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

An Adaptive Multistart Tabu Search Approach to solve the Maximum Clique Problem

Qinghua Wu · Jin-Kao Hao*

Accepted to Journal of Combinatorial Optimization, 17 November 2011

Abstract Given an undirected graph $G = (V, E)$ with vertex set $V = \{1, ..., n\}$ and edge set $E \subseteq V \times V$. The maximum clique problem is to determine in G a clique (i.e., a complete subgraph) of maximum cardinality. This paper presents an effective algorithm for the maximum clique problem. The proposed multistart tabu search algorithm integrates a constrained neighborhood, a dynamic tabu tenure mechanism and a long term memory based restart strategy. Our proposed algorithm is evaluated on the whole set of 80 DIMACS challenge benchmarks and compared with five state-of-the-art algorithms. Computational results show that our proposed algorithm attains the largest known clique for 79 benchmarks.

Keywords Tabu search · maximum clique · constrained neighborhood · informed restart · combinatorial optimization

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) is to determine a maximum clique. MCP is one of the first problems shown to be NP-complete in Karp's seminal paper on computational complexity (Karp (1972)). The MCP has many applications in real-life problems, such as classification theory, coding theory, fault diagnosis, biological analysis, cluster analysis and project selection (Pardalos and Xue (2002)). The MCP is equivalent to the maximum independent (stable) set problem and is tightly related to some other problems like vertex graph coloring.

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

Given the importance of the problem, many methods have been proposed in the literature. See Pardalos and Xue (2002) for a comprehensive review of these methods. Examples of exact methods based on the general branch-and-bound approach can be found in (Balas and Yu (1986); Carraghan and Pardalos (1990); Östergärd (2002); Tomita and Seki (2003); Rebennack et al (2011)). As an alternative approach, a number of approximate methods have been developed to find near-optimal solutions to large and dense graphs. In the following, we briefly review some representative heuristics.

Battiti and Protasi (2001) propose a reactive local search (RLS) procedure which is highly successful. Pullan and Hoos (2006) introduce a very effective stochastic local search algorithm which is one of the best clique methods. Other interesting and representative methods are based on deterministic and probabilistic tabu search (Gendreau et al. (1993)), variable depth search (Katayama et al. (2005)), quadratic optimization (Busygin et al. (2002); Busygin (2006)) and genetic search (Bui and Eppley (1995); Marchiori (1998, 2002); Barbosa and Campos (2004); Singh and Gupta (2008); Zhang et al. (2005)).

Many state-of-the-art methods (such as Battiti and Protasi (2001); Pullan and Hoos (2006); Pullan (2006); Grosso et al. (2008)) are based on a general approach that alternates between a clique expansion phase and a plateau search phase. During the expansion phase, one seeks to expand a clique of size k to a new clique of size $k+1$ by adding a vertex (which is adjacent to all the vertices of the current clique). When the current clique cannot be expended, one switches to plateau search during which vertices of the current partial clique are swapped with vertices outside the partial clique. Once the current clique can be further expanded, one switches back to the expansion phase and so on. Methods using this approach differ mainly from each other in the way they perform the plateau search.

One different and somewhat neglected approach is presented in Friden et al. (1989) for the equivalent maximum stable set problem. The algorithm (called STABULUS) seeks a clique of *fixed* size k by exploring a space of vertex subsets S such that $|S| = k$. For a given candidate solution S (i.e., a vertex subset of size k), STABULUS swaps one vertex in S against another vertex in $V \backslash S$ in order to maximize the number of edges induced by the new solution. This approach was later explored successfully in Fleurent and Ferland (1996).

In this paper, we follow the basic idea of Friden et al. (1989) and develop an effective heuristic algorithm based on tabu search (Glover and Laguna (1997)). Like STABULUS, the proposed approach searches a legal k-clique within a space of subsets S (legal and illegal k-cliques) of size k (Section 2.1). Yet, the main components of our algorithm are different from those of STABULUS. In particular, To allow the algorithm to explore more efficiently the search space, the swap operations of our algorithm are limited to vertices from two critical subsets A (a constrained subset of the candidate solution S) and B (a constrained subset of $V \backslash S$) (Section 2.2.2). To escape from local optima, our algorithm applies both a deterministic selection rule (Section 2.2.3) and a probabilistic selection rule to occasionally accept deteriorating solutions (Section 2.2.4). Our algorithm uses a dedicated tabu list to prevent the algorithm from revisiting previous encountered solutions (Section 2.2.5). Finally, to allow the algorithm to explore more new search regions, our algorithm employs a frequency-based restart strategy (Section 2.3).

Our proposed algorithm is extensively tested on the commonly used DIMACS clique benchmark instances (Section 3). Experimental results show the effectiveness of this algorithm in finding large cliques within reasonable computing times. Actually, except for one case (over a total of 80 graphs), our method attains the previously best known results on all the DIMACS instances. To the best of our knowledge, no single algorithm in the literature achieves such a performance. In addition to the computational results, we provide analyses about critical components of the algorithm.

2 Adaptive multistart tabu search

2.1 Solution strategy and general procedure

As noted in Friden et al. (1989), the maximum clique problem can be approximated by finding a series of k-cliques for increasing values of k (a k-clique is a clique of size k). Each time a k-clique is found, k is incremented by one and a new (larger) k-clique is sought. This process is repeated until no k-clique can be found. The last k-clique constitutes an approximation of the maximum clique of the graph. Consequently, the maximum clique problem comes down to the problem of finding k-cliques.

Our Adaptive Multistart Tabu Search algorithm is designed for this k-clique finding problem in a graph $G = (V, E)$. In this section, we describe the general procedure of AMTS while its components are detailed in Section 2.2.

For this purpose, we first define the search space Ω that is explored by AMTS. It is composed of all the vertex subsets S of fixed size k (k-subsets) including both feasible and infeasible cliques, i.e.,

$$\Omega = \{S \subset V : |S| = k\} \tag{1}$$

For any candidate solution $S \in \Omega$, its quality is assessed by the *evaluation function* $f(S)$ that counts the number of edges induced by S:

$$f(S) = \sum_{u,v \in S} e_{uv} \tag{2}$$

where $e_{uv} = 1$ if $\{u, v\} \in E$, $e_{uv} = 0$ otherwise.

Obviously, if a candidate solution S reaches the maximal value of this function, i.e., $f(S) = k * (k-1)/2$, any two vertices of S are connected by an edge and the candidate solution S is a legal k-clique. If $f(S) < k * (k-1)/2$, there must be at least two vertices in S which are not adjacent, consequently S is not a legal k-clique.

The objective of our AMTS algorithm is then to find in Ω a solution S that reaches the maximal value of f such that $f(S) = k * (k-1)/2$. The pseudo-code of AMTS is given in Algorithm 1.

AMTS explores the space Ω by employing an optimization procedure based on tabu search (Glover and Laguna (1997)) (we denote this procedure by TS^0, which is described in Section 2.2). More specifically, AMTS generates first an initial solution (k-subset) in Ω which is built greedily in k steps from an empty set S. At each step, a vertex $v \in V \backslash S$ is added to S such that v has the maximum number of edges that are connected to the vertices of S (ties are broken randomly).

¿From this initial solution S (a k-subset), AMTS runs TS^0 to improve S by maximizing the function f (Formula 2). During a round of TS^0, the search continues whenever TS^0 finds improved solutions. If the search is judged to be stagnating (the parameter L at line 5 is used for this purpose), the current round of TS^0 is stopped and then restarted from a new starting solution (More information about the restart mechanism is given in Sections 2.2.1 and 2.3). So a AMTS run is composed of multiple rounds of

Algorithm 1 Adaptive multistart tabu search for maximum clique

```
Require: Graph $G$, Integer $k$ (clique size), Integer $L$ (search depth), Integer $Iter_{max}$ (maximum allowed iterations)
Ensure: $k$-clique if found
 1: Begin
 2: $S \leftarrow Initialize(k)$ {Initial solution}
 3: $Iter \leftarrow 0$ {Iteration counter}
 4: while ($Iter < Iter_{max}$) do
 5:    $S^* \leftarrow TS^0(S, k, L, Iter)$ {Apply the tabu search procedure TS$^0$ to improve $S$, §2.2}
 6:    if $S^*$ is a legal $k$-clique then
 7:       Return($S^*$) and Stop
 8:    else
 9:       $S \leftarrow FrequencyBasedInitialize(k)$ {Construction of a new solution $S$, §2.3}
10:    end if
11: end while
12: End
13: Return(Failure)
```

the TS^0 procedure. While each round of TS^0 examines in detail a region of the search space, each restart displaces the search to a new region.

The AMTS algorithm stops when a legal k-clique is found by TS^0, in which case the found k-clique is returned. AMTS may also stop when the total number $Iter$ of iterations attains a prefixed maximum number ($Iter_{max}$) without finding a legal k-clique. In this case, a failure is reported. $Iter_{max}$ is a user-defined parameter which specifies the maximal search effort allowed to solve a given instance. Next we present in detail the tabu search procedure TS^0.

2.2 The tabu search procedure

2.2.1 Main idea

Our tabu search procedure TS^0 is based on the well-known tabu search method (Glover and Laguna (1997)). From a general point of view, tabu search explores the search space by iteratively replacing the current solution by a new solution taken from a given neighborhood. For each iteration, tabu search selects one of the best neighbors among the neighbor solutions. With this selection rule, tabu search visits solutions of increasing quality whenever solutions of better quality exist in the neighborhood. When no improving solutions can be found (i.e., when a local optimum is reached), tabu search still moves to a best neighbor (which is also the least worse solution within the neighborhood). This strategy allows tabu search to go beyond local optima encountered and continue its exploration toward possibly better solutions. To prevent the search from comes back to an already examined solution, tabu search adopts a so-called tabu list to record previously visited solutions. For a detailed presentation of tabu search, the reader is referred to Glover and Laguna (1997).

Our TS^0 procedure adapts the tabu search method to the problem of finding k-cliques in a graph $G = (V, E)$. The pseudo-code of our TS^0 procedure is given in Algorithm 2. TS^0 operates on candidate solutions represented by k-subsets. S and S^* designate respectively the current solution and the best solution found so far (according to the evaluation function f defined in Section 2.1). I is an iteration counter used for

Algorithm 2 The tabu search procedure TS^0 for k-clique finding

```
Require: Graph $G$, Initial solution $S$, Integer $k$ (clique size), Integer $L$ (depth of tabu search),
    Integer $Iter$ (iteration counter)
Ensure: The best solution $S^*$ found by the tabu search
 1: Begin
 2: $I \leftarrow 0$ {$I$ is the consecutive iterations during which $f(S)$ is not improved}
 3: $S^* \leftarrow S$ {$S^*$ records the best solution found so far}
 4: while ($I < L$) do
 5:    if There exist improving moves in neighborhood $CN$ then
 6:       Choose a best allowed $swap(u, v)$ {§2.2.2 and 2.2.3}
 7:    else
 8:       Choose $swap(u, v)$ according to the Prob. Move Select. Rule {§2.2.4}
 9:    end if
10:    $S \leftarrow S \backslash \{u\} \cup \{v\}$ {Move to the new solution}
11:    Undate the tabu list {§2.2.5}
12:    if $S$ is a legal $k$-clique then
13:       Return $S$ and Stop
14:    end if
15:    $Iter \leftarrow Iter + 1$
16:    if $f(S) > f(S^*)$ then
17:       $S^* \leftarrow S$
18:       $I \leftarrow 0$
19:    else
20:       $I \leftarrow I + 1$
21:    end if
22: end while
23: End
24: Return (Clique $S^*$)
```

the restart of TS^0 while $Iter$ is the global iteration counter used by AMTS in its stop test (see Algorithm 1).

For each *while* loop of Algorithm 2 (lines 4-23), TS^0 moves from the current solution S (a k-subset in Ω) to a new neighbor solution (another k-subset in Ω). For this, TS^0 uses two different rules to select a dedicated vertex u in S and a specific vertex v outside S (lines 5-6 and 8-9, see Sections 2.2.2-2.2.4), and then swaps u and v to obtain a new solution (line 11). These swapped vertices are finally added in the tabu list preventing them from being selected again for the next iterations (line 12, see Section 2.2.5). If the new solution is a legal k-clique (i.e., $f(S) = k*(k-1)/2$), the algorithm stops and returns the k-clique found (lines 13-15). Otherwise, if the new solution S is better than the best solution S^* found so far ($f(S) > f(S^*)$), TS^0 updates S^* by S and continues to its next iteration (lines 17-21).

The *while* loop ends if no improved solution is found for L consecutive iterations (L is called the search depth). In this case, the search is judged to be trapped in a deep local optimum. To escape from this local optimum, AMTS restarts TS^0 from a new starting point (see Section 2.3).

In the rest of this section, we provide a detailed description of the main ingredients of the TS^0 procedure while in Section 2.3, we explain the solution construction procedure for each restart of the TS^0 procedure.

2.2.2 Constrained swap move and neighborhood

To explore the search space Ω of k-subsets (Formula (1)), one naive way is to start with any k-subset $S \in \Omega$ and subsequently swap a vertex of S with another vertex of $V \backslash S$. Clearly, such a unconstrained swap (used in Friden et al. (1989)) induces a neighborhood of size $k * (|V| - k)$ which may be quite large. More importantly such a unconstrained neighborhood is not sufficiently focused and will not enable an efficient exploration of the search space. For this reason, we introduce below the *constrained neighborhood* which is both more focused and smaller-sized.

Let $S \in \Omega$ be a candidate solution (k-subset). For each vertex $v \in V$, let $d(v)$ denote the degree of v relative to the subset S:

$d(v) = |\{i \in S \mid \{i, v\} \in E\}|$

Let *tabu_list* be the tabu list containing the vertices that are currently forbidden for migration (see Section 2.2.5).

Let $MinInS = min\{d(u)| \ u \in S, \ u \notin tabu\_list\}$ and

Let $MaxOutS = max\{d(v)| \ v \in V \backslash S, \ v \notin tabu\_list\}$

Define:

$A = \{u \in S \mid u \notin tabu\_list, \ d(u) = MinInS\}$

$B = \{v \in V \backslash S \mid v \notin tabu\_list, \ d(v) = MaxOutS\}$

Now, to obtain a neighbor solution S' from S, we swap one vertex $u \in A$ against a vertex $v \in B$. This transition (from S to S') can conveniently be characterized by a move denoted by $swap(u, v)$ and written formally as: $S' = S \oplus swap(u, v)$ or equivalently $S' = S \backslash \{u\} \cup \{v\}$. All possible swap moves induced by A and B define our constrained neighborhood $CN(S)$, i.e.,

$$CN(S) = \{S' : S' = S \backslash \{u\} \cup \{v\}, u \in A, v \in B\} \tag{3}$$

Given the definition of $d(v)$, it is easy to see that function $f(S)$ (Formula (2)) can be rewritten as:

$$f(S) = \frac{1}{2} * \sum_{i \in S} d(i) \tag{4}$$

For a given $swap(u, v)$, the move gain Δ_{uv}, i.e., the variation in the function value f induced by the swap move, can be conveniently computed by:

$$\Delta_{uv} = f(S') - f(S) = d(v) - d(u) - e_{uv} \tag{5}$$

where $e_{uv} = 1$ if $\{u, v\} \in E$, $e_{uv} = 0$ otherwise.

Consequently, for any $u \in A$ and $v \in B$, the following formulation can be concluded:

$$\Delta_{uv} = \begin{cases} MaxOutS - MinInS - 1, & \text{if } \{u, v\} \in E \\ MaxOutS - MinInS, & \text{otherwise.} \end{cases}$$

2.2.3 Move selection strategy

Obviously, the moves with $\Delta_{uv} = MaxOutS - MinInS$ are preferable since they give improvement of the evaluation function f mostly. Let T denote those swap moves with the increment value equal to $MaxOutS - MinInS$.

$$T = \{(u, v) : u \in A, v \in B, \{u, v\} \notin E, \Delta_{uv} = MaxOutS - MinInS\}$$

We apply the following strategy to determine the best neighbor solution. If T is not empty, then one pair (u, v) from T is randomly selected for swap. If T is empty, vertex u is randomly selected from A and v is randomly selected from B. Notice that in this latter case, u and v must be two adjacent vertices.

It can be easily showed that the solution $S' = S \backslash \{u\} \cup \{v\}$ obtained by swapping such a pair of vertices (u, v) is one of the best non-tabu neighbor solutions in the neighborhood $CN(S)$, i.e., for any solution $S'' \in CN(S)$, $f(S') \geq f(S'')$. In fact, if $T = \emptyset$, then for each $S'' \in CN(S)$, $f(S'') = f(S) + MaxOutS - MinInS - 1$, i.e., any solution in $CN(S)$ has the same f value and S' is among the best non-tabu solutions. If $T \neq \emptyset$, then $f(S') = f(S) + MaxOutS - MinInS$. For any other solution $S'' \in CN(S)$ (assume that $S'' = S \oplus swap(x, y)$), $f(S'') = f(S) + MaxOutS - MinInS - e_{xy} \leq f(S) + MaxOutS - MinInS = f(S')$. Once again, we can see that S' is one of the best solutions in $CN(S)$.

Finally, to prepare the next iteration of the algorithm, d, A, B, $MinInS$ and $MaxOutS$ are updated accordingly after each $swap(u, v)$ move.

2.2.4 Probabilistic diversifying move selection rule

The above move selection rule assures an exhaustive exploration of the constrained neighborhood. To encourage the search to visit new regions in the search space, we additionally employ a strategy that disables the usual move selection rule and prefers occasionally some deteriorating moves. Such an alternative strategy is triggered only in a controlled and probabilistic manner when the current solution S corresponds to a local optimum, i.e., for each allowed $swap(u, v)$, the new solution $S' = S \backslash \{u\} \cup \{v\}$ is not better than the current solution S ($f(S') \leq f(S)$). In this case, we apply the following Probabilistic Move Selection Rule (PMSR).

- With a low probability $P = min\{\frac{l+2}{|V|}, 0.1\}$ where $|V|$ is the order of the graph and $l = k * (k-1)/2 - f(S)$, select a (much worse) $swap(u, v)$ as follows. Pick u at random from S and pick v in $V \backslash S$ such that $d(v) < \lfloor k * \rho \rfloor$, where ρ is the density of the graph.
- With probability 1-P, select one best allowed $swap(u, v)$ according to the usual selection strategy defined in Section 2.2.3.

This strategy provides a way to allow the search to occasionally go to another region when no better solution can be found around the current solution.

2.2.5 Tabu list and tenure management

To define our tabu list, first recall that a neighbor solution of S is characterized by a pair of (u, v) where u is a specific vertex in $A \subset S$ and v outside S. To prevent the search from revisiting S, when a $swap(u, v)$ move is performed, vertex u is added in a data structure called tabu list and remains in the list for the next T_u iterations (called tabu tenure, see Glover and Laguna (1997)). We call vertex u tabu and forbid the search to add u back to a solution during the period fixed by T_u. Similarly, vertex v is marked tabu for the next T_v iterations, during which v cannot be removed from the solution. We call a $swap(u, v)$ move tabu if at least one of the two implied vertices is marked tabu.

Inspired by the tabu mechanism proposed in Galinier and Hao (1999), the tabu tenures T_u and T_v are dynamically adjusted by a function depending on the evaluation function $f(S)$. More precisely, let $l_1 = k * (k-1)/2 - f(S)$, $l = min\{l_1, 10\}$. Then, T_u and T_v are defined respectively as follows.

$T_u = l + Random(C)$ and

$T_v = 0.6 * l + Random(0.6 * C)$
where $C = max\{\lfloor k/40 \rfloor, 6\}$ are two parameters and the function $Random(X)$ returns randomly an integer number in $\{0, \ldots, X-1\}$. It is clear that $T_u > T_v$ holds.

The first part of the tabu tenure of T_u can be explained by the fact that a solution with a small evaluation function value should have a longer tabu tenure to escape from the local optimum trap. Since the exact value of the tabu tenure is unknown, the second part of T_u and T_u provides a random adjustment.

The reason for $T_u > T_v$ is that preventing vertices in the current solution S from being removed is much more restrictive than preventing vertices outside S from being added to S, since in general there are much fewer vertices contained in S than those outside S. In addition, preventing vertices added to S from being removed for a relatively long time can significantly inhibit available choices. Hence the tenure for the added vertex v should be made smaller by comparison to the removed vertex u.

In order to implement the tabu list, a vector *tabu_list* of $|V|$ elements is used. As suggested in Glover and Laguna (1997), each element $tabu\_list(i)$ $(1 \le i \le |V|)$ records $T_i + I$, where I is the current number of iterations (Algorithm 2) and T_i is the tabu tenure for vertex i. In this way, it is very easy to know if a vertex i is tabu or not at iteration j: if $tabu\_list(i) > j$, vertex i is forbidden to move; otherwise, i can be moved without restriction.

Finally, at each iteration, the tabu status of a move is canceled if the move leads to a better solution than the best solution S^* encountered so far.

2.3 A frequency-based strategy for new solution generation

To encourage the AMTS algorithm to explore new regions in the search space, we repeat the tabu search procedure TS^0 from different starting points. (This is what the term multistart means). Recall that a restart is triggered when TS^0 cannot find an improved solution during L consecutive iterations (Section 2.2.1).

To build a new initial solution for each TS^0 restart, we devise an informed procedure guided by a long-term frequency memory. In this memory, we keep track of the number of times a vertex has been moved during the search. To maintain the frequency g_i of vertex i, we use the following rules.

1. Initially, set $g_i = 0$ for each vertex $i \in V$.
2. Subsequently, during the search, each time vertex i is removed from or put into the current solution S, the frequency counter g_i of vertex i is incremented, $g_i = g_i + 1$.
3. If for all $i \in V$, $g_i > k$, then we reset $g_i = 0$ for all $i \in V$. This mechanism refreshes the memory over time and avoids the situation where a vertex is definitively prevented from being selected by the solution construction procedure (see below).

Given this frequency information, we create the new initial solution S for a restart as follows. Initialize S by randomly adding a vertex having the smallest frequency value in V and then repeat the above step until S contains exactly k vertices. For each step, select a vertex $v \in V \backslash S$ such that v has the maximum number of edges that connect to S. If several vertices satisfy the above criterion, select the vertex with the smallest frequency value (less moved). If there are still several vertices that satisfy the two criteria, select one of these vertices randomly.

Notice that if ATMS is given a maximum of allowed $Iter_{max}$ iterations, ATMS may perform at most $Iter_{max}/L$ restarts during its run. A small (respectively large) L

value implies more (respectively less) restart of the TS^0 procedure. We show a study of the influence of L on the performance of the AMTS algorithm.

3 Experimental results

3.1 DIMACS Challenge Benchmark

In this section, we present an extensive evaluation of our AMTS method using the set of second DIMACS Challenge Benchmark instances (Johnson and Trick (1996)). We also make comparisons with five state-of-the-art maximum clique algorithms from the literature.

The DIMACS Challenge Benchmark set comprises 80 graphs from a variety of applications such as coding theory, fault diagnosis problems, keller's conjecture on tilings using hypercubes and the Steiner triple problem. In addition, the set includes graphs generated randomly and graphs where the maximum clique has been hidden by incorporating low-degree vertices. The sizes of these instances range from less than 50 vertices and 1000 edges to greater than 3300 vertices and 5000000 edges. Columns 1 and 2 of Table 1 show the name and size of each graph.

Our AMTS algorithm[1] is programmed in C, and compiled using GNU GCC on a PC with 2.61 GHz CPU and 2G RAM.

3.2 Experimental settings

We report our computational results based on the parameters values given here, even though fine-tuning the parameters would lead to improved results.

Parameter setting. The two main parameters for AMTS are the number of allowed iterations ($Iter_{max}$) for each run and the search depth L of TS^0 (see Section 2.3). Since AMTS stops when a legal k-clique is found, $Iter_{max}$ can be safely given a very large value. In this paper, we use $Iter_{max} = 10^8$ as in Pullan and Hoos (2006) for their DLS-MC algorithm which is our main reference. Notice that for many graphs, AMTS attains a legal k-clique with much fewer iterations and stops long before reaching 10^8 iterations.

As to the search depth L, it is set equal to $|V| * k$ except for the structured brock and san graphs for which smaller values $4 * k$ are used. As a general rule, it is preferable to restart more frequently AMTS for structured graphs (by using a small L) in contrast to random graphs for which L should be set to a larger value. The effect of L on the algorithm is studied in Section 4.1.

Finally, since a maximum clique in a graph G is a maximum independent set in the complementary graph $\overline{G}$, when the density of G is greater than 0.5, it is transformed to its complement and AMTS is employed to solve the related maximum independent set problem.

[1] The source code of AMTS is available online at: `http://www.info.univ-angers.fr/pub/hao/amts.html`.

3.3 Computational results

Given the stochastic nature of our AMTS algorithm, we run the algorithm 100 times on each DIMACS benchmark instance with different random seeds, like in Pullan (2006); Pullan and Hoos (2006); Katayama et al. (2005); Battiti and Protasi (2001). To run AMTS on a graph, we set k to be the largest known clique size reported in the literature. During a AMTS run, legal cliques of size $k-1$ and $k-2$ are also recorded. These $k-1$ and $k-2$ cliques are reported if no k-clique is found for at least one of the 100 AMTS runs.

Table 1: The results obtained by AMTS on the set of 80 DIMACS benchmarks based on 100 independent runs per instance. The maximum known clique size for each instance is shown in the ω column (marked with an asterisk symbol when ω is proven to be optimal). Quality is shown in the form $a-b-c$ (column 4, see explanation). AvgTime is the CPU time in seconds, averaged over all successful runs. AvgSize is the clique size averaged over the 100 runs. The last column indicates the *total run time of the 100 runs* of AMTS for each instance. In 95% cases where a 100% success rate is reached, one single run suffices to attain the largest clique size reported in the literature.

Instance	Node	ω	Quality	AvgSize	AvgTime	Iter/sec	TotalTime
brock200_1	200	21*	100-0-0	21	0.0136	280013	13.6
brock200_2	200	12*	100-0-0	12	0.3625	270770	36.25
brock200_3	200	15*	100-0-0	15	0.0105	272734	1.05
brock200_4	200	17*	100-0-0	17	1.7582	272728	175.82
brock400_1	400	27*	100-0-0	27	37.7739	187507	3777.39
brock400_2	400	29*	100-0-0	29	1.1818	187515	118.18
brock400_3	400	31*	100-0-0	31	1.7909	157902	179.09
brock400_4	400	33*	100-0-0	33	0.5956	146373	59.56
brock800_1	800	23*	98-0-2	22.96	234.6277	85714	25326.85
brock800_2	800	24*	100-0-0	24	33.1439	85649	3314.39
brock800_3	800	25*	100-0-0	25	52.3981	78950	5239.81
brock800_4	800	26*	100-0-0	26	15.2340	70768	1523.40
C125.9	125	34*	100-0-0	34	0.0018	400214	0.18
C250.9	250	44*	100-0-0	44	0.0058	336700	0.58
C500.9	500	57	100-0-0	57	0.1263	206611	12.63
C1000.9	1000	68	100-0-0	68	1.1471	181180	114.71
C2000.5	2000	16	100-0-0	16	0.6611	31685	66.11
C2000.9	2000	80	1-93-6	78.95	450.0996	86199	115300.62
C4000.5	4000	18	100-0-0	18	126.6315	15422	12663.15
DSJC500.5	500	13*	100-0-0	13	0.0071	106723	0.71
DSJC1000.5	1000	15*	100-0-0	15	0.3113	59241	31.13
keller4	171	11*	100-0-0	11	< 0.0001	212000	0.01
keller5	776	27	100-0-0	27	0.0565	120772	5.65
keller6	3361	59	100-0-0	59	10.8103	47755	1081.03
MANN_a9	45	16*	100-0-0	16	0.0161	835681	1.61
MANN_a27	378	126*	100-0-0	126	0.0707	715188	7.07
MANN_a45	1035	345*	4-96-0	344.04	112.8498	436381	22450.52
MANN_a81	3321	1100	0-0-100	1098	27.5524	332219	2755.24
hamming6-2	64	32*	100-0-0	32	< 0.0001	581395	0.01
hamming6-4	64	4*	100-0-0	4	< 0.0001	245700	0.01
hamming8-2	256	128*	100-0-0	128	0.0005	236966	0.05
hamming8-4	256	16*	100-0-0	16	< 0.0001	177935	0.01
hamming10-2	1024	512*	100-0-0	512	0.3116	71123	31.16
hamming10-4	1024	40	100-0-0	40	0.9167	130548	91.67
gen200_p0.9_44	200	44*	100-0-0	44	0.0074	375939	0.74
gen200_p0.9_55	200	55*	100-0-0	55	0.0006	531914	0.06
gen400_p0.9_55	400	55	100-0-0	55	0.5476	211914	54.76
gen400_p0.9_65	400	65	100-0-0	65	0.0123	355871	1.23
gen400_p0.9_75	400	75	100-0-0	75	0.0415	200512	4.15
c-fat200-1	200	12*	100-0-0	12	0.0014	108675	0.14
c-fat200-2	200	24*	100-0-0	24	0.1742	91407	17.42
c-fat200-5	200	58*	100-0-0	58	0.1102	87719	11.02
c-fat500-1	500	14*	100-0-0	14	0.1354	47755	13.54
c-fat500-2	500	26*	100-0-0	26	0.2253	44150	22.53
c-fat500-5	500	64*	100-0-0	64	0.1009	39510	10.09

Table 1 – continued from previous page

Instance	Node	ω	Quality	AvgSize	AvgTime	Iter/sec	TotalTime
c-fat500-10	500	126*	100-0-0	126	2.6587	29629	265.87
johnson8-2-4	28	4*	100-0-0	4	< 0.0001	375939	0.01
johnson8-4-4	70	14*	100-0-0	14	< 0.0001	425531	0.01
johnson16-2-4	120	8*	100-0-0	8	< 0.0001	96993	0.01
johnson32-2-4	496	16*	100-0-0	16	< 0.0001	22857	0.01
p_hat300-1	300	8*	100-0-0	8	0.0008	130548	0.08
p_hat300-2	300	25*	100-0-0	25	0.0007	220750	0.07
p_hat300-3	300	36*	100-0-0	36	0.0016	255754	0.16
p_hat500-1	500	9*	100-0-0	9	0.0011	84175	0.11
p_hat500-2	500	36*	100-0-0	36	0.0008	165213	0.08
p_hat500-3	500	50	100-0-0	50	0.0053	284419	0.53
p_hat700-1	700	11*	100-0-0	11	0.0098	60518	0.98
p_hat700-2	700	44*	100-0-0	44	0.0012	155470	0.12
p_hat700-3	700	62	100-0-0	62	0.0053	233798	0.53
p_hat1000-1	1000	10	100-0-0	10	0.0008	45202	0.08
p_hat1000-2	1000	46	100-0-0	46	0.0009	105470	0.09
p_hat1000-3	1000	68	100-0-0	68	0.0813	200348	8.13
p_hat1500-1	1500	12*	100-0-0	12	2.1815	31628	218.15
p_hat1500-2	1500	65	100-0-0	65	0.3284	80123	32.84
p_hat1500-3	1500	94	100-0-0	94	0.3153	139885	31.53
san200_0.7_1	200	30*	100-0-0	30	0.2074	100102	20.74
san200_0.7_2	200	18*	100-0-0	18	0.2420	88909	24.20
san200_0.9_1	200	70*	100-0-0	70	0.1676	170024	16.76
san200_0.9_2	200	60*	100-0-0	60	0.1322	300293	13.22
san200_0.9_3	200	44*	100-0-0	44	0.0757	300263	7.57
san400_0.5_1	400	13*	100-0-0	13	11.4577	33336	1145.77
san400_0.7_1	400	40*	100-0-0	40	8.7633	40032	876.33
san400_0.7_2	400	30*	100-0-0	30	29.9791	42873	2997.91
san400_0.7_3	400	22*	100-0-0	22	56.2885	45024	5628.85
san400_0.9_1	400	100*	100-0-0	100	1.8674	42888	186.74
san1000	1000	15*	100-0-0	15	315.1698	37273	31516.98
sanr200-0.7	200	18*	100-0-0	18	0.0009	290697	0.09
sanr200-0.9	200	42*	100-0-0	42	0.0047	336700	0.47
sanr400-0.5	400	13*	100-0-0	13	0.0137	130548	1.37
sanr400-0.7	400	21	100-0-0	21	0.0048	182815	0.48

Table 1 gives the computational statistics using the same information as that employed in the literature on the maximum clique problem such as Pullan (2006); Pullan and Hoos (2006); Katayama et al. (2005); Battiti and Protasi (2001).

For each instance, we show in column 4 the solution quality by a triple $a - b - c$, where a is the number of runs (out of the 100 runs) in which a clique size of ω (ω is the maximum known clique size reported in the literature) is found, b is the number of runs in which the algorithm fails to find a clique size of ω, but attains a clique size of $\omega - 1$, c is the number of runs where only cliques of size $\omega - 2$ or worse are found. The next three columns provide other information: the averaged clique size over 100 runs, averaged CPU time in seconds over the successful runs and the average iterations per second. The last column indicates the *total run time of the 100 runs* of AMTS to solve an instance. As shown below, for most of the tested instances, one single run is sufficient to attain the largest clique size known in the literature.

Table 1 discloses that AMTS can find cliques of the largest known size for 79 out of the 80 benchmarks. The only instance for which AMTS fails to find the best known solution ($\omega = 1100$) is MANN_a81. For this instance, AMTS obtains consistently cliques of size 1098 in 100 of all the 100 runs. (The average time provided in Table 1 for the instance MANN_a81 is the average time to find cliques size of 1098.)

Of the 79 instances for which AMTS attains the best known solutions, in 76 cases it finds such a solution with a success rate of 100%. Consequently, one single run would suffice for AMTS to find a clique of the largest known size. For only three instances

Table 2 The performance of AMTS on the C2000.9 instance.

clique size(k)	AvgTime	AvgIter	success rate
80	450.09	38797622	1
79	338.39	29169205	93
78	33.52	2890191	100

(brock800_1, C2000.9, MANN_a45), not every run of AMTS can find a clique of the largest known size. Still each run can attain either a best known clique or cliques of sizes very close to the largest known size ω.

Indeed, for brock800_1 whose best known clique size is equal to 23, AMTS reaches with a very high probability of 0.98 cliques of this size with a single run. For C2000.9 which has a largest known clique size $\omega = 80$, the success rate is only 1%, but AMTS obtains consistently cliques of size 79 in 93 of 100 runs, while the remaining 6 runs finds cliques of size 78 (see Table 2). To the best of our knowledge, cliques of size 80 for C2000.9 have only been reported recently in Grosso et al. (2008). Not only AMTS attains this result, but also it can easily attain cliques of size 79 in reasonable time as shown in Table 2. Very similar comments can be made for MANN_a45.

If we check the computing times in Table 1, we observe that for 58 out of the 80 DIMACS instances (i.e., more than 72% cases), the average CPU time for attaining the best known solution is within 1 CPU second. A CPU time of 10 seconds to 7 minutes are required on average for the 22 remaining instances.

In sum, in 95% cases where a 100% success rate is reached, one single run of AMTS suffices to attain the largest clique size reported in the literature with a running time ranging from less than 1 second to several minutes. For the remaining 5% cases, each single run of AMTS is able to find legal k-cliques with k equaling or very close to the best known size ω.

3.4 Comparative results

In this section, we attempt to compare AMTS with 5 representative state-of-the-art methods from the literature. The main comparison criterion is the quality of the solutions found in terms of the largest and average clique size. Due to the differences among the programming languages, data structures, compiler options and computers, computing times are provided only for indicative purposes.

First, we recall the hardware and basic experimental conditions used by these reference methods.

– DLS-MC (Stochastic local search (Pullan and Hoos (2006))). The results of DLS-MC were based on a dedicated 2.2 GHz Pentium IV machine with 512KB L2 cache and 512MB RAM. For each instance, DLS-MC was run 100 times, each run being allowed 10^8 iterations like in our case.
– KLS (k-opt variable depth search algorithm (Katayama et al. (2005))). The results of of KLS were based on a Sun Blade 1000 Workstation (UltraSPARC-III 900 MHz, 2 GB memory). For each instance, KLS was run 100 trials. For each trial, KLS was repeatedly executed n times, where n was the number of nodes of a given graph.

- HSSGA (Heuristic based steady-state genetic algorithm (Singh and Gupta (2008))). HSSGA was run on a Pentium-III 1GHz Linux based system with 384 MB RAM. HSSGA was run 10 times on each graph instance. For each run, HSSGA was run until either the optimum solution value was found or a maximum of 20 000 generations was reached.
- RLS (Reactive local search (Battiti and Protasi (2001); Battiti and Mascia (2010))). RLS was run on a Pentium-II (450MHz CPU, 384MB RAM) machine. For each instance, 100 runs were performed, for each run, the number of iterations was fixed to $20000 \times n$.
- QUALEX-MS (Quick Almost Exact Motzkin-Straus-based search (Busygin (2006))). QUALEX-MS was run on a Pentium IV 1.4GHz computer under Red Hat Linux.

In Table 3, we first compare our AMTS method with DLS-MC which is the current best maximum clique algorithm. The comparison focuses on solution quality, i.e., the largest clique size found (averaged size is given in parenthesis if it is different from the largest one). As explained above, computing times are provided only as complementary information. Notice moreover that the results of DLS-MC were obtained after fine-tuning its parameter (Pullan and Hoos (2006)) on an instance-by-instance basis.

Table 3 shows that AMTS compares favorably with DLS-MC in terms of the best clique size. Indeed, AMTS can find the largest clique sizes for all the 80 instances except one case (MANN_a81) while DLS-MC can find the best known solutions for all the instances except three cases (MANN_a81, MANN_a45 and C2000.9). The difference between AMTS and DLS-MC can also be observed in terms of the average clique size obtained by the two algorithms; AMTS finds larger average clique size on three large and hard instances (C2000.9, MANN_a45 and MANN_a81) while the result of DLS-MC is better for one instance (brook800_1).

In terms of solution speed, DLS-MC shows better performance than AMTS on a number of instances, in particular some structured graphs. Indeed, for these instances (e.g., brock and san graphs), both algorithms can (rather easily) attain the best known solutions, but DLS-MC needs much less computing time.

In Table 4, we report the best and the average clique size obtained by AMTS in comparison with the other four algorithms (KLS, HSSGA, RLS and QUALEX-MS) on 37 DIMACS benchmark instances which are used by these reference algorithms. Table 5 summarizes the comparative results in terms of the number of instances on which these algorithms performs better or worse than AMTS.

Tables 4 and 5 show that AMTS finds larger cliques than KLS for 9 graphs, while the reverse is true only for one graph. Moreover, the average clique size found by AMTS is better than that of KLS on 14 instances whereas KLS outperforms AMTS on one instance. Regarding the other three algorithms (HSSGA, RLS and QUALEX-MS), AMTS can find an equal or better solution than these reference algorithms on each of the 37 benchmark instances.

4 Analysis of critical components of AMTS

4.1 Influence of restart

Recall that for each run of the algorithm, ATMS restarts from a new solution if the current solution is not improved for L consecutive iterations. So a small (large) value of

Table 3 Comparative results between AMTS and the top-performing maximum clique method DLS-MC. Results of DLS-MC are taken from Pullan and Hoos (2006). The results of both algorithms are based on 100 runs with a maximum of 10^8 iterations per run and per instance. For DLS-MC, average CPU times less than or equal to 0.0001 seconds are shown as ϵ. The focus is on solution quality. Computing times are provided only for indicative purposes.

	AMTS		DLS-MC			AMTS		DLS-MC	
Instance	Clique size	CPU(s)	Clique size	CPU(s)	Instance	Clique size	CPU(s)	Clique size	CPU(s)
brock200_1	21	0.0136	21	0.0182	johnson32_2_4	16	$< \epsilon$	16	$< \epsilon$
brock200_2	12	0.3625	12	0.0242	johnson8_2_4	4	$< \epsilon$	4	$< \epsilon$
brock200_3	15	0.0105	15	0.0367	johnson8_4_4	14	$< \epsilon$	14	$< \epsilon$
brock200_4	17	1.7582	17	0.0468	keller4	11	$< \epsilon$	11	$< \epsilon$
brock400_1	27	37.774	27	2.2299	keller5	27	0.0565	27	0.0201
brock400_2	29	1.1818	29	0.4774	keller6	59	10.810	59	170.483
brock400_3	31	1.7909	31	0.1758	MANN_a9	16	0.0161	16	$< \epsilon$
brock400_4	33	0.5956	33	0.0673	MANN_a27	126	0.0707	126	0.0476
brock800_1	23(22.96)	234.628	23	56.497	MANN_a45	345(344.04)	112.850	344	51.960
brock800_2	24	33.144	24	15.734	MANN_a81	1098	27.552	1098(1097.96)	264.009
brock800_3	25	52.398	25	21.920	p_hat300_1	8	0.0008	8	0.0007
brock800_4	26	15.234	26	8.8807	p_hat300_2	25	0.0007	25	0.0002
C125.9	34	0.0018	34	$< \epsilon$	p_hat300_3	36	0.0016	36	0.0007
C250.9	44	0.0058	44	0.0009	p_hat500_1	9	0.0011	9	0.0010
C500.9	57	0.1263	57	0.1272	p_hat500_2	36	0.0008	36	0.0005
C1000.9	68	1.1471	68	4.440	p_hat500_3	50	0.0053	50	0.0023
C2000.5	16	0.6611	16	0.9697	p_hat700_1	11	0.0098	11	0.0194
C2000.9	80(78.95)	450.100	78(77.93)	193.224	p_hat700_2	44	0.0012	44	0.0010
C4000.5	18	126.632	18	181.234	p_hat700_3	62	0.0053	62	0.0015
DSJC500.5	13	0.0071	13	0.0138	p_hat1000_1	10	0.0008	10	0.0034
DSJC1000.5	15	0.3113	15	0.7990	p_hat1000_2	46	0.0009	46	0.0024
c-fat200-1	12	0.0014	12	0.0002	p_hat1000_3	68	0.0813	68	0.0062
c-fat200-2	24	0.1742	24	0.0010	p_hat1500_1	12	2.1815	12	2.7064
c-fat200-5	58	0.1102	58	0.0002	p_hat1500_2	65	0.3284	65	0.0061
c-fat500-1	14	0.1354	14	0.0004	p_hat1500_3	94	0.3153	94	0.0103
c-fat500-2	26	0.2253	26	0.0004	san200_0.7_1	30	0.2074	30	0.0029
c-fat500-5	64	0.1009	64	0.0020	san200_0.7_2	18	0.2420	18	0.0684
c-fat500-10	126	2.6587	126	0.0015	san200_0.9_1	70	0.1676	70	0.0003
gen200-P0.9-44	44	0.0074	44	0.0010	san200_0.9_2	60	0.1322	60	0.0002
gen200-P0.9-55	55	0.0006	55	0.0003	san200_0.9_3	44	0.0757	44	0.0015
gen400-P0.9-55	55	0.5476	55	0.0268	san400_0.5_1	13	11.458	13	0.1641
gen400-P0.9-65	65	0.0123	65	0.0010	san400_0.7_1	40	8.7366	40	0.1088
gen400-P0.9-75	75	0.0415	75	0.0005	san400_0.7_2	30	29.979	30	0.2111
hamming6-2	32	$< \epsilon$	32	$< \epsilon$	san400_0.7_3	22	56.289	22	0.4249
hamming6-4	4	$< \epsilon$	4	$< \epsilon$	san400_0.9_1	100	1.8674	100	0.0029
hamming8-2	128	0.0005	128	0.0003	san1000	15	315.170	15	8.3636
hamming8-4	16	$< \epsilon$	16	$< \epsilon$	sanr200_0.7	18	0.0009	18	0.0020
hamming10-2	512	0.3116	512	0.0008	sanr200_0.9	42	0.0047	42	0.0127
hamming10-4	40	0.9167	40	0.0089	sanr400_0.5	13	0.0137	13	0.0393
johnson16-2-4	8	$< \epsilon$	8	$< \epsilon$	sanr400_0.7	21	0.0048	21	0.0230

L leads to more (less) frequent restart. To analyze the influence of the restart strategy on the performance of the AMTS algorithm, we focus on the effect of L and study the running profile of the evaluation function f (Formula (2), Section 2.2.1) by varying the value of L.

Experiments in this study are performed on a *structured* instance (brock800_2) and a *random* instance (C2000.9). To solve these instances, we consider 3 different values L

Table 4 Comparative results of AMTS with four other leading clique algorithms (KLS (Katayama et al. (2005)), HSSGA (Singh and Gupta (2008)), RLS (Battiti and Protasi (2001)) and QUALEX-MS (Busygin (2006)) on 37 DIMACS benchmark instances. The results of these methods are taken from the references. Graphs that are not shared by all these algorithms are not shown. The focus is on solution quality. Computing times are provided only for indicative purposes.

Instance	Node	Best	Max-Clique Algorithm									
			AMTS		KLS		HSSGA		RLS		QUALEX-MS	
			size	time	size	time	size	time	size	time	size	time
brock200_2	200	12*	12	0.3625	11	0.0035	12	0.29	12	9.605	12	< 1
brock200_4	200	17*	17	1.7582	16	0.0066	17(16.7)	1.14	17	19.491	17	< 1
brock400_2	400	29*	29	1.1818	25(24.84)	0.1334	29(25.1)	2.35	29(26.063)	42.091	29	3
brock400_4	400	33*	33	0.5956	25	0.0174	33(27.0)	2.76	33(32.423)	108.638	33	2
brock800_2	800	24*	24	33.144	21(20.86)	0.4993	21(20.7)	10.72	21	4.739	24	18
brock800_4	800	26*	26	15.234	21(20.67)	1.2160	21(20.1)	3.04	21	6.696	26	18
C125.9	125	34*	34	0.0018	34	0.0011	34	0.06	34	0.004	34	< 1
C250.9	250	44*	44	0.0058	44	0.0278	44(43.8)	0.34	44	0.029	44	1
C500.9	500	57	57	0.1263	57(56.15)	0.2699	56(54.2)	4.17	57	3.124	55	4
C1000.9	1000	68	68	1.1471	68(66.38)	2.0049	66(64.1)	14.27	68	41.660	64	27
C2000.5	2000	16	16	0.6611	16	2.8971	16(15.4)	27.52	16	9.976	16	278
C2000.9	2000	80	80(78.95)	450.10	77(74.90)	14.715	74(71.0)	117.66	78(77.575)	823.358	72	215
C4000.5	4000	18	18	126.63	18(17.02)	23.802	17(16.8)	158.42	18	2183.089	17	2345
DSJC500.5	500	13*	13	0.0071	13	0.0256	13	0.71	13	0.194	13	5
DSJC1000.5	1000	15*	15	0.3113	15(14.93)	0.6711	15(14.7)	7.38	15	6.453	14	36
keller4	171	11*	11	0.0001	11	0.0003	11	0.01	11	0.002	11	1
keller5	776	27	27	0.0565	27	0.0399	27(26.9)	4.04	27	0.171	26	16
keller6	3361	59	59	10.810	57(55.59)	52.364	57(54.2)	314.65	59	189.814	53	1291
MANN_a27	378	126*	126	0.0707	126	0.0178	126(125.5)	3.17	126	3.116	125	1
MANN_a45	1035	345*	345(344.04)	112.85	345(343.88)	6.2014	343(342.6)	65.25	345(343.602)	398.770	342	17
MANN_a81	3321	1100	1098	27.552	1100(1098.07)	39.484	1095(1094.2)	3996.65	1098	2830.820	1096	477
hamming8-4	256	16*	16	0.0001	16	0.0004	16	0.01	16	0.003	16	1
hamming10-4	1024	40	40	0.9167	40	0.2209	40(39.0)	10.21	40	0.078	40	45
gen200_p0.9_44	200	44*	44	0.0074	44	0.0317	44(43.1)	1.07	44	0.037	42	< 1
gen200_p0.9_55	200	55*	55	0.0006	55	0.0065	55	0.29	55	0.016	55	1
gen400_p0.9_55	400	55	55	0.5476	53(52.21)	0.2089	53(51.4)	1.83	55	1.204	51	2
gen400_p0.9_65	400	65	65	0.0123	65	0.0647	65(63.8)	1.71	65	0.050	65	2
gen400_p0.9_75	400	75	75	0.0415	75	0.0425	75	1.93	75	0.051	75	2
p_hat300-1	300	8*	8	0.0008	8	0.0021	8	0.02	8	0.018	8	1
p_hat300-2	300	25*	25	0.0007	25	0.0012	25	0.02	25	0.006	25	1
p_hat300-3	300	36*	36	0.0016	36	0.0118	36(35.9)	0.18	36	0.021	35	1
p_hat700-1	700	11*	11	0.0098	11	0.1245	11	1.02	11	0.186	11	10
p_hat700-2	700	44*	44	0.0012	44	0.0077	44	0.19	44	0.028	44	12
p_hat700-3	700	62	62	0.0053	62	0.0158	62(61.7)	2.01	62	0.035	62	11
p_hat1500-1	1500	12*	12	2.1815	12	2.6054	12(11.5)	14.62	12	30.274	12	95
p_hat1500-2	1500	65	65	0.3284	65	0.0625	65(64.9)	2.03	65	0.158	64	111
p_hat1500-3	1500	94	94	0.3153	94	0.4286	94(93.1)	2.91	94	0.192	91	108

Table 5 Comparison result of AMTS with KLS (Katayama et al. (2005)), HSSGA (Singh and Gupta (2008)), RLS (Battiti and Protasi (2001)) and QUALEX-MS (Busygin (2006) in terms of number of instances on which AMTS found better (or worse) results out of the 37 DIMACS benchmark instances. The symbol '-' used for QUALEX-MS indicates that the average clique size is not available.

	Best clique size		Average clique size	
	Better than AMTS	Worse than AMTS	Better than AMTS	Worse than AMTS
KLS	1	9	1	14
HSSGA	0	10	0	26
RLS	0	3	0	6
QUALEX-MS	0	14	-	-

= 100, 1000 and 10000. For each of these values, we perform 100 runs of AMTS, each run being given a maximum of $Iter_{max} = 10^7$ iterations.

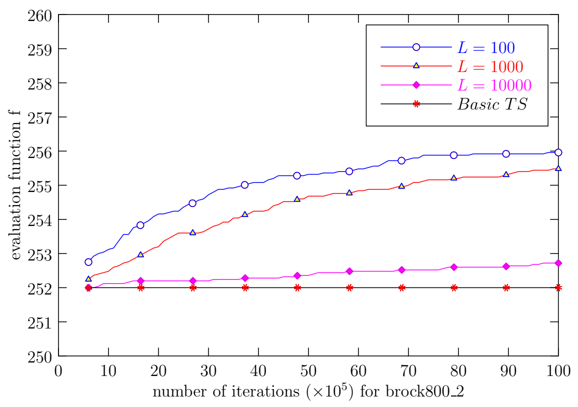

Fig. 1 Running profile of AMTS with $L = 100$, 1000 and 10000 as well as AMTS without restart (basic TS) on brock800_2.

The running profile is defined by the function $i \longmapsto f_*(i)$ where i is the number of iterations (counter $Iter$) and $f_*(i)$ is the best evaluation function value known at iteration i, averaged over 100 runs. More precisely, let j denote the jth run of AMTS ($j = 1...100$), $f_i^j(S^*)$ the value of the evaluation function f (defined by Formula (2) in Section 2.2.1) of the best solution S^* known at iteration i of AMTS jth run. For each plotted iteration i in the running profile, $f_*(i)$ is equal to $\sum_{j=1}^{100} f_i^j(S^*)/100$. Such a profile gives a natural way to observe the evolution of the best values of the objective function during a search (Galinier and Hao (1999)).

Figure 1 shows the running profiles of AMTS on the graph brock800_2 with $k = 24$. The figure shows also the running profile of AMTS without restart, i.e., with $L = Iter_{max}$ (we call this version basic TS). From Figure 1, we observe that AMTS with $L = 100$ dominates AMTS with $L = 1000$ and $L = 10000$. Since smaller L implies more restarts, this experiment suggests a frequent restart is quite useful for the instance brock800_2 (in fact for other special structured instances). One also notices that AMTS without restart performs the worst.

Figure 2 shows the running profiles of AMTS on C2000.9 with $k = 79$. It is interesting to observe that for this graph, AMTS performs better with large values $L = 1000$ or $L = 10000$ than with $L = 100$. AMTS without restart performs here quite well. This suggests that for C2000.9 (in fact for many random instances) a long search with the basic TS^0 engine is more effective than a search with frequent restarts.

The above observations are confirmed by the results reported in Table 6. In this table, we show the number of runs (out of the 100 runs) where a clique size of k is found successfully by AMTS with these L values and ATMS without restart. For brock800_2, search with frequent restarts makes ATMS more effective and robust whereas the reverse is true for C2000.9.

More generally, various experiments suggest that for some structured graphs, relatively smaller L values are preferable whereas for random graphs, it is advantageous to use relatively larger L values. This experiment also explains the choice of the L values

Table 6 Success rate of ATMS with different values of $L \in \{100, 1000, 10000\}$ and AMTS without restart (basic TS) for brock800_2 ($k = 24$) and C2000.9 ($k = 79$).

Graph	L=100	L=1000	L=10000	Basic TS
brock800_2	99	87	18	0
C2000.9	0	6	19	17

used in Section 3.3. In sum, compared to the $Iter_{max}$ parameter, L is more sensitive to the structure of the graph and should be tuned with more care.

4.2 The tabu list

As explained in Section 2.2.5, each time a $swap(u, v)$ move is performed, both the dropped vertex u and the added vertex v are marked tabu for respectively T_u and T_v iterations. We experiment here two additional tabu strategies which are summarized together with the previous one as follows.

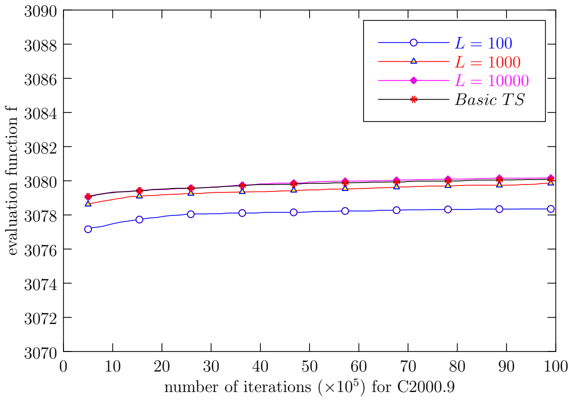

Fig. 2 Running profile of AMTS with L = 100, 1000 and 10000 as well as AMTS without restart (basic TS) on C2000.9.

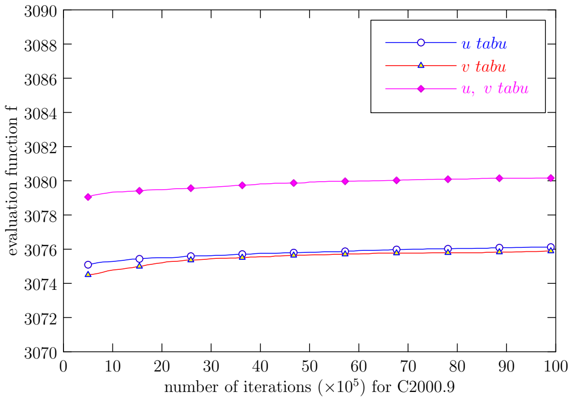

Fig. 3 The basic TS with three tabu strategies.

– *Strategy* 1: Only preventing the dropped vertex u from being put back into S in the next T_u iterations.
– *Strategy* 2: Only preventing the added vertex v from being removed from S in the next T_v iterations.
– *Strategy* 3: Preventing u from being put back into S in the next T_u iterations while preventing v from being removed from S in the next T_v iterations. This strategy is used in this paper.

We test these three strategies on C2000.9 with $k = 79$. Figure 3 shows the running profiles. From the figure, we can observe that strategy 3, which is used by our proposed AMTS algorithm, largely dominates strategy 1 and strategy 2 throughout the search.

5 Conclusions

Our proposed Adaptive Multistart Tabu Search algorithm represents a new approach for approximating the maximum clique problem. AMTS seeks a clique of fixed size k by effectively exploring subsets of vertices of size k. For this purpose, AMTS combines a TS procedure with a guided restart strategy. The TS engine is based a constrained neighborhood and an adaptive technique for tuning the double tabu tenures. To enable a more intensive exploration of the search space, AMTS uses an informed multistart strategy which relies on a long term memory (move frequencies) to regenerate new initial starting solutions.

AMTS shows an excellent performance on the complete set of 80 standard DIMACS benchmark instances. AMTS finds the current best known solutions for all the instances except one case (MANN_a81 for which cliques of size 1098 are found easily). The competitiveness of AMTS is further confirmed when it is compared with five state-of-the-art maximum clique procedures.

Most of the current top-performing algorithms for the maximum clique problem are based on an expansion and plateau search model. The proposed method constitutes an interesting alternative approach that probably merits more attention and research efforts.

Finally, the AMTS algorithm has been applied very recently with success to solve two combinatorial problems: graph coloring and graph sum coloring (Wu and Hao (2012a,b)). The algorithm with its source code that we will make publically available will certainly find more applications.

Acknowledgment

We are grateful to the referees for their comments and questions which helped us to improve the paper. This work was partially supported by the Region of "Pays de la Loire" (France) within the Radapop and LigeRO Projects.

References

Arora S, Lund C, Motwani R, Sudan M, Szegedy M (1992) Proof verification and the hardness of approximation problems. In Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science. Pittsburgh, pp 14-23

Balas E, Yu CS (1986) Finding a maximum clique in an arbitrary graph. SIAM Journal on Computing 15(4):1054-1068

Barbosa V, Campos L (2004) A novel evolutionary formulation of the maximum independent set problem. Journal of Combinatorial Optimization 8(4):419-437

Battiti R, Mascia F (2010) Reactive and dynamic local search for max-clique: Engineering effective building blocks. Computers and Operations Research 37(3):534–542

Battiti R, Protasi M (2001) Reactive local search for the maximum clique problem. Algorithmica 29(4):610-637

Bui T, Eppley P (1995) A hybrid genetic algorithm for the maximum clique problem. Proceedings of the 6th International Conference on Genetic Algorithms pp 478–484

Busygin S, Butenko S, Pardalos PM (2002) A Heuristic for the Maximum Independent Set problem based on optimization of a quadratic over a sphere. Journal of Combinatorial Optimization 6(3): 287–297

Busygin S (2006) A new trust region technique for the maximum weight clique problem. Discrete Applied Mathematics 154(1):2080-2096

Carraghan R, Pardalos PM (1990) An exact algorithm for the maximum clique problem. Operations Research Letters 9:375-382

Fleurent C, Ferland J (1996) Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. In: Johnson D, Trick M (eds) Proceedings of the 2nd DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, vol 26, pp 619-652

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

Galinier p, Hao JK (1999) Hybrid evolutionary algorithms for graph coloring. Journal of Combinatorial Optimization 3(4):379–397

Garey MR, Johnson DS (1979) Computers and Intractability. W. H. Freeman and Company

Gendreau M, Soriano P, Salvail L (1993) Solving the maximum clique problem using a tabu search approach. Annals of Operations Research 41:385–403

Glover F, Laguna M (1997) Tabu Search. Kluwer Academic Publishers

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

Hästad J (1999) Clique is hard to approximate within $n^{1-\varepsilon}$. Acta Mathematica 182:105–142

Johnson DS, Trick MA (1996) Second DIMACS implementation challenge: cliques, coloring and satisfiability, American Mathematical Society. volume 26 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science

Karp, RM (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.

Katayama K, Hamamoto A, Narihisa H (2005) An effective local search for the maximum clique problem. Information Processing Letters 95(5):503-511

Marchiori E (1998) A simple heuristic based genetic algorithm for the maximum clique problem. In Proceedings of of ACM Symposium on Applied Computing pp 366-373

Marchiori E (2002) Genetic, iterated and multistart local search for the maximum clique problem. In Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2279:112-121

Östergärd PJR (2002) A fast algorithm for the maximum clique problem. Discrete Applied Mathematics 120:195-205

Pardalos PM, Xue J. (2002) The maximum clique problem. Journal of Global Optimization 4:301-328

Pullan W (2006) Phased local search for the maximum clique problem. Journal of Combinatorial Optimization 12(3):303-323

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

Rebennack S, Oswald M, Theis DO, Seitz H, Reinelt G, Pardalos P.M (2011) A Branch and Cut solver for the maximum stable set problem. Journal of Combinatorial Optimization 21(4): 434–457

Singh A, Gupta AK (2008) A hybrid heuristic for the maximum clique problem. Journal of Heuristics 12:5-22

Tomita E, Seki T (2003) An efficient branch-and-bound algorithm for finding a maximum clique. In Discrete Mathematics and Theoretical Computer Science 2731:278-289

Zhang QF, Sun JY, Tsang E (2005) Evolutionary algorithm with the guided mutation for the maximum clique problem. IEEE Transactions on Evolutionary Computation 9(2):192-200

Wu Q., Hao JK (2012a) Coloring large graphs based on independent set extraction. Computers and Operations Research 39(2): 283–290

Wu Q, Hao JK (2012b) An effective heuristic algorithm for sum coloring of graphs. Computers and Operations Research 39(7): 1593–1600.