J Glob Optim (2014) 59:1–21
DOI 10.1007/s10898-013-0075-9

Speeding up branch and bound algorithms for solving the maximum clique problem

Evgeny Maslov · Mikhail Batsyn · Panos M. Pardalos

Received: 14 March 2013 / Accepted: 4 May 2013 / Published online: 16 May 2013

Abstract In this paper we consider two branch and bound algorithms for the maximum clique problem which demonstrate the best performance on DIMACS instances among the existing methods. These algorithms are MCS algorithm by Tomita et al. (2010) and MAXSAT algorithm by Li and Quan (2010a, b). We suggest a general approach which allows us to speed up considerably these branch and bound algorithms on hard instances. The idea is to apply a powerful heuristic for obtaining an initial solution of high quality. This solution is then used to prune branches in the main branch and bound algorithm. For this purpose we apply ILS heuristic by Andrade et al. (J Heuristics 18(4):525–547, 2012). The best results are obtained for *p_hat1000-3* instance and *gen* instances with up to 11,000 times speedup.

Keywords Maximum clique problem · Branch and bound algorithm · Heuristic solution · Graph colouring

Mathematics Subject Classification (2000) 05C69 · 05C85 · 90C27 · 90C59 · 90-08

1 Introduction

The maximum clique problem refers to the problem of finding a clique (a complete subgraph) with the largest number of vertices in a given graph. It has many applications because

E. Maslov · M. Batsyn (✉) · P. M. Pardalos
Laboratory of Algorithms and Technologies for Networks Analysis,
National Research University Higher School of Economics,
136 Rodionova Street, Niznhy Novgorod, Russia
e-mail: mbatsyn@hse.ru

E. Maslov
e-mail: lyriccoder@gmail.com

P. M. Pardalos
Center of Applied Optimization, University of Florida,
303 Weil Hall, Gainesville, FL 32611, USA
e-mail: pardalos@ufl.edu

many practical problems can be formulated in terms of the maximum clique problem [5]. Biochemistry and genomic problems represented by a clique-detection model include integration of genome mapping data, nonoverlapping local alignments, matching and comparing molecular structures, and protein docking [8]. Another problem is to find a binary code as large as possible which can correct a certain number of errors for a given size of the binary words (vectors) [7,28]. Among these binary words there must be two words which differ in a certain number of positions so that a misspelled word can be detected and corrected [10]. A clique depicts a feasible set of vectors for a code. Error-correcting codes are used in cellular phones, high-speed modems, and CD players (when computing checksums). Finding large cohesive subgroups (cliques) in social networks is used in criminal network analysis. One more practical application is analyzing cliques in a stock market graph [4].

A well-known algorithm for enumerating all cliques in a graph is the algorithm of Bron and Kerbosch [6]. It finds all maximal cliques (cliques which cannot be further enlarged by adding any vertex) in an arbitrary graph. Since this method has only branching and no bound is used to reduce the number of branches, it takes enormous amount of time to find the maximum clique even in small dense graphs. The worst-case running time of the Bron–Kerbosch algorithm is $O(3^{\frac{n}{3}})$ [16].

One of the first branch and bound algorithms is the algorithm developed by Carraghan and Pardalos [9]. The main idea is to use bound strategy and prune branches in case when future expanding will not lead us to a clique with a size bigger than the largest clique found so far. Another idea of this algorithm is to sort vertices in a special order which reduces the size of the search tree. This order is also used by one the recent exact algorithms—MCS algorithm developed by Tomita et al. [31].

The algorithm proposed by Fahle [11] suggests a more efficient bounding strategy. The idea is to use the chromatic number as an upper bound on the size of the maximum clique. To be more precise, a graph of candidate vertices is coloured by means of a heuristic sequential colouring. And the number of colours is then used as an upper bound for the maximum clique size. The algorithm also applies domain filtering techniques based on two particular observations. The first observation is the following: if there is a vertex in the set of candidates which is connected to all vertices of the current clique, then this vertex will be included in the clique in all branches. Such vertex should be added to the current clique immediately without any branching. The second observation is that a vertex in the set of candidates must be excluded from consideration if its degree in the subgraph of candidates plus the size of the current clique is less than the size of the currently best found clique.

MCQ algorithm developed by Tomita and Seki [30] uses the idea of graph colouring not only as a bounding strategy but also as a branching strategy. Initially vertices are sorted in a non-increasing degree order. At each branching step a greedy sequential colouring is computed for the subgraph of candidates. The candidate vertex which is coloured in the biggest colour (colour with the biggest number) is considered first.

MCR algorithm by Tomita and Kameda [29] and MCS algorithm by Tomita et al. [31] are further improvements of MCQ. The only difference between MCQ and MCR is in the ordering of vertices performed at the beginning. In MCR vertices are sorted in practically the same order as suggested in [9]. In MCS algorithm a new routine is added which tries to recolour a vertex with the biggest colour into a smaller one.

Another recent algorithm developed by Li and Quan [22] and its improved version [21] apply an upper bound based on partial maximum satisfiability problem. This upper bound is tighter than the colouring-based upper bound. According to the published results for DIMACS graphs MCS and MAXSAT algorithms report the best performance among the existing exact methods known to the authors.

Since the maximum clique problem is NP-complete [18], there exists a number of heuristic approaches which can find a solution of high quality. The greedy heuristics either try to create a clique by gradually adding a vertex to the current clique or to find a clique by repeatedly removing a vertex from the current set which is not a clique [20]. Genetic algorithms [23,27] start with some initial randomly generated population and then perform the routines of reproduction, crossover and mutation. There are a lot of other heuristics including simulated annealing [17], neural networks [3,13], GRASP [12], tabu search [14] and others.

In practice the most successful heuristic algorithms are local search heuristics. On each step a local search algorithm finds a maximal clique, then tries to improve this solution by, for example, a $(j, k) - swap$, that is removing some j vertices from a clique and adding other k vertices to it. Pullan and Hoos [26] developed a very efficient heuristic called *dynamic local search* which consists of fast neighbourhood search and usage of penalties to promote diversification. Grosso et al. [15] improved the performance of this algorithm by introducing restart rules. Their GLP algorithm has found new cliques unknown before for very large instances. One of the most efficient heuristic algorithms is iterated local search (ILS) developed by Andrade et al. [1]. These authors suggested to use local search instead of an elaborate plateau search applied in GLP algorithm. Their incremental implementation of the local search procedure is faster than the standard one and runs in sublinear time. In our work we use the ILS algorithm to obtain an initial solution to the maximum clique problem by solving heuristically the maximum independent set problem for a complementary graph.

Though it is a well-known fact that any branch and bound algorithm benefits much when used together with a heuristic applied to obtain an initial solution, almost none of the existing branch and bound approaches applies this powerful technique. We have considered two recent branch and bound algorithms: MCS algorithm by Tomita et al. [31] and MAXSAT algorithm by Li and Quan [21]. We suggest using ILS heuristic [1] to obtain an initial high-quality solution which is then used to prune branches in the main branch and bound algorithm. The computational study shows that this improvement results in a considerable reduction of the search tree size and computational time.

This paper is organized as follows. In Sect. 2 we give the formulation of the maximum clique problem. In Sect. 3 an example for MCS and MAXSAT and our improved versions of these algorithms (ILS&MCS and ILS&MAXSAT) is provided. Computational results showing a comparison with the original MCS and MAXSAT algorithms are presented in Sect. 4.

2 Maximum clique problem

Let $G = (V, E)$ be an undirected graph, where $V = \{1, 2, \ldots, n\}$ is the set of vertices, $E \subseteq V \times V$ is a set of edges. The adjacency matrix of $G(V, E)$ is denoted as $A = (a_{ij})$, where $a_{i,j} = 1$ if $(i, j) \in E$ and $a_{i,j} = 0$ if $(i, j) \notin E$.

A complementary graph of $G(V, E)$ is the graph $\overline{G}(V, \overline{E})$, where $\overline{E} = \{(i, j) \mid (i, j) \in (V \times V)\backslash E\}$. Graph $G(V, E)$ is *complete* if all its vertices are pairwise adjacent, i.e. $\forall i, j \in V,\ (i, j) \in E$. A *clique* C is a subset of V such that all vertices in this subset are pairwise adjacent. A clique which cannot be enlarged by adding any vertex to it is called *a maximal clique*. A clique which has the maximum size (number of vertices) in a graph is called *a maximum clique*. The number of vertices of a maximum clique in graph $G(V, E)$ is denoted by $\omega(G)$. The maximum clique problem refers to the problem of finding the maximum clique

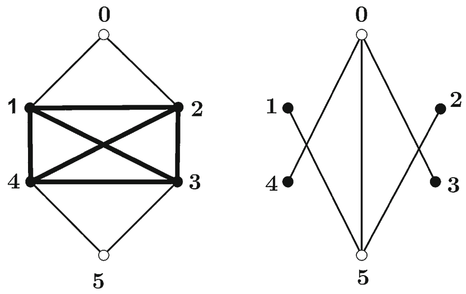

Fig. 1 Maximum clique and maximum independent set

in a given graph. The maximum clique problem has a number of mathematical programming formulations. One of them is the following:

$$\max \sum_{i=1}^{n} x_i \tag{1}$$

$$\text{s.t.} \;\; x_i + x_j \leq 1, \forall (i, j) \in \overline{E}, \tag{2}$$

$$x_i \in \{0, 1\}, i = 1, \ldots, n. \tag{3}$$

Here if $x_i = 1$ then vertex i is in the maximum clique C, otherwise $i \notin C$.

An independent set is a subset of V, which elements are pairwise non-adjacent. The maximum independent set problem consists in finding of the largest independent set in a graph.

The maximum clique and the maximum independent set problems are complementary: a clique in graph G is an independent set in the complementary graph $\overline{G}$ and vice versa (see Fig. 1).

A *colouring* of graph G is an assignment of colours to the graph vertices so that any two adjacent vertices have different colours. The smallest number of colours in which it is possible to colour a graph is called *a chromatic number*. Colouring can be used to obtain an upper bound for the maximum clique problem (Proposition 1).

Proposition 1 [2] *The maximum clique size of an arbitrary graph is not greater than its chromatic number.*

This proposition follows immediately from the fact that a clique of k vertices can be coloured only in k colours because its vertices are all pairwise adjacent. Note that the chromatic number of a graph can be arbitrarily greater than its clique number (maximum clique size). The following theorem of Mycielski demonstrates this fact.

Theorem 1 [25] *For any natural number n there exists a finite triangle-free graph which cannot be coloured in n colours.*

It is clear that the maximum clique size of any triangle-free graph cannot be more than two. But the chromatic number of such a graph may be arbitrarily large. Figure 2 shows triangle-free graphs with chromatic numbers 2, 3, and 4, and the maximum clique size 2.

Maximum clique problem can be also formulated in terms of maximum satisfiability problem (proposition 2).

Definition 1 ([22]) Let G be a graph partitioned into independent sets. Then the independent set based MaxSAT encoding of the maximum clique is defined as follows: (1) each vertex

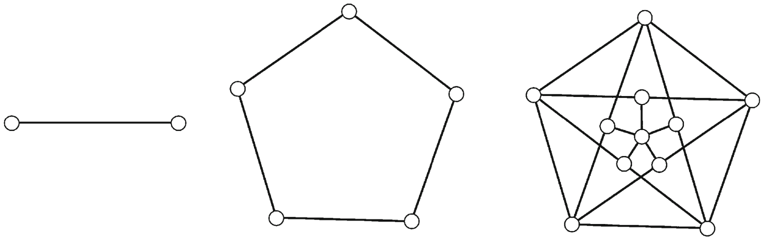

Fig. 2 Mycielski's graphs with chromatic number 2, 3, and 4 and maximum clique size 2

i in G is represented by a Boolean variable x_i, (2) a hard clause $\overline{x_i} \vee \overline{x_j}$ is added for each pair of non-connected vertices (i, j), and (3) a soft clause is added for each independent set which is a logical or of the variables representing the vertices in the independent set.

Proposition 2 [22] *Let ϕ be an independent set based MaxSAT encoding for a graph G. Then the set of variables evaluated to true in any optimal assignment of ϕ gives a maximum clique of G.*

For example the graph shown in Fig. 1 can be partitioned into 4 independent sets $\{0, 3\}, \{1, 5\}, \{2\}, \{4\}$ using the greedy colouring. Then MaxSAT encoding ϕ for it has variables $x_0, x_1, x_2, x_3, x_4, x_5$, soft clauses $x_0 \vee x_3, x_1 \vee x_5, x_2, x_4$, and hard clauses $\overline{x_0} \vee \overline{x_3}, \overline{x_0} \vee \overline{x_4}, \overline{x_0} \vee \overline{x_5}, \overline{x_1} \vee \overline{x_5}, \overline{x_2} \vee \overline{x_5}$ according to definition 1. The objective is to find such an assignment of values 0 and 1 to variables x_i, that the maximum number of the soft clauses and all the hard clauses are satisfied (equal to 1). It is not difficult to check that this encoding is equivalent to the mathematical programming formulation (1)–(3). For our example the formulation (1)–(3) is as follows:

$$\begin{aligned}
&\max(x_0 + x_1 + x_2 + x_3 + x_4 + x_5)\\
&\quad \text{s.t.}\ \ x_0 + x_3 \leq 1, x_0 + x_4 \leq 1, x_0 + x_5 \leq 1, x_1 + x_5 \leq 1, x_2 + x_5 \leq 1,\\
&\quad x_0, x_1, x_2, x_3, x_4, x_5 \in \{0, 1\}.
\end{aligned}$$

Since only one variable in every soft clause can be equal to 1 (only one vertex from every independent set can be in the maximum clique), then $\max(x_0 + x_1 + x_2 + x_3 + x_4 + x_5) = \max(x_0 \vee x_3 + x_1 \vee x_5 + x_2 + x_4)$.

3 Improved MCS and MAXSAT algorithms

Applying a fast heuristic algorithm before running a slow exact algorithm always reduces the number of search tree nodes and usually reduces the total running time. A heuristic finds a high-quality solution which is not very far from the exact optimum of the objective function. This solution is then used as a tight lower bound, and all the branches which have an upper bound not greater than this lower bound are pruned. To obtain a good lower bound we run ILS heuristic at the beginning of the algorithm. Though this approach is rather simple and obvious almost none of the existing exact algorithms apply it. We show that running of ILS heuristic before MCS and MAXSAT algorithms allow to reduce the total computational time considerably for hard DIMACS instances.

We demonstrate MCS and MAXSAT algorithms and our improved versions of these algorithms (ILS&MCS and ILS&MAXSAT) on the graph shown in Fig. 3.

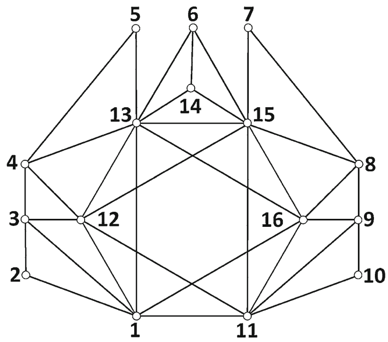

Fig. 3 A graph for algorithms demonstration

Before the main branch and bound procedure MCS algorithm makes a reordering of vertices in practically the same way as suggested by Carraghan and Pardalos [9]. Note that the same ordering was first suggested by Matula et al. [24] for an efficient heuristic colouring. The vertices are ordered: $\upsilon_1, \upsilon_2, \ldots, \upsilon_n$, so that for $k = 1, 2, \ldots, n$ vertex υ_k is a vertex with the minimal degree in graph $G \setminus \{\upsilon_{k+1}, \upsilon_{k+2}, \ldots, \upsilon_n\}$ (where graph $G \setminus \{\upsilon_{k+1}, \upsilon_{k+2}, \ldots, \upsilon_n\}$ is the graph G in which vertices $\upsilon_{k+1}, \ldots, \upsilon_n$ are removed together with their edges).

If several vertices have the same minimal degree then the algorithm chooses the vertex with the minimal sum of its neighbours (adjacent vertices) degrees. In our example vertices 2, 5, 7, 10 have minimal degree 2, for vertices 2, 10 the sum of neighbours degrees is equal to $6 + 4 = 10$ and for vertices 5, 7 it is equal to $4 + 8 = 12$. So we place vertex 2 to the last nth position in our sequence, then remove it from the graph, find that vertex 10 has the minimal degree and sum of neighbours degrees in the remaining graph, and place it to position $n - 1$ and so on.

For the last considered vertices $\upsilon_1, \upsilon_2, \ldots, \upsilon_r$ all having the same degree we do not compute the sum of neighbours degrees. This set of vertices is called R_{min}. In our example $R_{min} = \{1, 11, 12, 13, 15, 16\}$. The subgraph formed by R_{min} vertices is coloured by the greedy sequential colouring in the lexicographic sequence. For our example we get the following colours correspondingly: 1, 2, 3, 2, 1, 3. And then these vertices are sorted by their colours so that the first vertex has the minimal colour: $\{1, 15, 11, 13, 12, 16\}$.

As a result we obtain the following sequence of all vertices: 1, 15, 11, 13, 12, 16, 8, 9, 4, 3, 14, 6, 7, 5, 10, 2. Then according to MCS algorithm we renumber the vertices of the graph so that this sequence 1, 15, 11, 13, 12, 16, 8, 9, 4, 3, 14, 6, 7, 5, 10, 2 becomes $1, 2, \ldots, 16$. This is made in order to increase the efficiency of processor cache because the vertices are always accessed in this sequence in MCS algorithm. The graph after renumbering of vertices is shown in Fig. 4.

Before the main branch and bound procedure MCS algorithm colours all the vertices except R_{min} with a dummy colouring one by one so that every vertex is coloured in a new colour until the colour number is not equal to $\Delta(G) + 1$, where $\Delta(G)$ is the maximal degree in graph G. All the remaining vertices are coloured in the same colour $\Delta(G) + 1$. For our example $\Delta(G) + 1 = 9$, R_{min} is coloured in colours 1, 1, 2, 2, 3, 3, and so all the vertices $1, 2, \ldots, 16$ will have colours 1, 1, 2, 2, 3, 3, 4, 5, 6, 7, 8, 9, 9, 9, 9, 9. Such a dummy colouring is used to keep the current order of vertices and at the same time provide the increasing sequence of colours.

The idea of MCS branch and bound algorithm is the following. For every vertex we iterate over maximal cliques, containing this vertex, pruning the branches which cannot lead to a maximal clique larger than the largest clique found so far. We prune the branches for which

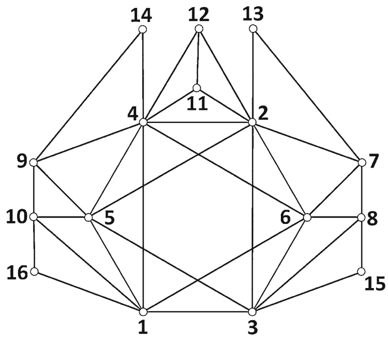

Fig. 4 The graph after renumbering of vertices

colouring gives an upper bound not greater than the size of the currently largest clique. We start searching for a maximal clique from one vertex, then find all its neighbours and add one of them to this vertex forming the current clique Q of two vertices. These neighbours are called candidates. Among the candidates we leave only those which are adjacent to the added vertex. Then again add one candidate to the current clique Q so that it contains three vertices now. And again leave only candidates adjacent to the added vertex. This process is repeated until there are no more candidates and thus a maximal clique is found. Then we return one level back and take the next candidate. We prune the current candidate υ if the size of the current clique Q plus the colour number c_υ of this candidate is not greater than the currently largest clique Q^*. Since MCS algorithm sorts candidates by their colour numbers all the next candidates have a smaller colour and we can immediately return one level back pruning all of them.

The main steps of the MCS branch and bound procedure for our example are provided in Table 1. The initial sequence of vertices is 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 and their colours are 1 1 2 2 3 3 4 5 6 7 8 9 9 9 9 9. Note that the candidates are always considered in reverse order starting from the last one with the biggest colour (vertex 16 on the first step).

The search tree for MCS algorithm is given in Fig. 5. The pruned branches are shown by dashed circles. The search tree has 14 nodes.

The improved algorithm—ILS&MCS runs ILS heuristic before starting MCS algorithm. Let us assume that for our example ILS has found the clique of 4 vertices. This solution with the size $|Q^*| = 4$ is then used to prune branches from the first step (see Table 2) The search tree for ILS&MCS algorithm is given in Fig. 6. The pruned branches are shown by dashed circles. The search tree now has only 9 nodes instead of 14.

Now let us show MAXSAT and ILS&MAXSAT algorithms on the same example. Following MaxCliqueDyn algorithm [19] MAXSAT algorithm first considers a vertex υ with minimum degree in the graph of candidates on the upper levels of the search tree and an arbitrary vertex on the lower levels. For our example we take the vertex with minimum degree on all the levels of the search tree. In every node of the search tree the algorithm has two branches: υ and $\backslash\upsilon$. In the left branch we add this vertex υ to the current clique Q and consider the graph of its neighbours G_υ. In the right branch we remove vertex υ and consider the remaining graph $G\backslash\upsilon$. Before branching in every node an upper bound for the current graph is calculated by solving a partial maximum satisfiability problem as it is described in [21].

The main steps of the MAXSAT branch and bound procedure for our example are provided in Table 3. The first vertex with minimum degree is vertex 2. Then in the remaining graph such vertex is vertex 5. An so on the sequence of vertices is 2 5 7 10 3 4 6 and for the

Table 1 MCS algorithm main steps

Clique Q	Candidates υ	Colours c_υ	Comments
	1 2 3 4 5 6	1 1 2 2 3 3	
	7 8 9 10 11	4 5 6 7 8	
	12 13 14 15 16	9 9 9 9 9	
16	1 10	1 2	Vertex 16 has neighbours 1 and 10
16 10	1	1	
16 10 1			Found a maximal clique, $Q^* = \{16, 10, 1\}$
16	1	1	This branch is pruned since $\|Q\| + c_\upsilon \leq \|Q^*\|$
15	3 8	1 2	Vertex 15 has neighbours 3 and 8
			Both branches are pruned since $1 + 2 \leq 3$
14	4 9	1 2	Vertex 14 has neighbours 4 and 9
			Both branches are pruned since $1 + 2 \leq 3$
13	2 7	1 2	Vertex 13 has neighbours 2 and 7
			Both branches are pruned since $1 + 2 \leq 3$
12	2 4 11	1 2 3	Vertex 12 has neighbours 2, 4, 11
12 11	2 4	1 2	
12 11 4	2	1	
12 11 4 2			Found a maximal clique, $Q^* = \{12, 11, 4, 2\}$
12 11	2	1	This branch is pruned since $2 + 1 \leq 4$
12	2 4	1 2	This branch is pruned since $1 + 2 \leq 4$
11	2 4	1 2	Vertex 11 has neighbours 2 and 4
			Both branches are pruned since $1 + 2 \leq 4$
10	1 5 9	1 2 1	Vertex 10 has neighbours 1, 5, 9
10	1 9 5	1 1 2	The candidates are ordered by colours
			All 3 branches are pruned since $1 + 2 \leq 4$
9	4 5	1 2	Vertex 9 has neighbours 4 and 5
			Both branches are pruned since $1 + 2 \leq 4$
8	3 6 7	1 2 1	Vertex 8 has neighbours 3, 6, 7
8	3 7 6	1 1 2	The candidates are ordered by colours
			All 3 branches are pruned since $1 + 2 \leq 4$
	1 2 3 4 5 6 7	1 1 2 2 3 3 4	All 7 branches are pruned since $0 + 4 \leq 4$

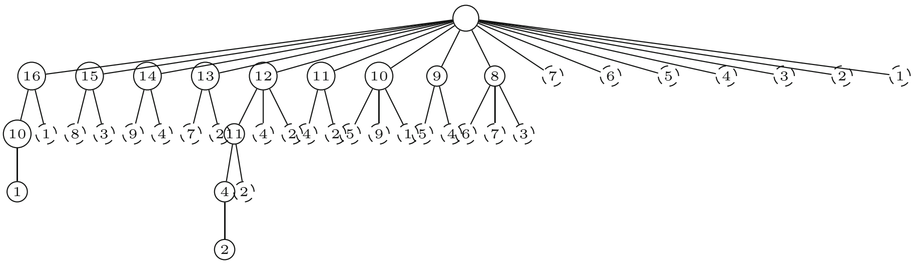

Fig. 5 Search tree for MCS algorithm

Table 2 ILS&MCS algorithm main steps

Clique Q	Candidates υ	Colours c_υ	Comments
	1 2 3 4 5 6	1 1 2 2 3 3	
	7 8 9 10 11	4 5 6 7 8	
	12 13 14 15 16	9 9 9 9 9	
16	1 10	1 2	Vertex 16 has neighbours 1 and 10
			Both branches are pruned since $1 + 2 \leq 4$
15	3 8	1 2	Vertex 15 has neighbours 3 and 8
			Both branches are pruned since $1 + 2 \leq 3$
14	4 9	1 2	Vertex 14 has neighbours 4 and 9
			Both branches are pruned since $1 + 2 \leq 3$
13	2 7	1 2	Vertex 13 has neighbours 2 and 7
			Both branches are pruned since $1 + 2 \leq 3$
12	2 4 11	1 2 3	Vertex 12 has neighbours 2, 4, 11
			All 3 branches are pruned since $1 + 3 \leq 4$
11	2 4	1 2	Vertex 11 has neighbours 2 and 4
			Both branches are pruned since $1 + 2 \leq 4$
10	1 5 9	1 2 1	Vertex 10 has neighbours 1, 5, 9
10	1 9 5	1 1 2	The candidates are ordered by colours
			All 3 branches are pruned since $1 + 2 \leq 4$
9	4 5	1 2	Vertex 9 has neighbours 4 and 5
			Both branches are pruned since $1 + 2 \leq 4$
8	3 6 7	1 2 1	Vertex 8 has neighbours 3, 6, 7
8	3 7 6	1 1 2	The candidates are ordered by colours
			All 3 branches are pruned since $1 + 2 \leq 4$
	1 2 3 4 5 6 7	1 1 2 2 3 3 4	All 7 branches are pruned since $0 + 4 \leq 4$

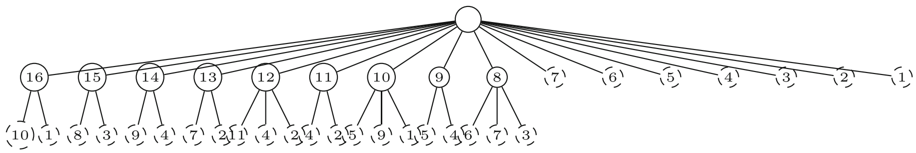

Fig. 6 Search tree for ILS&MCS algorithm

remaining graph without these vertices the upper bound is less than the lower bound and so the algorithm stops.

We demonstrate the calculation of the max-sat-based upper bound on the whole graph in the first node of the search tree. The upper bound calculation is practically the same for $\backslash\upsilon$ nodes (except node $\backslash 6$) or is trivial and the upper bound coincides with the colouring-based upper bound for υ nodes. We also show the calculation of the upper bound in the last node $\backslash 6$ where it allows us to stop branching.

First the graph is coloured with the greedy sequential colouring which colours every vertex one by one using the smallest possible colour on every step (see Fig. 7). This gives the following partitioning of the graph into 5 independent sets: {1, 4, 6, 7, 9}, {2, 5, 8, 10, 12, 14},

Table 3 MAXSAT algorithm main steps

Clique Q	Candidates	Upper bound UB	Comments
	1 2 3 4 5 6	5 Colours, $UB = 4$	Vertex 2 has minimum degree
	7 8 9 10 11		
	12 13 14 15 16		
2	1 3	2 Colours, $UB = 2$	Vertex 2 has neighbours 1 and 3
2 1	3	1 Colour, $UB = 1$	
2 1 3			Maximal clique, $Q^* = \{2, 1, 3\}$
2	3	1 Colour, $UB = 1$	Prune since $\|Q\| + UB \leq \|Q^*\|$
	1 3 4 5 6	5 Colours, $UB = 4$	Vertex 5 has minimum degree
	7 8 9 10 11		
	12 13 14 15 16		
5	4 13	2 Colours, $UB = 2$	Vertex 5 has neighbours 4 and 13
			Prune since $1 + 2 \leq 3$
	1 3 4 6	5 Colours, $UB = 4$	Vertex 7 has minimum degree
	7 8 9 10 11		
	12 13 14 15 16		
7	8 15	2 Colours, $UB = 2$	Vertex 7 has neighbours 8 and 15
			Prune since $1 + 2 \leq 3$
	1 3 4 6	5 Colours, $UB = 4$	Vertex 10 has minimum degree
	8 9 10 11		
	12 13 14 15 16		
10	9 11	2 Colours, $UB = 2$	Vertex 10 has neighbours 9 and 11
			Prune since $1 + 2 \leq 3$
	1 3 4 6 8 9 11	5 Colours, $UB = 4$	Vertex 3 has minimum degree
	12 13 14 15 16		
3	1 4 12	2 Colours, $UB = 2$	Vertex 3 has neighbours 1, 4, 12
			Prune since $1 + 2 \leq 3$
	1 4 6 8 9 11	5 Colours, $UB = 4$	Vertex 4 has minimum degree
	12 13 14 15 16		
4	5 12 13	2 Colours, $UB = 2$	Vertex 4 has neighbours 5, 12, 13
			Prune since $1 + 2 \leq 3$
	1 6 8 9 11	5 Colours, $UB = 4$	Vertex 6 has minimum degree
	12 13 14 15 16		
6	13 14 15	3 Colours, $UB = 3$	Vertex 6 has neighbours 13, 14, 15
6 13	14 15	2 Colours, $UB = 2$	
6 13 14	15	1 Colour, $UB = 1$	
6 13 14 15			Maximal clique, $Q^* = \{6, 13, 14, 15\}$
6 13	15	1 Colour, $UB = 1$	Prune since $2 + 1 \leq 4$
6	14 15	2 Colours, $UB = 2$	Prune since $1 + 2 \leq 4$
	1 8 9 11	5 Colours, $UB = 4$	Prune since $0 + 4 \leq 4$
	12 13 14 15 16		

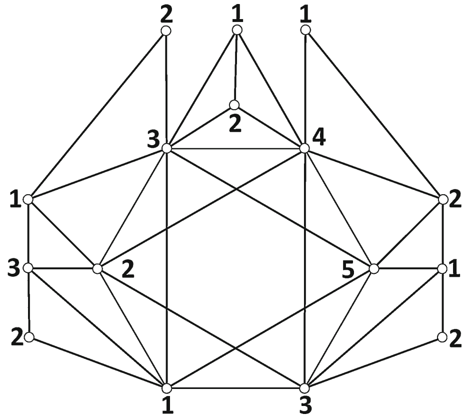

Fig. 7 Greedy colouring of the graph

{3, 11, 13}, {15}, {16}, or the following 5 soft clauses: $(x_1 \vee x_4 \vee x_6 \vee x_7 \vee x_9)$, $(x_2 \vee x_5 \vee x_8 \vee x_{10} \vee x_{12} \vee x_{14})$, $(x_3 \vee x_{11} \vee x_{13})$, (x_{15}), (x_{16}). Only one variable from every soft clause can be equal to 1 since only one vertex from an independent set can be in the maximum clique. Our objective is to maximize the number of satisfied (equal to 1) soft clauses while satisfying all the hard clauses. A hard clause is added for every pair of non-adjacent vertices. For example for a pair of vertices 1 and 4 we add a hard clause $\overline{x_1} \vee \overline{x_4}$.

We use sets notation for soft and hard clauses. For every vertex the corresponding hard clauses can be described by a set of vertices not adjacent to this vertex. Such vertices cannot be in one clique with this vertex. The sets of non-adjacent vertices corresponding to the hard clauses are given below.

$$\begin{aligned}
&1 : \{4, 5, 6, 7, 8, 9, 10, 14, 15\}\\
&2 : \{4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}\\
&3 : \{5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16\}\\
&4 : \{1, 2, 6, 7, 8, 9, 10, 11, 14, 15, 16\}\\
&5 : \{1, 2, 3, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16\}\\
&6 : \{1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 16\}\\
&7 : \{1, 2, 3, 4, 5, 6, 9, 10, 11, 12, 13, 14, 16\}\\
&8 : \{1, 2, 3, 4, 5, 6, 10, 11, 12, 13, 14\}\\
&9 : \{1, 2, 3, 4, 5, 6, 7, 12, 13, 14, 15\}\\
&10 : \{1, 2, 3, 4, 5, 6, 7, 8, 12, 13, 14, 15, 16\}\\
&11 : \{2, 3, 4, 5, 6, 7, 8, 13, 14\}\\
&12 : \{2, 5, 6, 7, 8, 9, 10, 14, 16\}\\
&13 : \{2, 3, 7, 8, 9, 10, 11\}\\
&14 : \{1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 16\}\\
&15 : \{1, 2, 3, 4, 5, 9, 10\}\\
&16 : \{2, 3, 4, 5, 6, 7, 10, 12, 14\}
\end{aligned}$$

We have an upper bound UB equal to 5 due to the colouring. The main idea of the heuristic applied to improve the colouring-based upper bound is to prove that all the 5 soft clauses cannot be satisfied (equal to 1) simultaneously. For this purpose we first take the shortest soft clause and set it equal to 1. In our example it is soft clause {15}. So let $x_{15} = 1$, then all the vertices not adjacent to vertex 15 cannot be in the maximum clique. All these variables are enumerated in the hard clauses set for vertex 15. Thus for $i \in \{1, 2, 3, 4, 5, 9, 10\}$ $x_i = 0$. After substituting these $x_i = 0$ to the soft clauses (except the already satisfied clause {15}) we get the following soft clauses: {6, 7}, {8, 12, 14}, {11, 13}, {16}. This procedure is called unit propagation. Now among the soft clauses there is an unary clause {16} and we set $x_{16} = 1$ to satisfy it. Again variables from the hard clauses for vertex 16 become equal to 0: for $i \in \{2, 3, 4, 5, 6, 7, 10, 12, 14\}$ $x_i = 0$. When we substitute these x_i to the remaining soft clauses we come to an empty soft clause: $\varnothing$, {8}, {11, 13}. In the initial problem this empty clause is clause {1, 4, 6, 7, 9}. This means that clauses {1, 4, 6, 7, 9}, {15}, {16} cannot be satisfied simultaneously. The set of such clauses is called an inconsistent subset of soft clauses. So it is clear that in the optimal solution at least one of these clauses is not equal to 1 and thus we can decrease the current upper bound UB by 1: $UB = 4$.

Since we know that at least one of the soft clauses from the found inconsistent subset is equal to 0 in the optimal solution (but do not know which one) we add dummy variables x_{17}, x_{18}, x_{19} to these clauses and a hard constraint $x_{17} + x_{18} + x_{19} = 1$. So now the soft clauses are {1, 4, 6, 7, 9, 17},{2, 5, 8, 10, 12, 14},{3, 11, 13},{15, 18},{16, 19}. We continue searching for another inconsistent subset of the soft clauses with the next shortest clause—{16, 19}. We should consider its both literals and try to prove that in both cases we get an empty soft clause (a clause which cannot be satisfied). First let $x_{16} = 1$ then for $i \in \{2, 3, 4, 5, 6, 7, 10, 12, 14\}$ $x_i = 0$ and the soft clauses will be: {1, 9, 17}, {8}, {11, 13}, {15, 18}. We continue the unit propagation setting $x_8 = 1$. Then for $i \in \{1, 2, 3, 4, 5, 6, 10, 11, 12, 13, 14\}$ $x_i = 0$ and the soft clauses will be: {9, 17}, $\varnothing$, {15, 18}. So again we have found an inconsistent subset of the soft clauses: {2, 5, 8, 10, 12, 14}, {3, 11, 13}, {16, 19} when $x_{16} = 1$. Now let $x_{19} = 1$ then $x_{17} = x_{18} = 0$ and the soft clauses will be: {1, 4, 6, 7, 9}, {2, 5, 8, 10, 12, 14}, {3, 11, 13}, {15}. We continue the unit propagation setting $x_{15} = 1$. Then for $i \in \{1, 2, 3, 4, 5, 9, 10\}$ $x_i = 0$ and the soft clauses will be: {6, 7}, {8, 12, 14}, {11, 13}. There are no unit clauses, but two clauses are binary and we satisfy clause {6, 7} considering both case: $x_6 = 1$ and $x_7 = 1$. First let $x_6 = 1$ then for $i \in \{1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 16\}$ $x_i = 0$ and the soft clauses will be: {14}, {13}. These clauses are consistent with each other and can be satisfied simultaneously. So for $x_{19} = 1$ there are no inconsistent subset and the original clause {16, 19} can be satisfied simultaneously with all other clauses. Thus we cannot further decrease the upper bound $UB = 4$.

For the last node of the search tree \6 the graph has vertices 1, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 and the corresponding edges. It can be coloured with the greedy sequential colouring in 5 colours giving the following partitioning into 5 independent sets: {1, 7, 9, 14}, {8, 10, 12}, {11, 13}, {15}, {16}. The sets of non-adjacent vertices corresponding to the hard clauses are given below.

$$
\begin{aligned}
&1 : \{7, 8, 9, 10, 14, 15\}\\
&7 : \{1, 9, 10, 11, 12, 13, 14, 16\}\\
&8 : \{1, 10, 11, 12, 13, 14\}\\
&9 : \{1, 7, 12, 13, 14, 15\}\\
&10 : \{1, 7, 8, 12, 13, 14, 15, 16\}\\
&11 : \{7, 8, 13, 14\}
\end{aligned}
$$

$$12 : \{7, 8, 9, 10, 14, 16\}$$
$$13 : \{7, 8, 9, 10, 11\}$$
$$14 : \{1, 7, 8, 9, 10, 11, 12, 16\}$$
$$15 : \{1, 9, 10\}$$
$$16 : \{7, 10, 12, 14\}$$

We have an upper bound UB equal to 5 due to the colouring. We start searching for an inconsistent subset of soft clauses from the shortest clause—$\{15\}$. Let $x_{15} = 1$ then for $i \in \{1, 9, 10\}$ $x_i = 0$ and the soft clauses will be: $\{7, 14\}$, $\{8, 12\}$, $\{11, 13\}$, $\{16\}$. We continue the unit propagation setting $x_{16} = 1$. Then for $i \in \{7, 10, 12, 14\}$ $x_i = 0$ and the soft clauses will be: $\varnothing$, $\{8\}$, $\{11, 13\}$. So we have found an inconsistent subset of soft clauses: $\{1, 7, 9, 14\}$, $\{15\}$, $\{16\}$. This means that the upper bound UB is 4 now.

We add dummy variables x_{17}, x_{18}, x_{19} to these clauses and a hard constraint $x_{17} + x_{18} + x_{19} = 1$ and continue searching for another inconsistent subset with the next shortest clause—$\{16, 19\}$. The current soft clauses are: $\{1, 7, 9, 14, 17\}, \{8, 10, 12\}, \{11, 13\}, \{15, 18\}, \{16, 19\}$ We should consider its both literals. First let $x_{16} = 1$ then for $i \in \{7, 10, 12, 14\}$ $x_i = 0$ and the soft clauses will be: $\{1, 9, 17\}$, $\{8\}$, $\{11, 13\}$, $\{15, 18\}$. We continue the unit propagation setting $x_8 = 1$. Then for $i \in \{1, 10, 11, 12, 13, 14\}$ $x_i = 0$ and the soft clauses will be: $\{9, 17\}$, $\varnothing$, $\{15, 18\}$. So we have found an inconsistent subset of soft clauses: $\{8, 10, 12\}$, $\{11, 13\}$, $\{16, 19\}$ when $x_{16} = 1$. Now let $x_{19} = 1$ then $x_{17} = x_{18} = 0$ and the soft clauses will be: $\{1, 7, 9, 14\}$, $\{8, 10, 12\}$, $\{11, 13\}$, $\{15\}$. We continue the unit propagation setting $x_{15} = 1$. Then for $i \in \{1, 9, 10\}$ $x_i = 0$ and the soft clauses will be: $\{7, 14\}$, $\{8, 12\}$, $\{11, 13\}$. There are no unit clauses, but three clauses are binary and we try to satisfy clause $\{7, 14\}$. First let $x_7 = 1$ then for $i \in \{1, 9, 10, 11, 12, 13, 14, 16\}$ $x_i = 0$ and the soft clauses will be: $\{8\}$, $\varnothing$. So clauses $\{7, 14\}$ and $\{11, 13\}$ are inconsistent if $x_7 = 1$. We should check the second case: $x_{14} = 1$. Then for $i \in \{1, 7, 8, 9, 10, 11, 12, 16\}$ $x_i = 0$ and the soft clauses will be: $\varnothing$, $\{13\}$. So clauses $\{7, 14\}$ and $\{8, 12\}$ are inconsistent if $x_{14} = 1$. This means that when $x_{19} = 1$ clauses $\{1, 7, 9, 14, 17\}$, $\{8, 10, 12\}$, $\{11, 13\}$, $\{15, 18\}$ cannot be satisfied simultaneously. Combining it with the case $x_{16} = 1$ we conclude that clause $\{16, 19\}$ cannot be satisfied simultaneously with clauses $\{1, 7, 9, 14, 17\}$, $\{8, 10, 12\}$, $\{11, 13\}$, $\{15, 18\}$. So at least one of this clauses is equal to 0 and the upper bound can be decreased by one: $UB = 3$.

We add dummy variables $x_{20}, x_{21}, x_{22}, x_{23}, x_{24}$ to the clauses of the found inconsistent subset and constraint $x_{20} + x_{21} + x_{22} + x_{23} + x_{24} = 1$ and continue searching for another inconsistent subset of soft clauses with the next shortest clause—$\{11, 13, 22\}$. The current soft clauses are: $\{1, 7, 9, 14, 17, 20\}$, $\{8, 10, 12, 21\}$, $\{11, 13, 22\}$, $\{15, 18, 23\}$, $\{16, 19, 24\}$ First let $x_{11} = 1$ then for $i \in \{7, 8, 13, 14\}$ $x_i = 0$ and the soft clauses will be: $\{1, 9, 17, 20\}$, $\{10, 12, 21\}$, $\{15, 18, 23\}$, $\{16, 19, 24\}$. There are no unit or binary clauses and according to MAXSAT algorithm we stop here with $UB = 3$.

The search tree for MAXSAT algorithm is given in Fig. 8. The pruned branches are shown by dashed circles. The search tree has 13 nodes.

The improved algorithm—ILS&MAXSAT runs ILS heuristic before starting MAXSAT algorithm. We assume that for our example ILS has found the clique of 4 vertices. This solution with the size $|Q^*| = 4$ is then used to prune branches. Since the max-sat-based upper bound $UB = 4$ for the whole graph is not greater than the lower bound $|Q^*| = 4$ we stop in the first node without any branching (see Table 4).

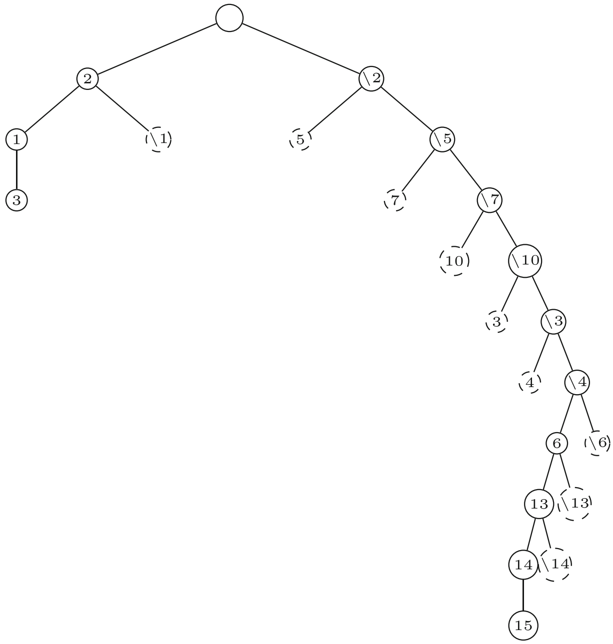

Fig. 8 Search tree for MAXSAT algorithm

Table 4 ILS&MAXSAT algorithm steps

Clique Q	Candidates	Upper bound UB	Comments
	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	5 Colours, $UB = 4$	Prune since $0 + 4 \leq 4$

4 Computational results

We compare MCS and MAXSAT algorithms with our improved versions ILS&MCS and ILS&MAXSAT. Our computational results for graphs from DIMACS library are presented in Tables 6 and 7. All the considered DIMACS instances are presented in Table 5. Table 6 shows the search tree size for MCS, MAXSAT algorithms and its improved versions ILS&MCS and ILS&MAXSAT. The last two columns contain the size of the maximum clique and the size of the best clique found by ILS heuristic. It is clear that the better clique is found by this heuristic the greater is the reduction in the search tree size we have. We run the ILS heuristic with 100000 scans for all the considered instances except *gen400_p0.9_55* and *p_hat1000-3* for which we use 60 millions scans because these two instances are computationally difficult. The greatest reduction of the search tree size is obtained for *gen400* and *p_hat1000-3* instances. For example for *gen400_p0.9_75* instance the search tree of ILS&MCS is 240,000 times smaller than the search tree of MCS algorithm. For MAXSAT algorithm the reduction of the search tree is more than 300 times for this instance.

The computational times are given in Table 7. The total computational time over all DIMACS instances is reduced by 76 % for MCS algorithm and by 70 % for MAXSAT

Table 5 Instance information

Instance	Vertices	Edges	Density	ω	ω_{ILS}
brock200_1	200	14834	0.74543	21	21
brock200_2	200	9876	0.49628	12	12
brock200_3	200	12048	0.60543	15	15
brock200_4	200	13089	0.65774	17	17
brock400_1	400	59723	0.74841	27	23
brock400_2	400	59786	0.74920	29	24
brock400_3	400	59681	0.74788	31	24
brock400_4	400	59765	0.74893	33	26
brock800_1	800	207505	0.64926	23	19
brock800_2	800	208166	0.65133	24	20
brock800_3	800	207333	0.64873	25	19
brock800_4	800	207643	0.64970	26	19
c-fat200-1	200	1534	0.07709	12	12
c-fat200-2	200	3235	0.16256	24	24
c-fat200-5	200	8473	0.42578	58	58
c-fat500-1	500	4459	0.03574	14	14
c-fat500-10	500	46627	0.37376	126	126
c-fat500-2	500	9139	0.07326	26	26
c-fat500-5	500	23191	0.18590	64	64
gen200_p0.9_44	200	17910	0.90000	44	44
gen200_p0.9_55	200	17910	0.90000	55	55
gen400_p0.9_55	400	71820	0.90000	55	54
gen400_p0.9_65	400	71820	0.90000	65	64
gen400_p0.9_75	400	71820	0.90000	75	75
hamming10-2	1024	518656	0.99022	512	512
hamming6-2	64	1824	0.90476	32	32
hamming6-4	64	704	0.34921	4	4
hamming8-2	256	31616	0.96863	128	128
hamming8-4	256	20864	0.63922	16	16
johnson16-2-4	120	5460	0.76471	8	8
johnson8-2-4	28	210	0.55556	4	4
johnson8-4-4	70	1855	0.76812	14	14
keller4	171	9435	0.64912	11	11
keller5	776	225990	0.75155	27	27
MANN_a27	378	70551	0.99015	126	126
MANN_a45	1035	533115	0.99630	345	344
MANN_a9	45	918	0.92727	16	16
p_hat1000-1	1000	122253	0.24475	10	10
p_hat1000-2	1000	244799	0.49009	46	44
p_hat1000-3	1000	371746	0.74424	68	68
p_hat1500-1	1500	284923	0.25343	12	11

Table 5 continued

Instance	Vertices	Edges	Density	ω	ω_{ILS}
p_hat1500-2	1500	568960	0.50608	65	61
p_hat300-1	300	10933	0.24377	8	8
p_hat300-2	300	21928	0.48892	25	25
p_hat300-3	300	33390	0.74448	36	35
p_hat500-1	500	31569	0.25306	9	9
p_hat500-2	500	62946	0.50458	36	34
p_hat500-3	500	93800	0.75190	50	48
p_hat700-1	700	60999	0.24933	11	11
p_hat700-2	700	121728	0.49756	44	42
p_hat700-3	700	183010	0.74805	62	60
san1000	1000	250500	0.50150	15	15
san200_0.7_1	200	13930	0.70000	30	30
san200_0.7_2	200	13930	0.70000	18	18
san200_0.9_1	200	17910	0.90000	70	70
san200_0.9_2	200	17910	0.90000	60	60
san200_0.9_3	200	17910	0.90000	44	44
san400_0.5_1	400	39900	0.50000	13	13
san400_0.7_1	400	55860	0.70000	40	40
san400_0.7_2	400	55860	0.70000	30	15
san400_0.7_3	400	55860	0.70000	22	22
san400_0.9_1	400	71820	0.90000	100	100
sanr200_0.7	200	13868	0.69688	18	18
sanr200_0.9	200	17863	0.89764	42	42
sanr400_0.5	400	39984	0.50105	13	12
sanr400_0.7	400	55869	0.70011	21	20

Table 6 Search tree size

Instance	MCS	ILS&MCS	MAXSAT	ILS&MAXSAT
brock200_1	145354	130378	38887	38887
brock200_2	2526	1747	1265	999
brock200_3	8281	8276	2481	2399
brock200_4	31267	16850	11534	6070
brock400_1	87946118	88555048	19991548	19878493
brock400_2	34145195	30682956	8395282	7918390
brock400_3	66379744	66280298	16632868	16484392
brock400_4	29696341	17963868	9875411	5337967
brock800_1	1097174023	1095645796	386532897	386528718
brock800_2	972110520	970862419	315456602	315441226
brock800_3	625234820	625139200	191276571	191269817
brock800_4	424101537	424176492	134517995	134517995

Table 6 continued

Instance	MCS	ILS&MCS	MAXSAT	ILS&MAXSAT
c-fat200-1	188	188	4	4
c-fat200-2	176	176	1	1
c-fat200-5	142	142	26	26
c-fat500-1	486	486	1	1
c-fat500-10	374	374	1	1
c-fat500-2	474	474	6	6
c-fat500-5	436	436	4	4
gen200_p0.9_44	33254	17917	9609	5793
gen200_p0.9_55	62184	588	9049	46
gen400_p0.9_55	3425049256	55079436	13710615743	3567404301
gen400_p0.9_65	6500277298	822991	2183044625	92631755
gen400_p0.9_75	10140428816	41445	583227541	1825520
hamming10-2	511	511	1	1
hamming6-2	31	0	1	1
hamming6-4	81	81	33	33
hamming8-2	127	0	1	1
hamming8-4	31793	31782	2250	2250
johnson16-2-4	237951	237951	72345	11
johnson8-2-4	25	25	10	10
johnson8-4-4	125	114	11	11
keller4	6156	6118	1569	1569
keller5	10339211493	10337321299	267489494	261876583
MANN_a27	9089	8816	2475	2331
MANN_a45	221476	219979	77218	76483
MANN_a9	42	25	8	8
p_hat1000-1	120818	116527	50308	49258
p_hat1000-2	12842526	11822141	4721336	4404368
p_hat1000-3	–	8773710250	7059591235	2195384814
p_hat1500-1	821535	772219	341308	319060
p_hat1500-2	660539819	607200969	242104729	193783540
p_hat300-1	1493	1493	500	304
p_hat300-2	2022	1611	1181	611
p_hat300-3	276636	133550	96544	51734
p_hat500-1	8007	8007	4789	4523
p_hat500-2	47563	33735	24680	10627
p_hat500-3	7732392	7261601	2308431	2082024
p_hat700-1	22509	14041	15614	10204
p_hat700-2	326749	229048	152541	99656
p_hat700-3	98911559	81631372	39268710	28213254
san1000	84314	465	23433	1
san200_0.7_1	406	80	161	11
san200_0.7_2	807	62	439	1

Table 6 continued

Instance	MCS	ILS&MCS	MAXSAT	ILS&MAXSAT
san200_0.9_1	17292	85	440	4
san200_0.9_2	2487	107	8044	38
san200_0.9_3	4734	112	12583	10571
san400_0.5_1	1689	184	515	1
san400_0.7_1	31206	200	4872	1576
san400_0.7_2	14463	190	778	778
san400_0.7_3	125373	174	40461	142
san400_0.9_1	2213	201	23734	1457
sanr200_0.7	67778	56750	22604	20090
sanr200_0.9	2618612	1577939	388858	220045
sanr400_0.5	170260	169838	76786	76533
sanr400_0.7	29275634	29363286	8632873	8632115

Table 7 Computational time

Instance	ILS	MCS	ILS&MCS	MAXSAT	ILS&MAXSAT
brock200_1	6.1	0.52497	6.56446	0.35	6.469
brock200_2	12.1	0.00662	12.10550	0.02	12.116
brock200_3	9.8	0.02657	9.82708	0.041	9.837
brock200_4	8.5	0.09639	8.55635	0.097	8.574
brock400_1	25.8	384.737	411.441	229.197	250.792
brock400_2	25.4	166.361	181.599	103.637	125.677
brock400_3	25.6	269.289	294.048	175.736	199.894
brock400_4	25.3	134.355	117.259	112.402	93.8
brock800_1	0.001	5215.95	5200.5	5060.008	5059.954
brock800_2	0.001	4713.61	4712.3	4431.803	4431.588
brock800_3	0.001	3152.91	3149.4	2929.898	2929.796
brock800_4	0.001	2333.28	2332.9	2223.352	2223.014
c-fat200-1	11.1	0.00018	11.10028	0.006	11.105
c-fat200-2	10	0.00026	10.00023	0.005	10.007
c-fat200-5	6.6	0.00049	6.60072	0.008	6.607
c-fat500-1	61.7	0.00050	61.70054	0.058	61.759
c-fat500-10	41.7	0.00361	41.70378	0.06	41.757
c-fat500-2	63.3	0.00076	63.30075	0.065	63.365
c-fat500-5	55.9	0.00149	55.90159	0.056	55.952
gen200_p0.9_44	2.2	0.23820	2.35478	0.176	2.303
gen200_p0.9_55	2	0.37943	2.00809	0.133	2.015
gen400_p0.9_55	3655	35013.1	4361.2	315523.148	72251.747
gen400_p0.9_65	7.7	65290	24.1017	43121.357	2423.131
gen400_p0.9_75	7	93551	8.09317	11324.809	72.164
hamming10-2	5	0.06055	5.86192	0.822	5.891

Table 7 continued

Instance	ILS	MCS	ILS&MCS	MAXSAT	ILS&MAXSAT
hamming6-2	0.3	0.00006	0.30001	0.001	0.302
hamming6-4	1.5	0.00008	1.50008	0.001	1.501
hamming8-2	1.2	0.00113	1.20001	0.023	1.223
hamming8-4	13.8	0.12928	13.93468	0.053	13.847
johnson16-2-4	2.1	0.13276	2.23412	0.489	2.101
johnson8-2-4	0.2	0.00003	0.20003	0.001	0.201
johnson8-4-4	0.7	0.00034	0.70028	0.001	0.702
keller4	6.6	0.01601	6.61663	0.022	6.621
keller5	96	77010.2	76997.7	5389.209	5397.849
MANN_a27	1.6	0.35960	1.99189	0.208	1.801
MANN_a45	4.6	105.52	109.16	27.862	34.098
MANN_a9	0.2	0.00014	0.20008	0.001	0.201
p_hat1000-1	302.1	0.31752	302.40619	1.74	303.888
p_hat1000-2	171.5	116.081	280.01	116.002	277.569
p_hat1000-3	54184.8	–	197887	373289.113	133589.2
p_hat1500-1	693.1	2.67632	695.71576	11.685	703.172
p_hat1500-2	345.6	9743.75	9325.8	9809.065	8269.908
p_hat300-1	27.3	0.00268	27.30271	0.039	27.336
p_hat300-2	16.5	0.01019	16.50828	0.047	16.544
p_hat300-3	10.1	1.57862	11.0	1.199	10.753
p_hat500-1	75.3	0.01836	75.31833	0.17	75.474
p_hat500-2	40.9	0.31434	41.14326	0.546	41.232
p_hat500-3	23.9	76.7215	95.7	46.196	67.567
p_hat700-1	146.9	0.06007	146.94318	0.501	147.342
p_hat700-2	76.9	2.91445	79.08934	3.406	79.466
p_hat700-3	44.1	1337.55	1155.92	1057.948	820.295
san1000	464.1	1.05469	464.11551	0.782	464.688
san200_0.7_1	6.4	0.00313	6.40082	0.014	6.411
san200_0.7_2	7.4	0.00316	7.40044	0.017	7.412
san200_0.9_1	1.8	0.11367	1.80094	0.021	1.814
san200_0.9_2	2	0.02653	2.00113	0.118	2.018
san200_0.9_3	2.2	0.02888	2.20109	0.192	2.364
san400_0.5_1	64.6	0.00929	64.60168	0.063	64.653
san400_0.7_1	25.7	0.42034	26	0.157	25.802
san400_0.7_2	32.8	0.10018	32.80426	0.092	32.884
san400_0.7_3	34.6	0.85591	34.6	0.469	34.671
san400_0.9_1	6.2	0.05921	6.20675	1.199	6.378
sanr200_0.7	7.3	0.21627	7.47879	0.188	7.459
sanr200_0.9	2.2	16.2453	13.2	4.913	4.9
sanr400_0.5	49.3	0.50248	49.75986	0.717	50.016
sanr400_0.7	31.3	104.268	138.4	88.247	116.187

algorithm. Thus the speedup in solving all the considered DIMACS instances in total is about 4 times for MCS and 3.5 times for MAXSAT algorithm. MCS algorithm is unable to solve *p_hat1000-3* (at least in 7 days) while ILS&MCS solves it in 55 h (more than 5 times speedup). MAXSAT algorithm solves *p_hat1000-3* instance in 55 h, while ILS&MAXSAT needs 37 h (about 1.5 times speedup). For *gen400_p0.9_55*, *gen400_p0.9_65*, and *gen400_p0.9_75* instances ILS&MCS gives 8, 2,700, and 11,500 times speedups correspondingly. And ILS&MAXSAT gives 4,500, 17, and 150 times speedups for these instances. However, our approach is usually slower for simple instances because it is not efficient to perform 100000 scans of ILS heuristic for such graphs. In whole we have much better results for several hard instances and comparable results for other instances.

Most of the DIMACS instances are solved faster by MAXSAT algorithm than by MCS. Only three *gen400* instances are solved by MCS algorithm faster. Probably for these instances max-sat-based upper bound is not much tighter than the colouring-based one. Colouring-based upper bound is computed faster than max-sat-based upper bound since the latter solves both vertex colouring and max-sat problems.

Acknowledgments The authors would like to thank professor Mauricio Resende and his co-authors for the source code of their powerful ILS heuristic. We are also thankful to Chu-Min Li and Zhe Quan for the source code of their efficient MAXSAT algorithm. The authors are supported by LATNA Laboratory, National Research University Higher School of Economics (NRU HSE), Russian Federation government grant, ag. 11.G34.31.0057. Mikhail Batsyn is supported by Federal Grant-in-Aid Program “Research and development on priority directions of development of the scientific-technological complex of Russia for 2007–2013” (Governmental Contract No. 14.514.11.4065).

References

1. Andrade, D.V., Resende, M.G.C., Werneck, R.F.: Fast local search for the maximum independent set problem. J. Heuristics **18**(4), 525–547 (2012). doi:10.1007/s10732-012-9196-4
2. Balas, E., Yu, C.S.: Finding a maximum clique in an arbitrary graph. SIAM J. Comput. **15**(4), 1054–1068 (1986)
3. Bertoni, A., Campadelli, P., Grossi, G.: A discrete neural algorithm for the maximum clique problem: analysis and circuit implementation. In: Proceedings of Workshop on Algorithm, Engineering, WAE’97, pp. 84–91 (1997)
4. Boginski, V., Butenko, S., Pardalos, P.M.: On structural properties of the market graph. In: Nagurney, A. (ed.) Innovations in Financial and Economic Networks, pp 29–45. Edward Elgar Publishing, London (2003)
5. Bomze, I., Budinich, M., Pardalos, P.M., Pelillo, M.: The Maximum Clique Problem. Handbook of Combinatorial Optimization. Kluwer, Dordrecht (1999)
6. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. Commun. ACM **16**(9), 575–577 (1973). doi:10.1145/362342.362367
7. Brouwer, A., Shearer, J., Sloane, N., Smith, W.: A new table of constant weight codes. IEEE Trans. Inf. Theory **36**(6), 1334–1380 (1990). doi:10.1109/18.59932
8. Butenko, S., Wilhelm, W.E.: Clique-detection models in computational biochemistry and genomics. Eur. J. Oper. Res. **173**(1), 1–17 (2006). doi:10.1016/j.ejor.2005.05.026
9. Carraghan, R., Pardalos, P.M.: An exact algorithm for the maximum clique problem. Oper. Res. Lett. **9**(6), 375–382 (1990). doi:10.1016/0167-6377(90)90057-C
10. Du, D., Pardalos, P.M.: Handbook of Combinatorial Optimization, Supplement vol A, p. 648. Springer, Berlin (1999)
11. Fahle, T.: Simple and fast: improving a branch-and-bound algorithm for maximum clique. In: Proceedings of the 10th Annual European Symposium on Algorithms (ESA ’02), pp 485–498. Springer, London, UK (2002)
12. Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. J. Glob. Optim. **6**(2), 109–133 (1995). doi:10.1007/BF01096763
13. Funabiki, N., Takefuji, Y., Lee, K.C.: A neural network model for finding a near-maximum clique. J. Parallel Distrib. Comput. **14**(3), 340–344 (1992). doi:10.1016/0743-7315(92)90072-U

14. Glover, F., Laguna, M.: Tabu Search. Kluwer, Dordrecht (1997)
15. Grosso, A., Locatelli, M., Pullan, W.: Simple ingredients leading to very efficient heuristics for the maximum clique problem. J. Heuristics **14**(6), 587–612 (2008). doi:10.1007/s10732-007-9055-x
16. Jenelius, E., Petersen, T., Mattsson, L.: Importance and exposure in road network vulnerability analysis. Transp. Res. Part A: Policy Pract. **40**(7), 537–560 (2006). doi:10.1016/j.tra.2005.11.003
17. Jerrum, M.: Large cliques elude the metropolis process. Random Struct. Algorithms **3**(4), 347–359 (1992). doi:10.1002/rsa.3240030402
18. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum, New York (1972)
19. Konc, J., Janezic, D.: An improved branch and bound algorithm for the maximum clique problem. MATCH Commun. Math. Comput. Chem. **58**, 569–590 (2007)
20. Kopf, R., Ruhe, G.: A computational study of the weighted independent set problem for general graphs. Found. Control Eng. **12**, 167–180 (1987)
21. Li, C.M., Quan, Z.: Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In: Proceedings of the 2010 22nd IEEE International Conference on Tools with Artificial Intelligence—Volume 01 (ICTAI'10), pp 344–351. IEEE, Arras, France (2010a)
22. Li, C.M., Quan, Z.: An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10), pp. 128–133. AAAI Press, Atlanta, USA (2010b)
23. Marchiori, E.: Genetic, iterated and multistart local search forthe maximum clique problem. In: Applications of Evolutionary Computing, Springer, LNCS, pp. 112–121. Springer, Berlin (2002)
24. Matula, D.W., Marble, G., Isaacson, J.D.: Graph coloring algorithms. In: Read, R.C. (ed.) Graph Theory and Computing, pp 109–122. Academic Press, New York (1972)
25. Mycielski, J.: Sur le coloriage des graphes. Colloq. Math. **3**, 161–162 (1955)
26. Pullan, W., Hoos, H.H.: Dynamic local search for the maximum clique problem. J. Artif. Int. Res. **25**(1), 159–185 (2006)
27. Singh, A., Gupta, A.K.: A hybrid heuristic for the maximum clique problem. J. Heuristics **12**(1–2), 5–22 (2006). doi:10.1007/s10732-006-3750-x
28. Sloane, N.J.A.: Unsolved problems in graph theory arising from the study of codes. Graph Theory Notes NY **18**(11), 11–20 (1989)
29. Tomita, E., Kameda, T.: An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. J. Glob. Optim. **37**(1), 95–111 (2007)
30. Tomita, E., Seki, T.: An efficient branch-and-bound algorithm for finding a maximum clique. In: Proceedings of the 4th International Conference on Discrete Mathematics and Theoretical Computer Science (DMTCS'03), pp. 278–289. Springer, Berlin, Heidelberg (2003)
31. Tomita, E., Sutani, Y., Higashi, T., Takahashi, S., Wakatsuki, M.: A simple and faster branch-and-bound algorithm for finding a maximum clique. In: Proceedings of the 4th International Conference on Algorithms and Computation (WALCOM'10), pp. 191–203. Springer, Berlin, Heidelberg (2010). doi: 10.1007/978-3-642-11440-3_18