Noname manuscript No.
(will be inserted by the editor)

Multi-neighborhood tabu search for the maximum weight clique problem

Qinghua Wu · Jin-Kao Hao* · Fred Glover

Annals of Operations Research, Accepted: 20 March 2012

Abstract Given an undirected graph $G = (V, E)$ with vertex set $V = \{1, ..., n\}$ and edge set $E \subseteq V \times V$. Let $w : V \rightarrow Z^+$ be a weighting function that assigns to each vertex $i \in V$ a positive integer. The maximum weight clique problem (MWCP) is to determine a clique of maximum weight. This paper introduces a tabu search heuristic whose key features include a combined neighborhood and a dedicated tabu mechanism using a randomized restart strategy for diversification. The proposed algorithm is evaluated on a total of 136 benchmark instances from different sources (DIMACS, BHOSLIB and set packing). Computational results disclose that our new tabu search algorithm outperforms the leading algorithm for the maximum weight clique problem, and in addition rivals the performance of the best methods for the unweighted version of the problem without being specialized to exploit this problem class.

Keywords multi-neighborhood search · maximum weight clique · maximum clique · tabu search · heuristics

1 Introduction

Let $G = (V, E)$ be an undirected graph with vertex set $V = \{1, \ldots, n\}$ and edge set $E \subset V \times V$. A clique C of G is a subset of V such that every two vertices are pairwise adjacent, i.e., $\forall u, v \in C, \{u, v\} \in E$. A clique is maximal if it is not contained in any other clique, a clique is maximum if its cardinality is the largest among all the cliques of the graph. The maximum clique problem (MCP) asks for

Qinghua Wu
LERIA, Université d'Angers , 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
E-mail: wu@info.univ-angers.fr

Jin-Kao Hao *(Corresponding author)*
LERIA, Université d'Angers , 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
E-mail: hao@info.univ-angers.fr

Fred Glover
OptTek Systems, Inc., 2241 17th Street Boulder,CO 80302, USA
E-mail: glover@opttek.com

a maximum clique. MCP is one of the first problems shown to be NP-complete in Karp's seminal paper on computational complexity [16].

An important generalization of MCP is the maximum weight clique problem (MWCP). Given $G = (V, E)$, let $w : V \rightarrow Z^+$ be a weighting function that assigns to each vertex $i \in V$ a positive value. For a clique C of G, define its weight as $W(C) = \sum_{i \in C} w_i$. The MWCP is to determine a clique C^* of maximum weight, i.e., $\forall C \in \Omega, W(C^*) \geq W(C)$ where Ω is the set of all possible cliques of the graph. Applications of the MWCP arise in a number of domains like computer vision, pattern recognition and robotics [3].

The MCP, also called the unweighted maximum clique problem, can be considered as a special case of the MWCP where the weight of each vertex is set equal to 1 (i.e., $w : V \rightarrow \{1\}$). It is clear that a maximum clique of the MCP does not necessarily lead to a maximum weight clique of the MWCP, and the MWCP has at least the same computational complexity as the MCP.

To solve the MWCP, a number of exact algorithms have been reported in the literature (see e.g., [2,21]) which can be applied to instances of small sizes. For larger problems, various heuristic methods have been proposed to find approximative solutions. For instance, in [22], an efficient local search algorithm is presented which is based on ideas developed for the unweighted case [23]. Along with the proposed algorithm, a set of MWCP benchmark instances using the DIMACS graphs are also introduced (we use these instances in our experiments). In [20], an augmentation algorithm is described which is based on edge projections for the equivalent maximum weight stable set problem. Other representative studies include a deterministic iterated greedy construction algorithm using a nonlinear programming formulation [7], and a distributed computational network algorithm [6].

In this paper, we present a multi-neighborhood tabu search approach (denoted by MN/TS) for the maximum weight clique problem. In order to effectively explore the search space, the proposed algorithm combines three neighborhoods induced by three types of moves. The particularity of the combined neighborhood relies on the *union* of the underlying neighborhoods instead of the conventional sequential exploration of basic neighborhoods (Sections 2.3 and 2.4). At each iteration of the algorithm, our tabu search approach explores the union of these three neighborhoods and selects the overall best admissible neighboring solution. The algorithm integrates a dedicated tabu mechanism (Section 2.5) and a randomized restart strategy (Section 2.6).

The performance of the proposed MN/TS algorithm is assessed on a large set of benchmarks from the well-known DIMACS and BHOSLIB libraries and the set packing problem (Section 4). Extensive experimental tests disclose that the proposed approach finds new best solutions for 26 DIMACS instances of the maximum weight clique problem, while matching the best known solution on all but one of the others. For the unweighted maximum clique problem, MN/TS is able to attain the best known solutions for the 120 tested instances except for only two cases, rivaling the performance of the best algorithms for the MCP problem without specializing our method to exploit the unweighted class. For an additional set of 16 instances derived from the set packing problem, MN/TS is able to attain the current best-known results while discovering 2 improved results, again without being designed to exploit the set packing structure (in contrast to the methods that have produced the previous best results). An analysis is also provided to show

the relevance of the union exploration of the underlying neighborhoods (Section 5).

2 Multi-neighborhood tabu search for the MWCP

In this section, we present our multi-neighborhood tabu search (MN/TS) approach for the general MWCP. MN/TS integrates several features which are responsible for its effectiveness, including three complementary neighborhoods defined by three basic move operators. These neighborhoods are explored in a *combined manner* employing a rule that selects the most favorable neighboring solution that is admissible subject to the tabu conditions. The method is driven by a dedicated tabu list strategy employing a restart mechanism for diversification.

2.1 Search space and evaluation function

For a given MWCP instance $G = (V, E, w)$, our MN/TS algorithm explores a search space Ω composed of all possible cliques of G, i.e., $\Omega = \{C : C \subset V$ such that $\forall i, j \in C, i \neq j, \{i, j\} \in E\}$. For any solution $C \in \Omega$, its *quality* is evaluated by its weight $W(C) = \sum_{i \in C} w_i$. Given two solutions C and C', C' is better than C if and only if $W(C') > W(C)$. Our objective function (to be maximized) is thus given by: $W : \Omega \rightarrow Z^+$.

2.2 Randomized procedure for initial solutions

Our algorithm starts from an initial clique $C \in \Omega$ and then uses the tabu search procedure (Sections 2.3-2.5) to improve C by maximizing its weights. The initial solution C is constructed as follows. We first select randomly a seeding vertex i from the graph and set the current clique C to the set consisting of this single vertex. We then randomly pick another vertex $v \notin C$ subject to the stipulation that v is connected to *all* the vertices of C (i.e., v is taken from the set $\{v : v \in V \backslash C, \{v, i\} \in E, \forall i \in C\}$). This process is repeated until no such vertex v exists. This procedure is also used to initialize each restart during a run of the MN/TS algorithm (see Section 2.6). This procedure has the advantage of being simple and fast, leading to diversified initial solutions for each round of the tabu search procedure.

2.3 Basic move operators and neighborhoods

In local search, a neighborhood is typically defined by a move operator mv, which transforms a given solution C to generate a neighboring solution C', denoted by $C' = C \oplus mv$. Let $M(C)$ be the set of all possible moves which can be applied to C, then the neighborhood N of C is defined by: $N(C) = \{C' : C' = C \oplus mv, mv \in M(C)\}$.

Our MN/TS algorithm explores jointly three neighborhoods which are defined by three basic move operators (denoted by ADD, $SWAP$ and $DROP$). These move

operators are based on the definition of two vertex subsets: PA and OM relative to a given clique C.

PA is composed of the vertices that are excluded from the clique C and connected to *all* the vertices of C: $PA = \{v : v \in V \backslash C, \{v, i\} \in E, \forall i \in C\}$.

OM contains the vertices that are excluded from the clique C and connected to *all but one* vertex of C: $OM = \{v : v \in V \backslash C, |A(v) \bigcap C| = |C| - 1\}$ where $A(v) = \{j : j \in V, \{j, v\} \in E\}$ is the set of vertices adjacent to v.

The relationship between a clique C and the associated subsets PA and OM is illustrated in Fig. 1.

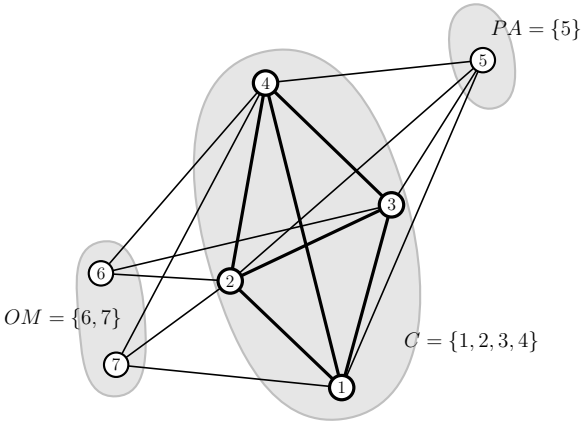

Fig. 1 A clique and its two associated subsets: $C = \{1, 2, 3, 4\}$, $PA = \{5\}$ and $OM = \{6, 7\}$.

The two subsets PA and OM just described form the basis for defining the ADD and $SWAP$ move operators while the $DROP$ move operator is defined independently of these subsets, as follows.

– $ADD(i)$: This move operator (which applies when PA is not empty) consists in adding a vertex i from the set PA to the current clique C. The neighborhood defined by this move operator is given by $N_1 = \{C' : C \oplus ADD(i), i \in PA\}$. After a $ADD(i)$ move, the change in the clique weight (i.e., the move gain denoted by Δ_i) is given by the following expression:

$$\Delta_i = w_i \tag{1}$$

where w_i is the weight associated to vertex i. Since the move gain is always positive for a ADD move, such a move always leads to an improved neighboring solution. The size of this neighborhood is clearly bounded by $O(n)$.

– *SWAP(i,j)*: This move operator (which applies when OM is not empty) consists in exchanging a vertex i from the set OM with the only vertex j of C which is not connected to i in C. The neighborhood defined by this move operator is given by $N_2 = \{C' : C \oplus SWAP(i, j),\ i \in OM, j \in C, \{i, j\} \notin E\}$.

For a given *SWAP(i,j)* move, the move gain Δ_{ij} can be conveniently computed by:

$$\Delta_{ij} = w_i - w_j \tag{2}$$

Since Δ_{ij} can be either positive or negative, a *SWAP* move can improve or deteriorate the quality of the current solution. The size of this neighborhood is bounded by $O(n)$.

– *DROP(i)*: This move operator removes a vertex i from the current clique C. The neighborhood induced by the *DROP* move can be formally defined by $N_3 = \{C' : C \backslash \{i\}, i \in C\}$.
The move gain Δ_i of dropping vertex i can be calculated by:

$$\Delta_i = -w_i \tag{3}$$

We can see that a *DROP* move always leads to a decrease to the objective function.

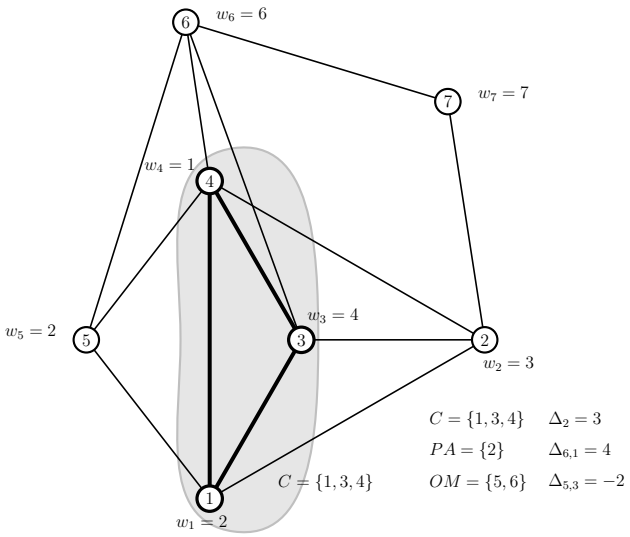

Fig. 2 From clique $C = \{1, 3, 4\}$, the *SWAP* move between vertices 6 and 1 ($SWAP(6, 1)$) leads to a solution $C_1 = \{3, 4, 6\}$, which is better than the solution $C_2 = \{1, 2, 3, 4\}$ obtained by the *ADD* move ($ADD(2)$).

2.4 Combined neighborhood and neighbor selection strategy

In the case of the unweighted maximum clique problem, *ADD* moves are always preferable to other moves (in a local sense) since they invariably increase the clique weight. However, for the weighted case (MWCP), a *SWAP* move may lead to a solution better than any solution that can be obtained by a *ADD* move. Fig. 2 shows an illustrative example where the current clique C contains three vertices.

From Fig. 2, we can see that swapping vertices 6 and 1 ($\Delta_{6,1} = w_6 - w_1 = 4$) leads to the solution $C_1 = \{3, 4, 6\}$, which is better than the solution $C_2 = \{1, 2, 3, 4\}$ obtained by adding vertex 2 ($\Delta_2 = 3$) to C.

Moreover, when no ADD move is possible ($PA = \emptyset$), a $DROP$ move may lead to a solution which is better than any solution that can be obtained by a $SWAP$ move (see Fig. 3 for an illustrative example). To summarize, for the MWCP, there is no absolute dominance of one move operator (and its neighborhood) over another move operator. The best move operator to be applied depends on the current search context and should be determined according to the context.

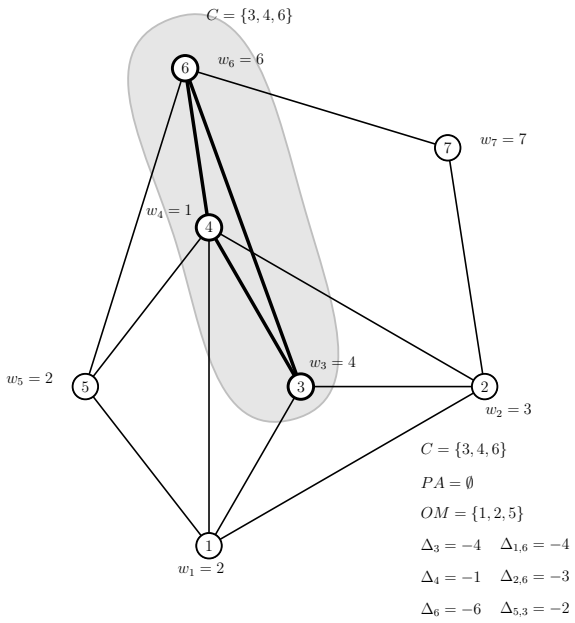

Fig. 3 From clique $C = \{3, 4, 6\}$, the $DROP$ move of dropping vertex 4 leads to a neighbor solution ($\{3, 6\}$) better than any other neighbor solution obtained by the $SWAP$ moves.

These observations lead us to create a combined neighborhood $\mathcal{N}$ which corresponds to the *union* of the three neighborhoods N_1, N_2 and N_3, denoted by $\mathcal{N} = N_1 \bigcup N_2 \bigcup N_3$. Using this union neighborhood, our tabu search algorithm selects at each iteration the most favorable move (i.e., with the largest Δ value) among all the ADD, $SWAP$ and $DROP$ moves to generate the next solution. Ties are broken at random.

2.5 Tabu list and tabu tenure management

Tabu search [13] characteristically introduces a tabu list to forbid recently visited solutions from being revisited. In our MN/TS algorithm, we adopt the following general prohibition rule: a vertex that leaves the current clique C (by a $SWAP$ or $DROP$ move) is forbidden to move back to C for the next tt iterations (tabu

tenure). A vertex that joins the clique C (by an *ADD* or *SWAP* move) is free to be removed from C without restriction.

With this prohibition rule, no tabu list is needed for the *ADD* moves. This choice can be intuitively explained by the fact that due to the objective of maximizing the clique weight, an added vertex has little chance to be removed anyway. (As noted in [13], a tabu tenure to prevent elements from being dropped should typically be smaller than one to prevent elements from being added. We have simply elected to make the smaller tenure 0.)

For the *SWAP* move, when a vertex $i \in OM$ is swapped with the only node $j \in C$ not connected to i, j is prohibited to be moved back to C for the next T_{swap} iterations while no tabu status is assigned to i. T_{swap} is tuned dynamically according to the cardinality of OM:

$$T_{swap} = random(|OM|) + T_1 \tag{4}$$

where T_1 is set equal to 7 and $random(|OM|)$ takes a random integer in the range $[1, ..., |OM|]$.

For the *DROP* move, each time a vertex i is removed from C, moving i back to C is declared tabu for the next T_1 iterations where $T_1 = 7$.

Our tabu restrictions (like most of those employed by tabu search) apply to attributes of solutions that are affected by the moves - in this case, the vertices affected by the moves. We call a move tabu if one of its attributes is tabu (hence in this case the vertex that would be added to C), and employ the common aspiration criterion that permits a move to be accepted in spite of being tabu if it produces a solution better than any found so far. A move that is not tabu or that satisfies the aspiration criterion is called *admissible*.

2.6 Multistart strategy and stop criteria

Our tabu search algorithm examines at each iteration the three neighborhoods and selects an admissible move that produces the most favorable neighboring solution. The inclusion of all three neighborhoods allows the algorithm to make a more thorough examination of the solutions around each solution. On the other hand, the tabu restrictions provide a form of local diversification by forcing the search to leave the regions already examined. To establish a more global form of diversification, and thereby reinforce the capacity of the algorithm to visit unexplored areas in the search space, we employ a multistart strategy to restart the search from new starting points. A restart is triggered each time the current search is judged to be trapped in a deep local optimum, a condition that is deemed to occur upon exceeding a maximum allowable number of consecutive iterations without improving the clique weight. We call this number the depth of the search (denoted by L).

Basically, our multistart tabu search algorithm iterates the following two steps until the stop criterion is satisfied:

1. Generate a new start point C (see Section 2.2).
2. Apply the tabu search procedure to improve the solution C until the fixed depth L is reached.

Algorithm 1 The multi-neighborhood tabu search approach for MWCP

Require: A weighted graph $G = (V, E, w)$, integer L (search depth), $Iter_{max}$ (max. allowed iterations)
Ensure: A clique C^* with its weight $W(C^*)$

Begin
$Iter = 0$ {Iteration counter}
$C^* = \emptyset$
{Each loop triggers a restart of the tabu search procedure}
while $(Iter < Iter_{max})$ **do**
  $C = Initialize()$ {Section 2.2}
  Initiate *tabu_list* {Section 2.5}
  $NI = 0$ {NI is the consecutive iterations during which $W(C)$ is not improved}
  $C_{local\_best} = C$ {C_{local_best} is the best solution during the inner *while* loop}
  {The inner *while* loop corresponds to a round of the tabu search procedure}
  while $(NI < L)$ **do**
    Construct neighborhoods N_1, N_2 and N_3 from C {Section 2.3}
    Choose an overall best allowed neighbor $C' \in N_1 \bigcup N_2 \bigcup N_3$ according to max gain criterion {Section 2.4}
    $C = C'$ {Move to the new solution}
    $NI = NI + 1$
    $Iter = Iter + 1$
    Update *tabu_list* {Section 2.5}
    if $(W(C) > W(C_{local\_best}))$ **then**
      $NI = 0$
      $C_{local\_best} = C$
    end if
  end while
  if $(W(C_{local\_best}) > W(C^*))$ **then**
    $C^* = C_{local\_best}$
  end if
end while
End
Return (Clique C^*))

The algorithm stops when it attains a predetermined maximum number of iterations ($Iter_{max}$). The complete MN/TS algorithm is described as Algorithm 1. Each outside *while* loop triggers a restart of the tabu search procedure which is realized in the inner *while* loop. The variables C and C' designate respectively the current solution and one of its neighboring solution. C_{local_best} is the best solution found during one inner *while* loop while C^* is the overall best solution found by the algorithm.

3 Discussion

The move operators *ADD*, *SWAP* and *DROP* (and particularly *ADD* and *SWAP*) have been widely used in previous studies for both the MMCP and MCP. However, previous studies have applied these operators independently and sequentially rather than making reference to their union as done here.

For instance, the PLS approach for the weighted MCP ([22]) alternates between a greedy expansion phase during which suitable vertices are added to the current clique followed by a plateau phase where vertices of the current clique are swapped with some vertices out of the clique. This strategy implicitly causes PLS to give a higher priority to *ADD* moves even if a *SWAP* move may lead to a solution better

than any *ADD* move (see Fig. 2 for an example). Such a sequential application of *ADD* and *SWAP* moves can miss favorable neighboring solutions. In short, the union neighborhood explored by MN/TS ensures a more aggressive and intensified examination of the search space, increasing the chance to find solutions of better quality. In Section 5, we give computational evidence of this assertion. Another difference between our method and PLS is that MN/TS picks the two vertices for a *SWAP(i,j)*, according to the move gain, while PLS selects a vertex i with the largest weight w_i in OM to exchange with the only vertex not connected to i in C.

Finally, for the unweighted MCP, most local search methods (such as [5, 11, 12, 14, 17, 23]) use these moves in manner similar to the way they are employed in PLS. These algorithms differ from each other chiefly in: (1) the strategies for exploring the neighborhoods, (2) the scheme of vertex selection and (3) the prohibition mechanism applied to the performed moves.

4 Experimental results

This section is dedicated to an intensive evaluation of the proposed algorithm.[1] For this purpose, we present computational results on a large panel of benchmark instances and show comparisons with state of the art algorithms when such comparisons are possible.

4.1 Benchmark instances and experimental settings

DIMACS and BHOSLIB unweighted benchmarks These test sets consist of popular benchmarks frequently used to assess *unweighted* clique algorithms. The DIMACS benchmark set was established for the Second DIMACS Implementation Challenge [15]. This set comprises 80 instances from a variety of real applications. The set also includes graphs generated randomly and graphs whose maximum clique has been hidden by incorporating low-degree vertices. These problem instances range in size from 50 vertices and 1000 edges to 3300 vertices and 5000000 edges.

The set of 40 BHOSLIB (Benchmark with Hidden Optimum Solutions) instances arose from the SAT'04 Competition. The BHOSLIB instances were translated from hard random SAT instances and have been known to be hard both theoretically and practically for maximum clique algorithms. The BHOSLIB benchmarks have been widely used in the recent literature to test new MCP heuristics. The full BHOSLIB set of instances is available from `http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm`.

BHOSLIB-W and DIMACS-W weighted benchmarks These test instances consist of benchmarks used to assess algorithms for the maximum *weight* clique problem. The weighted DIMACS-W benchmarks were obtained from the DIMACS benchmark instances by allocating weights to vertices. There are different ways to define the weighting function. We adopt the method described in [22]: for each

[1] The source code of our MN/TS algorithm is publicly available at: `http://www.info.univ-angers.fr/pub/hao/clique.html`.

vertex i, w_i is set equal to $(i \ mod \ 200) + 1$. Similarly, we apply the same weighting function to the (unweighted) BHOSLIB benchmark instances to obtain the weighted instances (denoted by BHOSLIB-W benchmarks).

Structured benchmarks from set packing These are a set of 16 instances derived from the set packing problem [18] with sizes ranging from 1000 to 2000.

Experimental settings Our MN/TS algorithm is programmed in C and compiled using GNU GCC on a PC with 2.83 GHz CPU and 8G RAM. Like [22] and given the stochastic nature of the MN/TS algorithm, each instance is solved 100 times independently by MN/TS with different random seeds. The maximum allowed iterations $Iter_{max}$ (see Algorithm 1) per run and per instance is set equal to 10^8. For the search depth L (see Section 2.6), we use $L = 4000$ for the instances of the weighted case (MWCP). For the unweighted case (MCP), we use $L = 10^4$ except for the brock and san families (DIMACS) for which L is equal to 100.

4.2 Experimental results for the maximum weight clique problem

In Tables 1 and 2, we show respectively the computational results of our algorithm on the set of 80 DIMACS-W instances and on the set of 40 BHOSLIB-W benchmarks. Columns 2–3 give the features of each tested instance: the number of vertices (Node) and the largest known clique size for the graph. In columns 4–9, we give the same computational statistics as in [22] (our main reference algorithm): the maximum weight obtained by MN/TS over the 100 independent trials (W_{best}), the cardinality of the obtained maximum weighted clique ($|C|$), the average weight over the 100 trials (W_{avg}), the number of successful trials in which MN/TS reached W_{best} ($Success$), the average time ($AvgTime$) and the average iterations ($Iteration$) over these successful trials.

Table 1: Results obtained by MN/TS on the 80 DIMACS-W benchmarks. The weight w_i of each vertex i is set equal to $(i \ mod \ 200) + 1$ according to [22]. Statistics are based on 100 executions of MN/TS on each benchmark instance.

Instance	Node	ω	W_{best}	$\|C\|$	W_{avg}	Success	AvgTime	Iteration
brock200_1	200	21*	2821	19	2821	100	< 0.01	2341
brock200_2	200	12*	1428	9	1428	100	< 0.01	2120
brock200_3	200	15*	2062	13	2062	100	< 0.01	2025
brock200_4	200	17*	2107	13	2107	100	< 0.01	2482
brock400_1	400	27*	3422	21	3422	100	0.03	7283
brock400_2	400	29*	3350	21	3350	100	0.03	7601
brock400_3	400	31*	3471	23	3471	100	0.03	7887
brock400_4	400	33*	3626	33	3626	100	4.70	1955245
brock800_1	800	23*	3121	20	3121	100	0.05	7658
brock800_2	800	24*	3043	18	3043	100	0.20	28159
brock800_3	800	25*	3076	20	3076	100	0.08	14002
brock800_4	800	26*	2971	26	2971	100	49.70	9033704
C125.9	125	34*	2529	30	2529	100	0.02	15579
C250.9	250	44*	5092	40	5092	100	0.06	23974
C500.9	500	57	6955	48	6955	100	0.07	21806
C1000.9	1000	68	9254	61	9254	100	8.9	3378709
C2000.5	2000	16	2466	14	2466	100	1.84	62469
C2000.9	2000	80	10999	72	10971.92	22	168.11	36667727
C4000.5	4000	18	2792	16	2792	100	80.56	1344781
DSJC500.5	500	13*	1725	12	1725	100	0.04	5471
DSJC1000.5	1000	15*	2186	13	2186	100	0.20	14184
keller4	171	11*	1153	11	1153	100	0.03	10257
keller5	776	27	3317	27	3317	100	3.17	445956
keller6	3361	59	8062	56	7939.49	5	606.15	53687525
MANN_a9	45	16*	372	16	372	100	< 0.01	2813
MANN_a27	378	126*	12281	126	12273.28	1	88.28	15724873
MANN_a45	1035	345*	34192	340	34172.9	1	390.58	25104191
MANN_a81	3321	1100	111128	1094	111108.31	1	832.24	18550423

Table 1 – continued from previous page

Instance	Node	ω	W_{best}	$\overline{C}$	W_{avg}	Success	AvgTime	Iteration
hamming6-2	64	32*	1072	32	1072	100	< 0.01	2305
hamming6-4	64	4*	134	4	134	100	< 0.01	2011
hamming8-2	256	128*	10976	128	10976	100	< 0.01	10742
hamming8-4	256	16*	1472	16	1472	100	< 0.01	2086
hamming10-2	1024	512*	50512	512	50512	100	0.92	118180
hamming10-4	1024	40	5129	35	5129	100	2.21	407528
gen200_p0.9_44	200	44*	5043	37	5043	100	< 0.01	3203
gen200_p0.9_55	200	55*	5416	52	5416	100	0.33	146200
gen400_p0.9_55	400	55	6718	47	6718	100	0.15	47748
gen400_p0.9_65	400	65	6940	48	6940	100	0.04	12861
gen400_p0.9_75	400	75	8006	75	8006	100	0.88	264342
c-fat200-1	200	12*	1284	12	1284	100	0.14	74106
c-fat200-2	200	24*	2411	23	2411	100	0.06	33207
c-fat200-5	200	58*	5887	58	5887	100	0.02	13569
c-fat500-1	500	14*	1354	12	1354	100	0.73	162912
c-fat500-2	500	26*	2628	24	2628	100	0.33	71620
c-fat500-5	500	64*	5841	62	5841	100	0.14	34146
c-fat500-10	500	126*	11586	124	11586	100	0.06	18163
johnson8-2-4	28	4*	66	4	66	100	< 0.01	3762
johnson8-4-4	70	14*	511	14	511	100	< 0.01	2038
johnson16-2-4	120	8*	548	8	548	100	0.23	101807
johnson32-2-4	496	16*	2033	16	2033	100	0.53	113880
p_hat300-1	300	8*	1057	7	1057	100	0.02	2986
p_hat300-2	300	25*	2487	20	2487	100	< 0.01	3141
p_hat300-3	300	36*	3774	29	3774	100	0.02	6862
p_hat500-1	500	9*	1231	8	1231	100	0.03	2707
p_hat500-2	500	36*	3920	31	3920	100	< 0.01	732
p_hat500-3	500	50	5375	42	5375	100	0.10	28840
p_hat700-1	700	11*	1441	9	1441	100	0.03	2446
p_hat700-2	700	44*	5290	40	5290	100	0.02	3542
p_hat700-3	700	62	7565	58	7565	100	0.38	90841
p_hat1000-1	1000	10	1514	9	1514	100	0.08	6153
p_hat1000-2	1000	46	5777	40	5777	100	0.11	13405
p_hat1000-3	1000	68	8111	58	8111	100	1.23	235604
p_hat1500-1	1500	12*	1619	10	1619	100	0.06	3271
p_hat1500-2	1500	65	7360	58	7360	100	0.82	75206
p_hat1500-3	1500	94	10321	84	10319.92	96	188.38	18432419
san200_0.7_1	200	30*	3370	30	3370	100	0.17	44988
san200_0.7_2	200	18*	2422	14	2422	100	0.02	4127
san200_0.9_1	200	70*	6825	70	6825	100	0.13	57024
san200_0.9_2	200	60*	6082	60	6082	100	0.21	79785
san200_0.9_3	200	44*	4748	34	4748	100	< 0.01	5745
san400_0.5_1	400	13*	1455	8	1455	100	0.06	5685
san400_0.7_1	400	40*	3941	40	3941	100	13.68	1688081
san400_0.7_2	400	30*	3110	30	3110	100	43.34	5330560
san400_0.7_3	400	22*	2771	18	2771	100	0.05	9267
san400_0.9_1	400	100*	9776	100	9776	100	1.29	306069
san1000	1000	15*	1716	9	1716	100	13.01	471338
sanr200-0.7	200	18*	2325	15	2325	100	< 0.01	2049
sanr200-0.9	200	42*	5126	36	5126	100	< 0.01	2168
sanr400-0.5	400	13*	1835	11	1835	100	0.02	2941
sanr400-0.7	400	21	2992	18	2992	100	< 0.01	2745

For problems in the MWCP class, studies in the literature are often based on DIMACS-W instances (different weighting functions may be used). We are unaware of studies reporting computational results on the BHOSLIB-W benchmarks. For this reason, our comparisons reported in the next section are based on DIMACS-W benchmarks (as well as a set of instances from the set packing problem) while our results on the BHOSLIB-W benchmarks can serve as a basis for performance assessment of other MWCP algorithms.

Table 2: Results obtained by MN/TS on the 40 BHOSLIB-W benchmarks. The weight w_i of each vertex i is set equal to $(i \ mod \ 200)+1$ according to [22]. Statistics are based on 100 executions of MN/TS on each benchmark instance.

Instance	Node	ω	W_{best}	$\overline{C}$	W_{avg}	Success	AvgTime	Iteration
frb30-15-1	450	30*	2990	27	2990	100	0.35	162927
frb30-15-2	450	30*	3006	28	3006	100	3.45	1628915
frb30-15-3	450	30*	2995	27	2995	100	4.72	2147505
frb30-15-4	450	30*	3032	28	3032	100	0.12	53148
frb30-15-5	450	30*	3011	27	3011	100	3.01	1404617
frb35-17-1	595	35*	3650	33	3650	100	25.80	10949043
frb35-17-2	595	35*	3738	33	3736.84	96	72.09	36780076

Table 2 – continued from previous page

Instance	Node	ω	W_{best}	$\|C\|$	W_{avg}	Success	AvgTime	Iteration
frb35-17-3	595	35*	3716	33	3716	100	7.72	3208297
frb35-17-4	595	35*	3683	35	3678.31	77	94.03	46627497
frb35-17-5	595	35*	3686	33	3686	100	8.09	3306241
frb40-19-1	760	40*	4063	37	4062.15	83	85.57	12557557
frb40-19-2	760	40*	4112	36	4111.16	87	134.58	29716520
frb40-19-3	760	40*	4115	36	4108.30	19	215.98	44792105
frb40-19-4	760	40*	4136	37	4135.56	89	96.65	13321879
frb40-19-5	760	40*	4118	36	4117.6	90	178.89	31692738
frb45-21-1	945	45*	4760	41	4748.66	44	126.26	41702954
frb45-21-2	945	45*	4784	42	4775.86	47	228.03	42332553
frb45-21-3	945	45*	4765	43	4756.90	26	125.35	42132692
frb45-21-4	945	45*	4799	42	4772.41	43	174.73	34953953
frb45-21-5	945	45*	4779	43	4777.38	82	193.82	35802284
frb50-23-1	1150	50*	5494	47	5484.74	6	186.62	52803333
frb50-23-2	1150	50*	5462	47	5434.14	3	149.66	45053333
frb50-23-3	1150	50*	5486	47	5480.29	53	158.71	45289811
frb50-23-4	1150	50*	5454	46	5451.69	9	176.41	49915555
frb50-23-5	1150	50*	5498	47	5495.70	89	110.85	36065699
frb53-24-1	1272	53*	5670	50	5637.94	5	233.22	54638030
frb53-24-2	1272	53*	5707	48	5676.56	6	145.22	40515069
frb53-24-3	1272	53*	5640	49	5610.79	15	215.79	62672666
frb53-24-4	1272	53*	5714	50	5645.61	7	449.39	73105032
frb53-24-5	1272	53*	5659	49	5628.77	5	294.00	47012340
frb56-25-1	1400	56*	5916	53	5836.85	3	308.90	49212581
frb56-25-2	1400	56*	5872	52	5807.70	1	73.25	17174823
frb56-25-3	1400	56*	5859	51	5799.38	1	181.93	47664235
frb56-25-4	1400	56*	5892	51	5839.16	3	104.58	28605284
frb56-25-5	1400	56*	5839	52	5768.39	1	322.70	91502378
frb59-26-1	1534	59*	6591	55	6547.53	3	166.20	42284765
frb59-26-2	1534	59*	6645	56	6567.07	3	212.49	54746666
frb59-26-3	1534	59*	6608	55	6514.18	1	232.77	60188544
frb59-26-4	1534	59*	6592	54	6498.37	1	318.39	47624522
frb59-26-5	1534	59*	6584	53	6522.57	1	161.47	30820580

4.3 Comparative results for the *weighted* maximum clique problem

In order to show the relative effectiveness of our MN/TS for the MWCP, we first compare MN/TS with two state of the art algorithms from the literature [22, 20]. The main comparison criterion is the quality of the solutions found. Due to the differences among the programming languages, data structures, compiler options and computers, computing times are provided only for indicative purposes. Since the reference algorithms report results only for DIMCAS-W benchmarks, our first comparisons are based on this set of instances.

Table 3: Comparative results between MN/TS and PLS [22] on the set of 80 DIMACS-W benchmarks. The weight of each vertex i is set equal to $(i \ mod \ 200)+1$. Statistics are based on 100 trials of each algorithm. An entry with "-" for PLS means that PLS was terminated because of excessive CPU time. An entry with "$< \epsilon$" signifies that the average CPU time required by PLS was less than 0.01 seconds. MN/TS finds improved solutions for 13 instances (in bold).

Instance	MS/TS			PLS [22]			$\Delta(MS/TS-PLS)$
	W_{best}	$Success$	$CPU(s)$	W_{best}	$Success$	$CPU(s)$	
brock200_1	2821	100	< 0.01	2821	100	0.19	0
brock200_2	1428	100	< 0.01	1428	100	0.02	0
brock200_3	2062	100	< 0.01	2062	100	0.01	0
brock200_4	2107	100	< 0.01	2107	100	0.70	0
brock400_1	3422	100	0.03	3422	32	437.19	0
brock400_2	3350	100	0.03	3350	61	415.95	0
brock400_3	3471	100	0.03	3471	100	12.04	0
brock400_4	3626	100	4.70	3626	100	0.05	0
brock800_1	3121	100	0.05	3121	100	31.46	0
brock800_2	3043	100	0.20	3043	69	893.42	0
brock800_3	3076	100	0.08	3076	100	3.35	0
brock800_4	2971	100	49.70	2971	100	3.77	0
C125.9	2529	100	0.02	2529	100	8.08	0
C250.9	5092	100	0.06	5092	17	247.69	0
C500.9	**6955**	100	0.07	6822	-	-	133
C1000.9	**9254**	100	8.90	8965	5	344.74	289

Table 3 – continued from previous page

Instance	MN/TS			PLS			$\Delta(MN/TS-PLS)$
	W_{best}	$Success$	$CPU(s)$	W_{best}	$Success$	$CPU(s)$	
C2000.5	2466	100	1.84	2466	18	711.27	0
C2000.9	**10999**	22	168.11	10028	-	-	971
C4000.5	2792	100	80.56	2792	-	-	0
DSJC500.5	1725	100	0.04	1725	100	0.95	0
DSJC1000.5	2186	100	0.20	2186	100	47.76	0
keller4	1153	100	0.03	1153	100	0.02	0
keller5	3317	100	3.17	3317	100	119.24	0
keller6	**8062**	5	606.15	7382	-	-	680
MANN_a9	372	100	< 0.01	372	100	$< \epsilon$	0
MANN_a27	**12281**	1	88.28	12264	-	-	17
MANN_a45	**34192**	1	390.58	34129	-	-	63
MANN_a81	**111128**	1	832.24	110564	-	-	564
hamming6-2	1072	100	< 0.01	1072	100	$< \epsilon$	0
hamming6-4	134	100	< 0.01	134	100	$< \epsilon$	0
hamming8-2	10976	100	< 0.01	10976	100	$< \epsilon$	0
hamming8-4	1472	100	< 0.01	1472	100	$< \epsilon$	0
hamming10-2	50512	100	0.92	50512	100	$< \epsilon$	0
hamming10-4	**5129**	100	2.21	5086	1	1433.07	43
gen200_p0.9_44	5043	100	< 0.01	5043	100	4.44	0
gen200_p0.9_55	5416	100	0.33	5416	100	0.05	0
gen400_p0.9_55	6718	100	0.15	6718	2	340.11	0
gen400_p0.9_65	**6940**	100	0.04	6935	4	200.79	5
gen400_p0.9_75	8006	100	0.88	8006	100	$< \epsilon$	0
c-fat200-1	1284	100	0.14	1284	100	$< \epsilon$	0
c-fat200-2	2411	100	0.06	2411	100	$< \epsilon$	0
c-fat200-5	5887	100	0.02	5887	100	$< \epsilon$	0
c-fat500-1	1354	100	0.73	1354	100	$< \epsilon$	0
c-fat500-2	2628	100	0.33	2628	100	0.01	0
c-fat500-5	5841	100	0.14	5841	100	$< \epsilon$	0
c-fat500-10	11586	100	0.06	11586	100	$< \epsilon$	0
johnson8-2-4	66	100	< 0.01	66	100	$< \epsilon$	0
johnson8-4-4	511	100	< 0.01	511	100	$< \epsilon$	0
johnson16-2-4	548	100	0.23	548	100	$< \epsilon$	0
johnson32-2-4	2033	100	0.53	2033	100	44.68	0
p_hat300-1	1057	100	0.02	1057	100	0.01	0
p_hat300-2	2487	100	< 0.01	2487	100	19.36	0
p_hat300-3	3774	100	0.02	3774	47	418.11	0
p_hat500-1	1231	100	0.03	1231	100	0.42	0
p_hat500-2	3920	100	< 0.01	**3925**	-	-	-5
p_hat500-3	**5375**	100	0.10	5361	-	-	14
p_hat700-1	1441	100	0.03	1441	100	0.20	0
p_hat700-2	5290	100	0.02	5290	100	78.51	0
p_hat700-3	7565	100	0.38	7565	12	718.40	0
p_hat1000-1	1514	100	0.08	1514	100	7.61	0
p_hat1000-2	5777	100	0.11	5777	87	940.62	0
p_hat1000-3	**8111**	100	1.23	7986	-	-	125
p_hat1500-1	1619	100	0.06	1619	100	48.91	0
p_hat1500-2	**7360**	100	0.82	7328	4	1056.19	32
p_hat1500-3	**10321**	96	188.38	10014	-	-	307
san200_0.7_1	3370	100	0.17	3370	100	$< \epsilon$	0
san200_0.7_2	2422	100	0.02	2422	66	397.38	0
san200_0.9_1	6825	100	0.13	6825	100	$< \epsilon$	0
san200_0.9_2	6082	100	0.21	6082	100	$< \epsilon$	0
san200_0.9_3	4748	100	< 0.01	4748	72	219.68	0
san400_0.5_1	1455	100	0.06	1455	100	200.44	0
san400_0.7_1	3941	100	13.68	3941	100	0.03	0
san400_0.7_2	3110	100	43.34	3110	100	0.05	0
san400_0.7_3	2771	100	0.05	2771	100	4.41	0
san400_0.9_1	9776	100	1.29	9776	100	$< \epsilon$	0
san1000	1716	100	13.01	1716	-	-	0
sanr200-0.7	2325	100	< 0.01	2325	100	0.62	0
sanr200-0.9	5126	100	< 0.01	5126	5	182.54	0
sanr400-0.5	1835	100	0.02	1835	100	0.67	0
sanr400-0.7	2992	100	< 0.01	2992	100	141.50	0

Table 3 summarizes the comparative results between our MN/TS and the well-known PLS algorithm [22]. For both algorithms, the number of the trials devoted to solving each instance was 100. In Table 3, we indicate the largest weights obtained by the two algorithms for each graph over the 100 independent trials (W_{best}), the number of successful trials where an algorithm reached W_{best} ($Success$), the average time (CPU) over these successful trials. Finally, column 8 indicates the difference in the largest weights obtained by MN/TS and PLS.

Table 3 discloses that, over the 80 instances tested, the quality of solutions obtained by our MN/TS algorithm matches or exceeds that of solutions obtained

by the PLS algorithm except in one case (p_hat500-2) where our method obtained a slightly worse solution. By contrast, the MN/TS method obtained strictly superior solutions on 13 out of the 80 instances (C500.9, C1000.9, C2000.9, keller6, MANN_a27, MANN_a45, MANN_a81, hamming10-4, gen400_p0.9_65, p_hat500-3, p_hat1000-3, p_hat1500-2, p_hat1500-3). For all of the remaining 66 instances on which the two algorithms attain the same largest weight (W_{best}), MN/TS has a success rate of 100%, while PLS has a 100% success rate on 52 of these instances.

According to [22], the experiments of PLS were performed on a computer that, when executing the DIMACS MC Machine Benchmark program (ftp://dimacs.rutgers.edu in directory /pub/dsj/clique), required respectively 0.31, 1.93 and 7.35 CPU seconds for the graphs r300.5, r400.5 and r500.5. Running this benchmark program on our computer leads to respectively 0.46, 2.79 and 10.44 CPU seconds for these three graphs. In other words, the computer used by PLS is slightly faster than the computer we used for our experiments. Table 3 shows that our MN/TS algorithm required in most cases less computing time to obtain solutions of the same or better quality.

To augment the above comparison, Table 4 contrasts the results of our MN/TS with those of the AugSearch algorithm reported in [20]. The authors of the AugSearch algorithm used a subset of 36 DIMACS graphs with weighting function in which the weight w_i of vertex i is set equal to $(i \ mod \ 10) + 1$. We have run our algorithm 100 times to solve each of these instances and report the computational statistics in Table 4. As demonstrated, our MN/TS algorithm attains easily all the best objective values W_{best} reported in [20] with a short computing time ranging from less than 1 second to 13 minutes. (The computing times of AugSearch are based on an IBM-RISC SYSTEM 6000 POWER station 375.) In addition, MN/TS obtains solutions better than those found by AugSearch in 11 cases out of the 36 instances.

4.4 Computational results on structured instances from set packing

In this section, we report the outcomes of testing the MN/TS algorithm on the set of 16 structured instances derived from the set packing problem [9,1], which have sizes ranging from 1000 to 2000. In [1], the set packing problem is solved via a unconstrained quadratic formulation while in [9], a dedicated GRASP heuristic was used. For the approach to convert a set packing problem into a maximum weight independent set problem, interested readers are referred to [18]. (A maximum weight independent set in a graph corresponds to a maximum weight clique in the complement of the graph.)

The results of this experiment are summarized in Table 5. Columns 1–3 give the problem identification [1]. Column 4 gives the previous best known results reported in [1] and [9], and columns 5–9 show the computational statistics of the MN/TS algorithm. Table 5 shows that MN/TS attains the previous best known results for all these 16 tested instances. Moreover, our algorithm discovers new best results for two instances (ID15 and ID16). This performance is surprising given that our MN/TS algorithm is not specifically designed for the set packing problem.

Table 4 Comparative results between MN/TS and AugSearch on 36 DIMACS *weighted* instances. The weight of each vertex i is set equal to $(i \ mod \ 10) + 1$ according to [20]. Notice that the results reported in [20] for two instances (MANN_a81 and keller6) are wrong since their respective W_{best} values are superior to their respective upper bounds (11000 and 590). MN/TS finds improved solutions for 11 instances (in bold).

Instance	MN/TS		AugSearch [20]		$\Delta(MN/TS - AugSearch)$
	W_{best}	Time	W_{best}	Time	
C125.9	215	< 0.01	215	5.28	0
C250.9	304	0.01	304	12.93	0
C500.9	**390**	0.08	385	3363.74	5
C1000.9	**491**	6.08	470	553.53	21
C2000.9	**585**	15.21	531	2430.77	54
C2000.5	**129**	2.43	113	324.07	16
DSJC500.5	**98**	0.05	94	19.19	4
DSJC1000.5	**114**	0.87	102	732.74	12
MANN_a27	867	0.50	867	0.01	0
MANN_a45	2403	100.23	2403	0.01	0
MANN_a81	7794	327.22	(18250)	0.01	-
brock200_2	**77**	< 0.01	76	0.19	1
brock200_4	114	< 0.01	114	114	0
brock400_2	178	< 0.01	178	1255.78	0
brock400_4	175	< 0.01	175	113.50	0
brock800_2	**159**	0.16	155	1841	4
brock800_4	**158**	0.13	153	803.11	5
gen200_p0.9_44	277	< 0.01	277	48.20	0
gen200_p0.9_55	293	0.57	293	37.26	0
gen400_p0.9_55	374	0.06	374	2220.10	0
gen400_p0.9_65	391	0.65	391	208.30	0
gen400_p0.9_75	401	88.88	401	2578.57	0
hamming8-4	106	< 0.01	106	0.1	0
hamming10-4	**294**	0.03	291	1499.09	3
keller4	87	< 0.01	87	2.48	0
keller5	**201**	0.21	195	1307.16	6
keller6	445	821.4	(1205)	922.18	-
p_hat300-1	71	< 0.01	71	2.38	0
p_hat300-2	180	< 0.01	180	0.40	0
p_hat300-3	260	< 0.01	260	6.60	0
p_hat700-1	77	0.02	77	36.17	0
p_hat700-2	284	0.01	284	200.45	0
p_hat700-3	418	< 0.01	418	42.95	0
p_hat1500-1	89	< 0.01	89	1113.43	0
p_hat1500-2	416	6.50	416	758.03	0
p_hat1500-3	595	1.46	595	517.60	0

4.5 Experimental results for the *unweighted* maximum clique problem

The results on the weighted instances have shown the efficacy of the MN/TS algorithm for the maximum weight clique problem. In this section, we additionally test the MN/TS algorithm on the unweighted maximum clique problem, using the DIMACS and BHOSLIB benchmark instances. For this experiment, the search depth L is set equal to 10^4 except for the brock and san graphs (DIMACS) for which L is set equal to 100.

Table 5 Computational results on the 16 weighted maximum clique instances from the set packing problem. The best known results (BKR) are published in [1] and [9]. MN/TS finds improved solutions for 2 instances (in bold).

Instance	n	m	BKR [1,9]	MN/TS				
				W_{best}	W_{avg}	Success	AvgTime	Iteration
1	1000	5000	67	67	67	100	< 0.01	987
2	1000	5000	4	4	4	100	< 0.01	2466
3	1000	5000	661	661	661	100	0.05	14412
4	1000	5000	48	48	48	100	0.05	11367
5	1000	1000	222	222	222	100	0.02	2160
6	1000	1000	15	15	15	100	0.09	9794
7	1000	1000	2260	2260	2259.38	63	164.12	54803313
8	1000	1000	175	175	175	100	0.53	81249
9	2000	10000	40	40	40	100	< 0.01	40
10	2000	10000	2	2	2	100	< 0.01	4
11	2000	10000	478	478	478	100	2.02	116784
12	2000	10000	32	32	32	100	4.17	229540
13	2000	2000	140	140	140	100	0.07	3136
14	2000	2000	9	9	9	100	0.06	1333
15	2000	2000	1784	**1811**	1806.81	6	304.52	51346666
16	2000	2000	131	**135**	135	100	4.96	1111260

Table 6: The computational results obtained by MN/TS on the 80 *unweighted* DIMACS benchmarks. For 78 of the 80 instances, MN/TS finds the previous best known results in less than 10 minutes.

Instance	Node	ω	W_{best}	W_{avg}	Success	AvgTime	Iteration
brock200_1	200	21*	21	21	100	< 0.01	3639
brock200_2	200	12*	12	12	100	0.06	27460
brock200_3	200	15*	15	15	100	0.07	37184
brock200_4	200	17*	17	17	100	0.09	53893
brock400_1	400	27*	27	27	100	10.27	2571929
brock400_2	400	29*	29	29	100	1.34	551643
brock400_3	400	31*	31	31	100	0.63	259892
brock400_4	400	33*	33	33	100	0.28	121510
brock800_1	800	23*	23	22.72	86	188.14	29580466
brock800_2	800	24*	24	23.88	96	156.47	26960764
brock800_3	800	25*	25	25	100	118.57	20949527
brock800_4	800	26*	26	26	100	62.38	10861330
C125.9	125	34*	34	34	100	< 0.01	114
C250.9	250	44*	44	44	100	< 0.01	706
C500.9	500	57	57	57	100	0.06	28868
C1000.9	1000	68	68	68	100	0.63	197084
C2000.5	2000	16	16	16	100	0.07	33553
C2000.9	2000	80	80	78.37	1	563.70	99176504
C4000.5	4000	18	18	18	100	144.37	3779319
DSJC500.5	500	13*	13	13	100	0.24	30471
DSJC1000.5	1000	15*	15	15	100	0.61	44184
keller4	171	11*	11	11	100	< 0.01	123
keller5	776	27	27	27	100	0.05	10431
keller6	3361	59	59	59	100	97.87	7407809
MANN_a9	45	16*	16	16	100	< 0.01	1853
MANN_a27	378	126*	126	126	100	3.42	564530
MANN_a45	1035	345*	340	340	6	90.58	2510419
MANN_a81	3321	1100	1090	1090	8	632.24	8550423
hamming6-2	64	32*	32	32	100	< 0.01	128
hamming6-4	64	4*	4	4	100	< 0.01	4
hamming8-2	256	128*	128	128	100	< 0.01	222
hamming8-4	256	16*	16	16	100	< 0.01	23
hamming10-2	1024	512*	512	512	100	< 0.01	1400
hamming10-4	1024	40	40	40	100	< 0.01	795
gen200_p0.9_44	200	44*	44	44	100	< 0.01	1116
gen200_p0.9_55	200	55*	55	55	100	< 0.01	396
gen400_p0.9_55	400	55	55	55	100	0.03	13285
gen400_p0.9_65	400	65	65	65	100	< 0.01	852
gen400_p0.9_75	400	75	75	75	100	< 0.01	511
c-fat200-1	200	12*	12	12	100	< 0.01	861
c-fat200-2	200	24*	24	24	100	0.07	35744
c-fat200-5	200	58*	58	58	100	< 0.01	2698
c-fat500-1	500	14*	14	14	100	0.02	4638
c-fat500-2	500	26*	26	26	100	< 0.01	1586
c-fat500-5	500	64*	64	64	100	0.02	6764
c-fat500-10	500	126*	126	126	100	< 0.01	3666

Table 6 – continued from previous page

Instance	Node	ω	W_{best}	W_{avg}	Success	AvgTime	Iteration
johnson8-2-4	28	4*	4	4	100	< 0.01	4
johnson8-4-4	70	14*	14	14	100	< 0.01	14
johnson16-2-4	120	8*	8	8	100	< 0.01	8
johnson32-2-4	496	16*	16	16	100	< 0.01	16
p_hat300-1	300	8*	8	8	100	< 0.01	94
p_hat300-2	300	25*	25	25	100	< 0.01	77
p_hat300-3	300	36*	36	36	100	< 0.01	346
p_hat500-1	500	9*	9	9	100	< 0.01	84
p_hat500-2	500	36*	36	36	100	< 0.01	124
p_hat500-3	500	50	50	50	100	< 0.01	616
p_hat700-1	700	11*	11	11	100	< 0.01	1071
p_hat700-2	700	44*	44	44	100	< 0.01	143
p_hat700-3	700	62	62	62	100	< 0.01	249
p_hat1000-1	1000	10	10	10	100	< 0.01	180
p_hat1000-2	1000	46	46	46	100	< 0.01	216
p_hat1000-3	1000	68	68	68	100	< 0.01	1630
p_hat1500-1	1500	12*	12	12	100	1.42	75661
p_hat1500-2	1500	65	65	65	100	< 0.01	998
p_hat1500-3	1500	94	94	94	100	< 0.01	1029
san200_0.7_1	200	30*	30	30	100	< 0.01	2716
san200_0.7_2	200	18*	18	18	100	0.05	16594
san200_0.9_1	200	70*	70	70	100	< 0.01	838
san200_0.9_2	200	60*	60	60	100	< 0.01	731
san200_0.9_3	200	44*	44	44	100	< 0.01	1749
san400_0.5_1	400	13*	13	13	100	0.34	37789
san400_0.7_1	400	40*	40	40	100	0.21	37048
san400_0.7_2	400	30*	30	30	100	0.35	61817
san400_0.7_3	400	22*	22	22	100	0.17	39006
san400_0.9_1	400	100*	100	100	100	< 0.01	2728
san1000	1000	15*	15	15	100	54.30	2215110
sanr200-0.7	200	18*	18	18	100	< 0.01	292
sanr200-0.9	200	42*	42	42	100	< 0.01	1251
sanr400-0.5	400	13*	13	13	100	< 0.01	3351
sanr400-0.7	400	21	21	21	100	< 0.01	1609

Table 6 shows the performance of MN/TS on the 80 DIMACS benchmarks. The different columns have the same interpretation as before. W_{best} (column 4) identifies the largest clique found by MN/TS. For 78 of the 80 instances, MN/TS finds the previous best known results in less than seven minutes. This performance matches the current best MCP algorithms like [8, 23, 24] and dominates other methods.

The outcomes of applying MN/TS to the BHOSLIB benchmark instances are displayed in Table 7, reinforcing these findings. In particular, for all of these 40 instances, MN/TS successfully obtains the known optimal solutions. In addition, for 23 instances, MN/TS finds optimal solutions with a success rate of 100%.

In sum, Tables 1 to 7 together demonstrate that our MN/TS algorithm is not only very effective for the maximum weight clique problem, but also very competitive for the conventional unweighted case for which it was not specially designed.

5 Influence of neighborhood combination

One of the most important features of a local search algorithm is certainly the definition of its neighborhood. When several neighborhoods are available, the issue of effective ways for using these neighborhoods becomes relevant [10, 19]. As previously noted, the three neighborhoods N_1, N_2 and N_3 induced respectively by the *ADD*, *SWAP* and *DROP* moves are natural components to embody in an overall choice strategy. In our case, at each iteration of our tabu search approach, we have elected to employ the combined neighborhood $N_1 \bigcup N_2 \bigcup N_3$, from which we select the admissible move (non-tabu or globally improving) yielding the largest move gain.

Table 7 The computational results obtained by MN/TS on the 40 *unweighted* BHOSLIB benchmark instances. For all the instances, MN/TS attains the known optimal solutions in less than 7 minutes.

Instance	Node	ω^*	W_{best}	W_{avg}	Success	AvgTime	Iteration
frb30-15-1	450	30	30	30	100	0.04	11933
frb30-15-2	450	30	30	30	100	0.06	16148
frb30-15-3	450	30	30	30	100	0.57	157523
frb30-15-4	450	30	30	30	100	0.03	8366
frb30-15-5	450	30	30	30	100	0.37	103338
frb35-17-1	595	35	35	35	100	1.56	600704
frb35-17-2	595	35	35	35	100	0.55	215200
frb35-17-3	595	35	35	35	100	0.08	28925
frb35-17-4	595	35	35	35	100	2.08	821830
frb35-17-5	595	35	35	35	100	0.02	88714
frb40-19-1	760	40	40	40	100	0.32	111699
frb40-19-2	760	40	40	40	100	9.15	3146725
frb40-19-3	760	40	40	40	100	1.63	560773
frb40-19-4	760	40	40	40	100	6.72	2267514
frb40-19-5	760	40	40	40	100	45.17	14755785
frb45-21-1	945	45	45	45	100	7.89	2223000
frb45-21-2	945	45	45	45	100	21.71	5824014
frb45-21-3	945	45	45	45	100	53.67	15251125
frb45-21-4	945	45	45	45	100	10.40	2867683
frb45-21-5	945	45	45	45	100	37.22	10616570
frb50-23-1	1150	50	50	49.84	84	116.92	30626106
frb50-23-2	1150	50	50	49.47	47	161.77	43081194
frb50-23-3	1150	50	50	49.15	15	214.58	55481196
frb50-23-4	1150	50	50	50	100	11.91	3136579
frb50-23-5	1150	50	50	50	100	50.90	8056548
frb53-24-1	1272	53	53	52.03	3	240.36	56242252
frb53-24-2	1272	53	53	52.30	30	209.89	48183139
frb53-24-3	1272	53	53	52.91	91	253.96	33611417
frb53-24-4	1272	53	53	52.45	45	178.01	42556535
frb53-24-5	1272	53	53	52.90	90	278.31	39408553
frb56-25-1	1400	56	56	55.22	22	174.02	40273969
frb56-25-2	1400	56	56	55.12	12	127.16	30201302
frb56-25-3	1400	56	56	55.25	25	209.48	47435029
frb56-25-4	1400	56	56	55.85	85	158.14	36531402
frb56-25-5	1400	56	56	56	100	85.57	18921353
frb59-26-1	1534	59	59	58.05	5	242.75	53070428
frb59-26-2	1534	59	59	58.01	1	396.38	86144885
frb59-26-3	1534	59	59	58.23	23	197.36	43876938
frb59-26-4	1534	59	59	58.10	11	192.45	41319157
frb59-26-5	1534	59	59	58.99	99	96.09	20416819

For traditional approaches which have been previously applied to the unweighted maximum clique problem, the basic moves consist of the addition or removal of a single vertex from the current clique. (Swap moves thus trivially decompose into two separate moves [4]). Within this setting of traditional methods for the MCP, the *ADD* moves are applied whenever possible as they are the only moves that augment the current clique. *DROP* moves are considered only when no *ADD* or *SWAP* move exists. In this section, we perform tests to apply this traditional way of combining neighborhoods to the MWCP: When admissible *ADD* moves are present, we select the one yielding the largest move gain (i.e., drawing the move from N_1). Otherwise, if admissible *SWAP* moves are present, we similarly select one of these moves with the largest gain (drawing the move from neighbor-

Table 8 The comparative results between two neighborhood combinations

Instance	$N_1 \bigcup N_2 \bigcup N_3$				$N_1 \rightarrow N_2 \rightarrow N_3$			
	W_{best}	W_{avg}	$Success$	$Iteration$	W_{best}	W_{avg}	$Success$	$Iteration$
brock800_1	3121	3121	100	7658	3121	3121	100	28550
C1000.9	9254	9254	100	3378709	9254	9187.99	43	49203658
C2000.5	2466	2466	100	62469	2466	2466	100	324754
C2000.9	10999	10971.92	22	36667727	10891	10758.24	1	96090684
keller6	8062	7939.49	5	53687525	7706	7456.32	1	68412368
hamming10-4	5129	5129	100	407528	5129	5128.96	97	30752818
p_hat500-2	3920	3920	100	732	3290	3290	100	10702
p_hat1000-3	8111	8111	100	235604	8111	8111	100	8255957
frb59-26-1	6591	6547.53	3	42284765	6591	6556.92	9	53125555
frb59-26-4	6592	6498.37	1	47624522	6575	6534.39	18	35506666

hood N_2). If none of these two types of moves is available, a *DROP* move is applied to remove from C the vertex with the minimum weight (N3). In this approach, the neighborhoods are explored sequentially, as denoted by $N_1 \rightarrow N_2 \rightarrow N_3$.

We apply our MN/TS algorithm to 10 MWCP instances from the DIMACS-W and BHOSLIB-W benchmarks to compare our $N_1 \bigcup N_2 \bigcup N_3$ neighborhood combination with the $N_1 \rightarrow N_2 \rightarrow N_3$ combination. Each version of the algorithm was run 100 times on each instance with $Iter_{max} = 10^8$. Table 8 shows that on all these 10 instances, MN/TS with $N_1 \bigcup N_2 \bigcup N_3$ matches or outperforms MN/TS with $N_1 \rightarrow N_2 \rightarrow N_3$. For three instances (C2000.9, Keller6 and frb59-26-4), MN/TS with $N_1 \bigcup N_2 \bigcup N_3$ achieves a better weight (W_{best}) than MN/TS with $N_1 \rightarrow N_2 \rightarrow N_3$. One also observes that MN/TS with $N_1 \bigcup N_2 \bigcup N_3$ requires significantly fewer iterations to reach the same W_{best}.

To augment these observations, we show in Fig. 4 the running profiles of our algorithms with $N_1 \bigcup N_2 \bigcup N_3$ and $N_1 \rightarrow N_2 \rightarrow N_3$ on the instances C1000.9 and brock800_1. A running profile is defined by the function $i \mapsto f_*(i)$ where i is the number of iterations and $f_*(i)$ is the best value of the objective function (averaged over 100 runs) known at iteration i. Such a profile gives a natural way to observe the evolution of the best values of the objective function during a search.

Fig. 4 shows that MN/TS with $N_1 \bigcup N_2 \bigcup N_3$ strongly dominates MN/TS with $N_1 \rightarrow N_2 \rightarrow N_3$ on these two test instances by obtaining a faster and better convergence to the best result.

6 Conclusion

A natural concern in local search is to identify how to exploit several different neighborhoods so as to increase the ability of the algorithm to explore the search space more effectively. In this work, we have presented a tabu search algorithm for the maximum weight clique problem based on a combined neighborhood induced by three types of moves. The algorithm explores all these moves at each iteration and selects the best admissible (non-tabu or globally improving) solution that yields the largest weight gain. The tabu mechanism creates an effective local diversification and a multistart strategy is employed to create a global diversification.

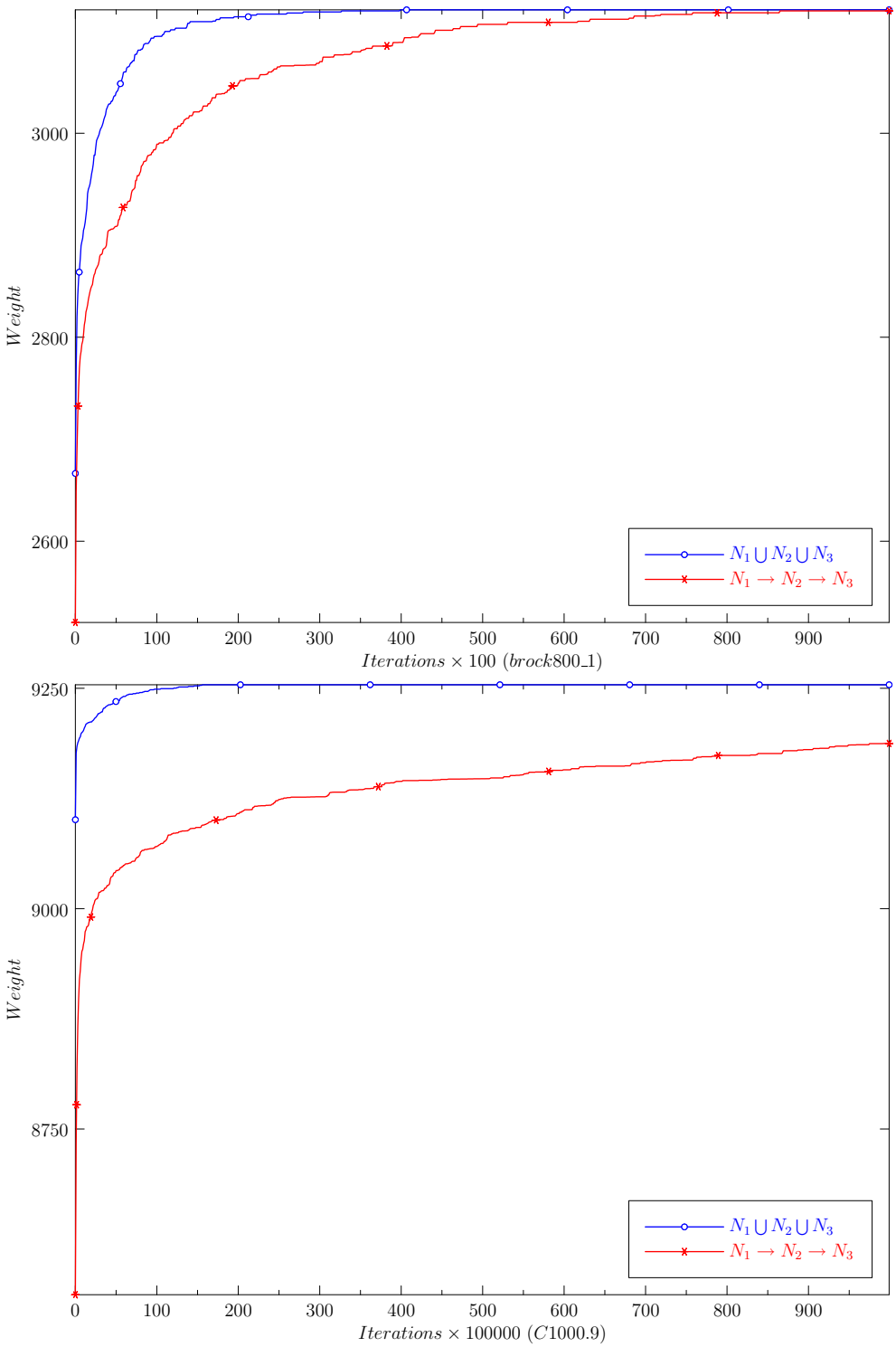

Fig. 4 Running profile of the two algorithms base on $N_1 \bigcup N_2 \bigcup N_3$ and $N_1 \rightarrow N_2 \rightarrow N_3$ on C1000.9 and brock800_1

Our proposed algorithm is evaluated on a large number of WMCP benchmarks from the BHOSLIB-W and DIMACS-W test sets (containing 40 instances and 80 instances, respectively) and and is also applied to 16 instances derived from the set partitioning problem. Compared with leading reference algorithms from the literature, our MN/TS algorithm finds new best solutions in 26 cases (24 DIMACS-W instances and 2 set packing instances). Moreover, our MN/TS approach exhibits an excellent performance when applied to the classical maximum clique problem, obtaining the best-known solutions for all the BHOSLIB instances and for 78 out of the 80 DIMACS instances. All these results are achieved with a computing time ranging from less than one second to 15 minutes on a standard laptop.

We also provided an analysis to show the relevance of the union combination of the underlying neighborhoods by comparing it to the sequential exploration of these neighborhoods. The outcomes suggest that the union combination of neighborhoods plays a key role in contributing to the effectiveness of the proposed algorithm.

Acknowledgment

We are grateful to the referees for their suggestions. The work is partially supported by the "Pays de la Loire" Region (France) within the RaDaPop (2009-2013) and LigeRO (2010-2013) projects.

References

1. Alidaee, B., Kochenberger, G.A., Lewis, K., Lewis, M., & Wang, H. (2008). A new approach for modelling and solving set packing problems, European Journal of Operational Research, 86 (2), 504-512.
2. Babel, L. (1994). A fast algorithm for the maximum weight clique problem. Computing, 52, 3138.
3. Ballard, D., & Brown, C. (1982). Computer Vision. Prentice-Hall, Englewood Cliffs.
4. Battiti, R., & Mascia, F. (2010). Reactive and dynamic local search for the Max-Clique problem: engineering effective building blocks. Computers and Operations Research, 37, 534–542.
5. Battiti, R., & Protasi, M. (2001). Reactive local search for the maximum clique problem. Algorithmica, 29(4), 610-637.
6. Bomze, I.M., Pelillo, M., & Stix, V. (2000). Approximating the maximum weight clique using replicator dynamics. IEEE Trans. Neural Networks, 11, 1228-1241.
7. Busygin, S. (2006). A new trust region technique for the maximum weight clique problem. Discrete Applied Mathematics, 154, 2080-2096.
8. Cai, S., Su, K., Chen Q. (2011). Local search with edge weighting and configuration checking heuristics for minimum vertex cover. Artificial Intelligence 175(9-10),1672–1696.
9. Delorme, X., Gandibleux, X., & Rodriguez, J. (2004). GRASP for set packing problems. European Journal of Operational Research, 153, 564-80.
10. Di Gaspero, L., & Schaerf, A. (2006). Neighborhood portfolio approach for local search applied to timetabling problems. Journal of Mathematical Modelling and Algorithms, 5(1), 65-89.
11. Friden, C., Hertz, A., & de Werra, D. (1989). Stabulus: a technique for finding stable sets in large graphs with tabu search. Computing, 42,35–44.
12. Gendreau, M., Soriano, P., & Salvail, L. (1993). Solving the maximum clique problem using a tabu search approach. Annals of Operations Research, 41, 385–403.
13. Glover, F., & Laguna, M. (1997). Tabu search. Kluwer Academic Publishers, Boston.
14. Grosso, A., Locatelli, M., & Croce, F.D. (2004). Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem. Journal of Heuristics, 10,135152.

15. Johnson, D.S., & Trick M.A. (Eds). (1996). Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. vol 26, AMS, Providence, RI.
16. Karp, R.M. (1972). Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, Complexity of Computer Computations, pages 85-103. Plenum Press, New York.
17. Katayama, K., Hamamoto, A., & Narihisa, H. (2005). An effective local search for the maximum clique problem. Information Processing Letters, 95, 503–511.
18. Kwon, R.H. (2005). Data dependent worst case bounds for weighted set packing. European Journal of Operational Research, 167(1), 68–76.
19. Lü, Z., Hao, J.K., & Glover, F. (2011). Neighborhood analysis: a case study on curriculum-based course timetabling. Journal of Heuristics, 17(2), 97–118.
20. Mannino, C., & Stefanutti, E. (1999). An augmentation algorithm for the maximum weighted stable set problem. Computational Optimization and Applications, 14, 367-381.
21. Östergård, P.R.J. (2001). A new algorithm for the maximum weight clique problem. Nordic Journal of Computing, 8,424–436.
22. Pullan, W. (2008). Approximating the maximum vertex/edge weighted clique using local search. Journal of Heuristics, 14(2), 117-134.
23. Pullan, W., & Hoos, H.H. (2006). Dynamic local search for the maximum clique problem. Journal of Artificial Intelligence Research, 25, 159-185.
24. Wu Q. & Hao J.K. (2012) An adaptive multistart tabu search approach to solve the maximum clique problem. Accepted to Journal of Combinatorial Optimization. DOI:10.1007/s10878-011-9437-8.