indicates a better (worse) result compared to the current best-known value.

Table 4 indicates that both ReTS-I and ReTS-II attain better results than the 3 reference algorithms on 4 instances (highlighted in bold font). Though the MANN_a*XX* instances were reported as challenging for heuristic algorithms, ReTS-I and ReTS-II reach the optimal solution of MANN_a27 and better solutions on MANN_a45 and MANN_a81. BLS and BQP-PTS reach the best-known solutions for all other instances while the other algorithms fail on 1 or 2 instances. However, they achieve such a performance by using a larger cutoff time, which is also confirmed by the fact that the average time of BLS and BQP-PTS is significantly longer than the 3 other algorithms. In an additional experiment, we used a maximum of 1.6×10^8 iterations per run (the same condition as that used by BLS) and re-ran ReTS-I to solve frb53-23-4 and frb56-2-5, ReTS-II to solve frb56-25-3 (setting $\rho = 0.3$ in this case) and frb56-25-4. The results, shown in Table 5, indicate that ReTS-I and ReTS-II, like BLS and BQP-PTS, are also able to hit all the BKVs with a similar computational effort. Finally, we observe that MN/TS is the most time effective heuristic among the compared algorithms.

To further compare the competing algorithms, we extract the rows from Table 4 where $gap = 0$ for all 5 algorithms (18 rows in

Table 4
Experimental results of ReTS-I and ReTS-II in comparison with 3 reference algorithms on 27 selected DIMACS and BHOSLIB instances.

		ReTS-I			ReTS-II			MN/TS			BLS			BQP-PTS		
Instances	BKV	*Gap*	*Hit*	Time	*Gap*	*Hit*	*Time*	*Gap*	*Hit*	*Time*	*Gap*	*Hit*	*Time*	*Gap*	*Hit*	Time
C2000.9	10,999	0	92	417.56	0	82	474.23	0	72	168.11	0	74	1152.78	0	72	2711.97
MANN_a27	12,283*	**0**	78	82.77	**0**	99	60.03	−2	1	88.28	−2	16	396.58	−6	4	12,264
MANN_a45	34,265*	**−6**	1	157.98	**−13**	58	357.19	−73	1	390.58	−36	1	929.41	−71	2	17,524.05
MANN_a81	111,386	**−16**	1	990.02	**−109**	1	477.75	−258	1	832.24	−149	1	2942.54	−249	1	6167.28
p_hat1000-3	8111	0	100	0.19	0	100	0.21	0	96	188.38	0	100	1.78	0	100	0.65
brock800_4	2971*	0	31	835.03	0	93	506.41	0	100	49.70	0	100	339.07	0	8	105.35
keller6	8062	0	100	532.74	0	96	929.74	0	5	606.15	0	44	1980.16	0	2	3418.36
frb50-23-1	5494	0	4	154.05	0	4	590.72	0	6	186.62	0	11	1221.72	0	20	1911.49
frb50-23-2	5462	0	9	393.53	0	44	458.97	0	3	14,966	0	5	2837.74	0	15	2338.40
frb50-23-3	5486	0	57	358.92	0	87	292.35	0	53	158.71	0	98	537.96	0	100	418.35
frb50-23-4	5454	−1	91	243.84	0	6	548.32	0	9	176.41	0	14	1190.43	0	28	1957.22
frb50-23-5	5498	0	100	118.21	0	94	277.51	0	89	110.85	0	100	388.18	0	100	751.84
frb53-24-1	5670	0	33	349.95	0	58	325.66	0	5	233.22	0	13	1056.82	0	43	981.33
frb53-24-2	5707	0	1	880.86	0	5	415.40	0	6	145.22	0	3	147.65	0	25	1265.70
frb53-24-3	5640	**15**	3	417.69	**15**	3	457.64	0	15	215.79	0	48	984.53	0	90	1486.24
frb53-24-4	5714	0	4	421.52	0	7	402.50	0	7	449.39	0	13	1604.50	0	25	1753.36
frb53-24-5	5659	0	1	777.93	0	5	381.24	0	5	294.00	0	4	278.91	0	6	2802.83
frb56-25-1	5916	0	59	344.18	0	51	428.24	0	3	308.90	0	5	1764.87	0	19	1035.00
frb56-25-2	5886	0	9	516.06	0	37	470.10	−14	1	73.25	0	1	1013.85	0	3	1428.18
frb56-25-3	5859	0	1	450.99	−5	1	30.19	0	1	191.93	0	1	101.48	0	5	1756.22
frb56-25-4	5892	0	2	477.79	−7	2	449.13	0	3	104.58	0	12	1256.9	0	5	1756.22
frb56-25-5	5853	−12	1	354.28	0	2	514.91	0	1	322.70	0	1	4386.6	0	1	3549.57
frb59-26-1	6591	0	20	521.08	0	32	432.31	0	3	166.20	0	17	1435.99	0	67	2228.21
frb59-26-2	6645	0	13	505.28	0	25	660.05	0	3	212.49	0	13	1834.93	0	40	1820.56
frb59-26-3	6608	0	1	973.94	0	25	455.84	0	1	232.77	0	1	507.93	0	1	2561.16
frb59-26-4	6592	0	71	377.92	0	18	541.34	0	1	318.39	0	6	952.34	0	5	3322.64
frb59-26-5	6584	0	3	320.54	0	9	399.46	0	1	161.47	0	5	1512.09	0	9	747.80

Table 5
Improved results of ReTS-I on frb50-23-4 and frb56-25-5 and improved results of ReTS-II on frb56-25-3 and frb56-25-4 with an extended cutoff time limit.

Solver	Instance	BKV	*Best*(*hit*)	*ave*(*std*)	*Time*
ReTS-I	frb50-23-4	5454	5454(3)	5452.99(0.44)	504.07
	frb56-25-5	5853	5853(5)	5820.14(14.23)	763.00
ReTS-II	frb56-25-3	5859	5859(2)	5831.98(14.07)	1386.24
	frb56-25-4	5885	5885(5)	5863.16(13.09)	1003.07

Table 6
Average hits on 18 selected instances.

ReTS-I	ReTS-II	MN/TS	BLS	BQP-PTS
38.8	46.4	25.5	34.0	36.5

total), and recalculate the average number of the best trials (*hits*) for these 18 rows. Results are shown in Table 6, and indicate that ReTS-II and ReTS-I are the most robust algorithms, followed by BQP-PTS, BLS and MN/TS. We conclude thus that ReTS-I and ReTS-II compete favorably compared to the reference algorithms in terms of solution quality, robustness and computational time on the DIMACS and BHOSLIB instances.

Finally, Table 7 summarizes the results of ReTS-I, ReTS-II, MN/TS and CPLEX on 10 representative WDP instances, including the most challenging ones (with respect to results taken from Table 3). If we look at the solution quality, we observe that the three heuristic algorithms (ReTS-I, ReTS-II, MN/TS) attain the best-known solutions for the tested *in* instances (in101, in108, in109) and the unique *Uniform2000_400_10* instance while CPLEX fails to solve these instances. However, CPLEX is able to find the optimum solutions of Decay2000_*yyy* and Paths2000_100, contrary to the three heuristic methods. Among the compared heuristic algorithms, ReTS-I finds the best solution on 1 instance, ReTS-II on 2 instances and MN/TS on 2 (marked by bold font). Therefore, no algorithm outperforms the other algorithms in terms of solution quality and computational time. So, on the WDP instances, ReTS-I, ReTS-II and MN/TS perform similarly. Finally, this experiment confirms that exact solvers like CPLEX and heuristics like ReTS-I, ReTS-II and MN/TS are complementary solution methods and together can enlarge the class of MVWCP instances that can be solved effectively.

Table 7
Comparison of our ReTS-I and ReTS-II algorithms with MN/TS, CPLEX on the WDP instances.

		ReTS-I			ReTS-II			MN/TS			CPLEX	
Instances	BKV	*Gap*	*Hit*	*Time*	*Gap*	*Hit*	*Time*	*Gap*	*Hit*	*Time*	*Gap*	*Time*
in101	72,724.61	0	63	216.92	0	100	0.31	0	100	5.46	*−5622.67*	3600
in108	75,813.21	0	100	5.02	0	100	1.87	0	73	113.53	*−1175.42*	3600
in209	87,268.96	0	11	254.65	0	100	0.98	0	100	11.25	*−4102.57*	3600
Decay2000_200	159.67∗	*−2.70*	10	496.68	*−2.33*	3	215.57	**−0.49**	3	220.01	0	0.39
Decay2000_300	226.82∗	*−5.65*	1	116.72	**−4.9**	9	201.19	*−6.16*	1	226.23	0	0.70
Decay2000_400	277.01∗	*−10.11*	5	127.14	**−6.74**	5	495.26	−11.14	1	256.66	0	0.72
Decay2000_500	340.81∗	**−6.45**	1	114.02	*−7.01*	5	552.07	*−24.70*	1	189.36	0	1.23
Uniform2000_400_10	22.02	0	100	83.76	0	90	197.04	0	100	79.70	*−2.84*	3600
Paths2000_100	36.77∗	*−1.21*	88	45.90	*−0.45*	72	54.53	**−0.36**	1	225.39	0	0.09
Regions2000_40	4558.90	0	19	223.79	0	54	220.75	0	100	4.63	0	0.22

Table 8
Impact of the parameter ρ on the results of ReTS-I and ReTS-II.

Algorithm	ρ	in604		Decay2000_500		C2000.9		MANN_a45		keller6		frb50-23-2		frb59-26-5	
		ave(std)	*Time*	*ave(std)*	*Time*	*ave(std)*	*Time*	*ave(std)*	*Time*	*ave(std)*	*Time*	*ave(std)*	*Time*	*ave(std)*	*Time*
ReTS-I	0	107,733.80(0.00)	3.07	330.59(2.21)	24.37	10,947.45(33.72)	54.14	34,175.80(5.82)	44.17	7818.50(105.20)	57.88	5446.20(7.37)	49.42	6539.65(15.13)	48.74
	0.1	107,733.80(0.00)	3.60	329.09(1.26)	25.08	10,951.20(37.20)	59.43	34,235.95(4.93)	55.55	7841.05(103.14)	59.34	5445.20(8.33)	53.60	6530.80(12.52)	56.83
	0.2	107,733.80(0.00)	2.45	329.32(1.36)	37.79	10,946.60(29.54)	44.86	34,241.75(3.08)	49.92	7856.15(86.90)	60.70	5446.80(6.38)	63.48	6535.80(12.34)	46.74
	0.3	107,733.80(0.00)	3.38	329.75(1.45)	38.69	10,937.20(25.06)	57.10	34,244.55(3.01)	63.80	7861.85(81.05)	66.02	5445.10(7.39)	59.91	6538.90(16.73)	48.74
	0.4	107,733.80(0.00)	3.12	329.30(1.12)	35.88	10,958.35(34.90)	66.12	34,247.15(3.95)	53.13	7809.75(69.26)	62.93	5443.75(9.09)	54.26	6539.05(15.55)	58.78
	0.5	107,733.80(0.00)	2.07	329.97(1.40)	27.37	10,935.85(29.43)	65.60	34,249.30(3.74)	40.57	7866.25(100.82)	72.52	5446.35(7.53)	61.54	6541.45(15.72)	50.78
	0.6	107,733.80(0.00)	2.75	329.69(1.05)	45.96	10,950.05(27.00)	52.82	34,250.70(2.43)	54.44	7845.90(96.06)	50.91	5448.65(3.64)	53.85	6537.50(10.57)	69.19
	0.7	107,733.80(0.00)	2.76	330.03(1.71)	38.22	10,956.50(26.82)	55.75	34,251.85(1.93)	38.78	7899.10(102.42)	53.65	5439.60(8.21)	44.17	6543.95(13.65)	41.12
	0.8	107,733.80(0.00)	3.34	329.69(1.26)	38.46	10,934.05(29.88)	42.43	34,252.65(0.85)	50.09	7883.55(88.75)	56.29	5445.30(8.59)	53.97	6544.60(13.71)	54.71
	0.9	107,733.80(0.00)	2.54	330.16(1.11)	52.61	10,943.70(36.12)	72.01	34,252.75(0.94)	35.58	7933.75(98.15)	68.88	5449.30(6.49)	61.68	6543.15(15.37)	58.47
	1	107,733.80(0.00)	3.76	329.33(0.87)	37.26	10,945.25(24.52)	54.04	34,248.10(15.94)	32.01	7935.70(103.42)	56.19	5447.90(8.39)	63.21	6543.95(13.61)	76.84
ReTS-II	0	106,814.21(471.76)	55.32	329.92(1.73)	17.48	10,963.80(8.62)	27.10	34,174.50(2.25)	56.36	7935.70(67.82)	49.87	5430.60(9.25)	36.61	6529.30(11.33)	46.01
	0.1	106,950.50(512.80)	54.46	329.37(1.53)	27.42	10,967.00(13.82)	38.95	34,245.40(4.10)	62.46	7976.70(57.01)	48.09	5439.90(12.62)	47.10	6529.45(15.90)	50.60
	0.2	106,758.26(423.24)	55.67	329.73(1.85)	22.10	10,964.15(9.03)	38.18	34,247.40(3.50)	68.88	7964.20(51.37)	48.16	5435.40(10.98)	47.06	6530.65(11.72)	43.64
	0.3	106,726.69(335.70)	72.29	329.10(1.45)	20.09	10,971.30(16.23)	48.58	34,249.15(3.00)	46.54	7995.70(49.62)	47.75	5441.65(9.89)	41.88	6531.35(16.94)	54.23
	0.4	106,702.31(359.79)	56.44	329.51(1.79)	21.91	10,966.75(13.74)	38.46	34,251.90(2.05)	57.77	7977.65(56.80)	52.74	5437.85(9.47)	61.21	6542.45(17.16)	68.41
	0.5	106,590.41(106.27)	62.78	329.84(2.34)	16.85	10,971.95(16.13)	44.63	34,251.45(1.66)	54.16	7990.00(60.01)	45.35	5440.20(10.47)	54.10	6540.95(16.87)	46.51
	0.6	106,838.59(447.61)	55.59	329.87(2.02)	27.76	10,965.10(11.83)	46.16	34,252.20(1.83)	57.73	8010.65(54.62)	48.87	5445.00(9.14)	47.43	6541.75(14.87)	53.48
	0.7	106,805.36(486.01)	58.48	328.95(1.83)	18.86	10,964.80(11.78)	33.79	34,252.30(1.14)	52.75	7998.70(50.97)	51.47	5445.85(8.64)	52.97	6544.35(11.89)	48.30
	0.8	106,726.69(335.70)	43.21	329.06(2.13)	17.22	10,964.75(11.76)	57.51	34,251.90(1.30)	39.51	7974.90(64.72)	55.58	5441.80(9.82)	31.95	6545.85(12.59)	34.31
	0.9	106,539.06(756.19)	73.47	329.36(2.42)	20.08	10,967.85(13.51)	36.50	34,251.60(1.62)	38.47	8018.30(37.98)	48.27	5444.55(8.96)	43.34	6544.50(9.46)	61.41
	1	106,511.78(449.03)	39.02	327.97(2.48)	7.60	10,968.50(15.40)	41.86	34,251.60(1.43)	58.75	7944.55(64.39)	61.43	5444.95(8.95)	44.45	6551.55(11.07)	42.17

Table 9
Value of ρ which allows each algorithm to reach its best performance.

	in604	Decay2000_500	C2000.9	MANN_a45	keller6	frb50-23-2	frb59-26-5
ReTS-I	0.0–1.0	0.0	0.4	0.9	1.0	0.9	0.8
ReTS-II	0.1	0.0	0.5	0.7	0.9	0.7	1.0

5. Effectiveness of restart strategy

As shown in Section 3, the proposed approach uses a restart strategy to displace the search to new regions when a deep local optimum is attained by tabu search (Algorithm 1, lines 6–9). The restart strategy initializes the new starting solution of the next round of TS with either the reconstruction procedure (Section 3.2) or the random procedure (Section 3.1). The choice between these two restarting procedures is determined with a probability ρ. Intuitively, the reconstruction procedure leads the search to a nearby region (since it is guided by means of the objective function), while the random procedure diversifies more strongly the search.

In this section, we investigate the impact of the joint use of these two restart procedures by testing various probabilistic values $\rho \in \{k/10\}$ ($k \in [\![1, 10]\!]$). The two extreme values $\rho = 0$ and $\rho = 1$ correspond to the cases where only the random or the reconstruction procedure is applied. This study was based on 7 representative instances selected from the 3 benchmark sets. Each instance was solved 20 times by ReTS-I and ReTS-II with a given ρ value, each run being limited to 120 seconds.

Table 8 reports the results of this experiment. Column $ave(std)$ indicates the average and standard deviation of the best objectives for the 20 runs, and column *Time* shows the average time in seconds needed to reach the best objective values of the 20 runs. We additionally report in Table 9 the values of parameter ρ which lead to the maximum average objective values.

According to Table 8, setting ρ to small values close to 0 lead to better results on instances in604 and Decay2000_500, while high values of ρ are preferable on other instances, which indicates that the reconstruction procedure guided by the objective function is helpful to attain better solutions in most cases. This observation also emphasizes the use of $\rho = 0.7$ for the experiments of Section 4. Moreover, Table 9 shows that for each instance, ReTS-I and ReTS-II have close best ρ values, which attests the relevance of integrating both algorithms into the same search framework.

Finally, Table 8 discloses that the impact of ρ on the performance of ReTS-I and ReTS-II varies according to the instances. In particular, for MANN_a45, the result is gradually improved with increasing ρ values, reaching the best objective value when $\rho = 0.7$ for ReTS-I and 0.9 for ReTS-II respectively.

6. Conclusion and perspective

In this paper, we presented the generalized *PUSH* operator for the Maximum Vertex Weight Clique Problem (MVWCP). *PUSH(v,C)* adds to the current clique C a vertex v taken from a *candidate push set* of vertices, and removes from C any vertex which is not adjacent to v to keep the resulting clique feasible. By customizing the candidate push set, the *PUSH* operator can be used to define various dedicated neighborhoods which can be explored by any local optimization algorithm. In particular, we showed that the traditional *ADD* and *SWAP* operators as well as some restart and perturbation rules are also covered by the *PUSH* operator.

To demonstrate the usefulness of the *PUSH* operator for solving MVWCP, we introduced two restart tabu search algorithms (ReTS-I and ReTS-II) which apply *PUSH* on different candidate push sets. In ReTS-I, *PUSH* operates with the single largest candidate push set $V \backslash C$, while ReTS-II explores three customized candidate push sets. Both ReTS-I and ReTS-II also share the same restart strategy which generates, according to a probability, new starting solutions either with an objective-guided reconstruction procedure or a randomized procedure.

ReTS-I and ReTS-II were evaluated on 3 sets (DIMACS, BHOSLIB and WDP) of 142 benchmark instances. Experimental results indicated that both algorithms compete very favorably with the state-of-the-art algorithms on the tested instances in terms of computational effort and solution quality. Both algorithms are even able to find an improved best-known result (new lower bound) for one instance (frb53-24-3). In addition to these interesting results, the generality of the *PUSH* operator enables a wider application surpassing the studied tabu search procedures.

As future work, since the ρ parameter impacts the performance of both algorithms, it would be interesting to investigate ways of making this parameter self-adaptive during the search. Also, given that the idea of the proposed *PUSH* operator is rather general, it is worth of testing the idea on other similar problems like relaxed clique problems.

Acknowledgment

We are grateful to the anonymous referees for valuable suggestions and comments which helped us improve the paper. The work was partially supported by the PGMO (2014-0024H) project from the Jacques Hadamard Mathematical Foundation (Paris, France). Support for Yi Zhou from the China Scholarship Council is also acknowledged.

References

Alidaee, B., Glover, F., Kochenberger, G., & Wang, H. (2007). Solving the maximum edge weight clique problem via unconstrained quadratic programming. *European Journal of Operational Research, 181*(2), 592–597.
Ballard, D. H., & Brown, C. M. (1982). *Computer vision*. Prentice-Hall Englewood Cliff.
Benlic, U., & Hao, J. K. (2013). Breakout local search for maximum clique problems. *Computers & Operations Research, 40*(1), 192–206.
Bomze, I. R., Pelillo, M., & Stix, V. (2000). Approximating the maximum weight clique using replicator dynamics. *IEEE Transactions on Neural Networks, 11*(6), 1228–1241.
Busygin, S. (2006). A new trust region technique for the maximum weight clique problem. *Discrete Applied Mathematics, 154*, 2080–2096.
Cai, S. (2015). Balance between complexity and quality: local search for minimum vertex cover in massive graphs. *Proceedings of the twenty-fourth international joint conference on artificial intelligence* (pp. 25–31). IJCAI.
Carraghan, R., & Pardalos, P. M. (1990). An exact algorithm for the maximum clique problem. *Operations Research Letters, 9*(6), 375–382.
van Dijkhuizen, G., & Faigle, U. (1993). A cutting-plane approach to the edge-weighted maximal clique problem. *European Journal of Operational Research, 69*(1), 121–130.
Fang, Z., Li, C.-M., & Xu, K. (2016). An exact algorithm based on MaxSAT reasoning for the maximum weight clique problem. *Journal of Artificial Intelligence Research, 55*, 799–833.
Glover, F., & Laguna, M. (2013). *Tabu search*. Springer.
Grosso, A., Locatelli, M., & Pullan, W. (2008). Simple ingredients leading to very efficient heuristics for the maximum clique problem. *Journal of Heuristics, 14*(6), 587–612.
Karp, R. M. (1972). *Reducibility among combinatorial problems*. Springer.
Kumlander, D. (2004). A new exact algorithm for the maximum-weight clique problem based on a heuristic vertex-coloring and a backtrack search. In *Proceedings of the 5th international conference on modelling, computation and optimization in information systems and management sciences, citeseer* (pp. 202–208).
Lau, H. C., & Goh, Y. G. (2002). An intelligent brokering system to support multi--agent web-based 4th-party logistics. *Proceedings of the 14th IEEE international conference on tools with artificial intelligence* (pp. 154–161). IEEE.
Leyton-Brown, K., Pearson, M., & Shoham, Y. (2000). Towards a universal test suite for combinatorial auction algorithms. *Proceedings of the 2nd ACM conference on electronic commerce* (pp. 66–76). ACM.

Mannino, C., & Stefanutti, E. (1999). An augmentation algorithm for the maximum weighted stable set problem. *Computational Optimization and Applications, 14*(3), 367–381.

Neveu, B., Trombettoni, G., & Glover, F. (2004). Id walk: a candidate list strategy with a simple diversification device, principles and practice of constraint programming (CP). *Lecture notes in computer science 3258* (pp. 423–437). Springer.

Östergård, P. R. (2001). A new algorithm for the maximum-weight clique problem. *Nordic Journal of Computing, 8*(4), 424–436.

Östergård, P. R. (2002). A fast algorithm for the maximum clique problem a new algorithm for the maximum-weight clique problem. *Discrete Applied Mathematics, 120*(1–3), 197–207.

Pullan, W. (2008). Approximating the maximum vertex/edge weighted clique using local search. *Journal of Heuristics, 14*(2), 117–134.

Pullan, W., & Hoos, H. H. (2006). Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research, 25*, 159–185.

Sandholm, T. (2002). Algorithm for optimal winner determination in combinatorial auctions. *Artificial intelligence, 135*(1), 1–54.

Singh, A., & Gupta, A. K. (2006). A hybrid heuristic for the maximum clique problem. *Journal of Heuristics, 12*(1–2), 5–22.

Wang, Y., Hao, J. K., Glover, F., Lü, Z., & Wu, Q. (2016). Solving the maximum vertex weight clique problem via binary quadratic programming. *Journal of Combinatorial Optimization, 32*(2), 531–549.

Warren, J. S., & Hicks, I. V. (2006). Combinatorial branch-and-bound for the maximum weight independent set problem. *Technical report*. Texas A&M University. http://www.caam.rice.edu/~ivhicks/jeff.rev.pdf

Wu, Q., & Hao, J. K. (2015a). A review on algorithms for maximum clique problems. *European Journal of Operational Research, 242*(3), 693–709.

Wu, Q., & Hao, J. K. (2015b). Solving the winner determination problem via a weighted maximum clique heuristic. *Expert Systems with Applications, 42*(1), 355–365.

Wu, Q., & Hao, J. K. (2016). A clique-based exact method for optimal winner determination in combinatorial auctions. *Information Sciences, 334*, 103–121.

Wu, Q., Hao, J. K., & Glover, F. (2012). Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research, 196*(1), 611–634.