Computers & Operations Research 64 (2015) 293–303

Contents lists available at ScienceDirect

Computers & Operations Research

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

Infra-chromatic bound for exact maximum clique search

Pablo San Segundo [a,*], Alexey Nikolaev [b], Mikhail Batsyn [b]

[a] Centro de Automática y Robótica (CAR), UPM-CSIC, C/ Jose Gutiérrez Abascal, 2, 28006 Madrid, Spain
[b] Laboratory of Algorithms and Technologies for Networks Analysis, National Research University Higher School of Economics, 136 Rodionova Street, Niznhy Novgorod, Russia

ARTICLE INFO

Available online 24 June 2015

Keywords:
Approximate coloring
Branch-and-bound
MaxSAT
Combinatorial optimization

ABSTRACT

Many efficient exact branch and bound maximum clique solvers use approximate coloring to compute an upper bound on the clique number for every subproblem. This technique reasonably promises tight bounds on average, but never tighter than the chromatic number of the graph.

Li and Quan, 2010, AAAI Conference, p. 128–133 describe a way to compute even tighter bounds by reducing each colored subproblem to maximum satisfiability problem (MaxSAT). Moreover they show empirically that the new bounds obtained may be lower than the chromatic number.

Based on this idea this paper shows an efficient way to compute related "infra-chromatic" upper bounds without an explicit MaxSAT encoding. The reported results show some of the best times for a stand-alone computer over a number of instances from standard benchmarks.

1. Introduction

For a given graph, a complete subgraph, alias clique, is a graph which vertices are all pairwise adjacent. Finding a hidden clique with the maximum number of vertices is a deeply studied NP-hard problem known as the *maximum clique problem* (MCP). MCP has found many applications in a wide scope of fields [1]. Correspondence related problems appear in computational biology, [2], robotics [3–4], computer vision [5]. Finding cohesive clusters in networks is another typical application. Its use in mining correlated stocks is also worth noting.

Exhaustive clique enumeration behind exact MCP can be traced back to the Bron and Kerbosch algorithm [6]. A basic branch and bound (BnB) algorithm which computes primitive bounds for every subproblem was described in [7]. Since then, researchers have tried to devise ways of establishing tighter bounds. Fahle [8] and Régin [9] use a constraint-based approach to prune the search space, but the majority of current efficient solvers are color-based, i.e. they employ a greedy coloring heuristic to compute an upper bound for the clique number of every subproblem, as in [10–19]. Interesting recent comparison surveys for exact maximum clique algorithms have been reported by Prosser [20] and Wu and Hao's [24]. They show MCS [15], MaxCLQ [18], and bit optimized BBMC [12–13] as the current fastest algorithms at present.

The theoretical foundation for approximate color-based algorithms can be found in the following proposition [21]: *the chromatic number of every graph G is an upper bound on its clique number*

$$\chi(G) \geq \omega(G) \tag{1}$$

It is worth noting that, although color-based bounds are reasonably tight on average, Mycielski showed how to build graphs in which $\chi(G) - \omega(G) \geq n$, $\forall n \in \mathbb{N}$ [22]. This constitutes a drawback for the color-based approach.

Li and Quan in [17,18] show empirically that it is possible to compute better approximations to the clique number than the chromatic number by encoding a colored graph to MaxSAT and apply typical logical inferences.

This paper describes a new efficient upper bound related to the previous bound but reasoning with color set information. The procedure is applicable to a more constrained set of cases than [17,18] but is also faster to compute since it does not require excessive MaxSAT inferences.

1.1. Preliminaries

A simple undirected graph $G=(V,E)$ consists of a finite set of vertices $V=\{v_1, v_2, \cdots, v_n\}$ and edges $E \subseteq V \times V$ that pair distinct vertices. Two vertices are said to be adjacent (neighbors) if they are connected by an edge. For any vertex $v \in V$, $N_G(v)$ (or simply $N(v)$ when the graph is clear from the context) refers to the neighbor set of v in G. Any subset of vertices $U \subseteq V$ *induces* a new subgraph $G=(U,E')=G[U]$ with vertex set U and edge set $E' \subseteq E$ such that

* Corresponding author.
Tel.: +34 91 7454660, +34 91 3363061; fax: +34 91 3363010.
E-mail address: pablo.sansegundo@upm.es (P. San Segundo).

http://dx.doi.org/10.1016/j.cor.2015.06.009
0305-0548/

both endpoints of any edge in E' are in U. $G_v \subseteq G$ denotes the graph induced by the neighbor set of vertex v in G.

Some definitions used in this paper are:

- *maximal clique*: a clique that cannot be enlarged by any other vertex in the graph;
- *maximum clique*: a clique which has maximum order (number of vertices);
- *independent set*: a subset of a graph G, which elements are pairwise non-adjacent;
- *vertex coloring*: an assignment of colors $c(v): V \to \mathbb{N}$ to every vertex of a graph so that any two adjacent vertices have different colors;
- *chromatic number* $\chi(G)$: the minimum number of colors in which it is possible to color graph G. Finding the chromatic number of a graph is an NP-hard problem;
- *clique number* $\omega(G)$: the number of vertices of a maximum clique in G;
- *sequential vertex coloring*: an approximate coloring procedure that iteratively colors vertices in some predefined order;
- *greedy vertex coloring* (SEQ): a sequential vertex coloring in which every vertex is assigned the smallest possible color.

Additional standard notation in the paper is $\deg(v)$for vertex v degree and ΔG for the maximum degree of a graph G. Color set notation $C(G) = \{C_1, C_2, \cdots, C_k\}$denotes a vertex coloring of size k, i.e. a coloring that employs k different color numbers. $C(G)$ partitions the vertex set into k disjoint independent color sets C_i, each one containing every vertex with color number i.

2. The branch-and-bound algorithm

Most efficient BnB algorithms for the MCP employ some form of systematic enumeration of maximal cliques together with SEQ heuristic to compute an upper bound for the clique number in every subproblem.

Although other formulations are possible (e.g. a binary tree), clique enumeration is typically described as a recursive procedure that branches on every candidate vertex in order to enlarge the clique in the current node. This leads to a binomial search tree in which, at depth level k, every possible clique of size k is considered [7]. Leaf nodes hold maximal cliques in the path to the root node and are evaluated to test whether their clique number is greater than the best clique found so far. If this is true, they replace the existing solution.

Let S be the clique at the current node and let S_{max} be the best solution found during search at any moment. All enumerations starting from S that cannot improve S_{max} may be pruned. For any candidate vertex v, color-based BnB solvers use $c(v)$ as an upper bound for the clique number of the hanging subproblem G_v. The pruning condition is typically formulated as:

$$|S| + c(v) \leq |S_{max}| \tag{2}$$

Konc and Janečič in [10] made an interesting interpretation of (2). They define parameter $k_{min} = |S_{max}| - |S| + 1$, so that the pruning condition (2) becomes $c(v) < k_{min}$. Leading algorithms use k_{min} explicitly to advantage during the bounding phase, as in the paper.

Tomita and Seki in [14] introduced the idea of branching on maximum color. This has two important consequences: 1. If any candidate vertex in $C_{k_{min}-1}$ is pruned, then all the remaining vertices are also pruned; 2. Search is directed towards large cliques (a coloring of a clique of size k must have size k).

Candidate vertices with the smallest degree should be picked first at the root node to reduce the size of the search tree. A common strategy for initial vertex sorting is the following heuristic used to compute graph degeneracy: at the beginning, a vertex v with minimum degree is removed from the initial set V and placed last in the new list V'. In the next iteration, the vertex with minimum degree in graph $G\backslash\{v\}$ is placed one before last in V'; the process continues until all vertices are ordered. At the root node vertices should be selected by non-decreasing degree to reduce branching, so they are picked in *reverse order* from V'. It is worth noting that a number of more sophisticated orderings concerning tie-breaks have been suggested in practice (see e.g. [15]).

A simple upper bound for every vertex at the root node is the minimum value between its position in the ordering and the maximum degree of the graph plus 1. It is used in leading solvers BBMC and MCS and others. Tighter bounds may possibly produce smaller search trees as suggested in [19].

Another important idea also used in MCS and BBMC is to keep the initial vertex ordering fixed throughout the search. This produces on average tighter SEQ colorings compared to vertices sorted according to color as proposed in earlier algorithms [10,14]. Besides this *implicit branching strategy*, MCS also describes *recoloring* (Re-NUMBER is the term given to the procedure in the original paper). Recoloring is a repair mechanism which is also related to infra-chromatic bounding described in this paper. It is discussed in Section 4 inside the proposed new algorithmic framework.

Procedures described in the paper employ the following variables:

- U: a list of candidate vertices sorted according to the initial ordering;
- U_v: a list of candidate vertices in the child subproblem which contains only neighbors of vertex v;
- S: the current clique;
- S_{max}: the best clique found so far;
- L: a list of candidate vertices sorted according to color (highest color last);
- L_v: a list of candidate vertices sorted according to color in the child subproblem which contains only neighbors of vertex v;
- F: a list of forbidden colors;
- $c(v)$: the color number assigned to vertex v;
- $N_U(v)$: the neighbor set of vertex v from the list of candidate vertices in U;
- k_{min}: a pruning threshold, i.e. vertices assigned a lower color number will be pruned;
- $C(G_v)$: SEQ coloring of G_v, the subgraph induced by v;
- C_k: a color set of a SEQ coloring that contains all vertices with color number k.

Listing 1 outlines the reference MCP algorithm described previously. Steps 1–2, 4–10, 13–15 implement systematic enumeration (with repetitions). Step 3 is the pruning step, which depends on the upper bounds computed in UPPERBOUND (step 12). UPPERBOUND calls the new bounding procedure described in Section 5.

Listing 1. Outline of the reference MCP algorithm

Input: a simple graph $G = (V, E)$with vertices sorted by *smallest degree-last*
Output: a maximum clique with vertex set S_{max}
REFMC (U, S, S_{max}, C, L)
Initial step:
$U \leftarrow V$, $S \leftarrow \phi$, $S_{max} \leftarrow \phi$, $c(v_i) := \min\{i, \Delta G + 1\}$, $L \leftarrow V$
1. **repeat until** $U = \phi$
2. select a vertex v from L in reverse order //maximum color branching
3. **if** $(|S| + c(v) \leq |S_{max}|)$ **then return** //pruning step

```
4.    U←U\{v}
5.    S←S ∪ {v}
6.    U_v←N_U(v)    //U_v preserves initial relative order of vertices
7.    if U_v = φ then
8.      if |S| > |S_max| then S_max←S    //maximal clique found
9.      S←S\{v} and goto step 2          //next candidate
10.   endif
11.   k_min:= | S_max|−|S|+1
12    UPPERBOUND(U_v, k_min, C(U_v), L_v)
13.   REFMC (U_v, S, S_max, C(U_v), L_v)
14.  S←S\{v}
15.  endrepeat
```

3. The greedy coloring algorithm

UPPERBOUND procedure in Listing 2 has been designed as a *class coloring* framework rather than a concrete algorithm.

Class coloring is an alternative implementation of SEQ in which color sets are computed sequentially. It outputs exactly the same coloring as standard SEQ so it is not an independent coloring heuristic. Regarding maximum clique color-based algorithms it presents a number of advantages [12]:

- It implicitly determines the subset of vertices that will not be pruned, given the first $k_{min}-1$ color sets.
- It has been employed successfully by BBMC, our leading bit optimized algorithm. In general, it integrates well with a bitstring encoding for MCP.
- The new bounding procedure proposed in this paper also integrates well with class coloring. This is further developed in Section 5.

Steps 1–5, 9–14 in listing 2 implement class coloring. Steps 15–17 sort vertices in L according to color. Step 7 contains a template DO function for a possible repair mechanism which tries to improve color numbers obtained by class coloring. DO is called for every potential candidate vertex (i.e. with color number not less than threshold k_{min}). All parameters except F (i.e. the current vertex v and its color set C_k, k_{min} threshold and the color sets below the threshold) are typical for a recoloring procedure. F is a list of forbidden colors, which may or may not be used in the specific instantiation. It could also be modified inside DO.

Two implementations of DO have been considered in this paper. Section 4 briefly discusses recoloring within the framework because it is very much connected with the new idea. Recoloring does not make use of the forbidden colors list F. In Section 6 the new infra-chromatic upper bound procedure is described.

Listing 2. Outline of the class coloring framework

Input: An induced graph $G_v = G[U_v]$; threshold $k_{min} = |S_{max}| - |S| + 1$
Output: 1) A coloring $C(G_v)$;
2) A list of candidate vertices L sorted in non-decreasing order of colors

UPPERBOUND (U_v, k_{min}, C, L)
Variables: a list U (for vertices), a list F (for forbidden colors), an integer k

Initial step: $U \leftarrow U_v$, $F \leftarrow \phi$, $k \leftarrow 1$, $C \leftarrow \phi$

```
1.  repeat until U = φ
2.    C_k←U
3.    repeat until all vertices in C_k have been selected
4.      choose the first vertex v from C_k not selected
        previously
5.      result← FAIL
6.      if k ≥ k_min then
7.          result←DO (v, k_min, C_1, C_2, ⋯, C_{k_min−1}, C_k, F)
8.      endif
9.      if result=FAIL then C_k←C_k\N(v)
10.     U←U\{v}              //updates uncolored vertices
11.   endrepeat
12.   C←C ∪ C_k
13.   k : =k+1
14. endrepeat
15. for i := 1 to k-1
16.   L←L ∪ C_i //copies candidate vertices to L sorted by colors
17. endfor
```

4. Recoloring

Recoloring (termed *Re-Number* in the original paper) was first described in MCS algorithm [15]. It is an attempt to recolor a vertex v with color number $c(v) \geq k_{min}$ by *repairing* partial coloring $C = \{C_1, C_2, \cdots, C_{k_{min}-1}\}$ so that $c(v) < k_{min}$ in a new vertex coloring of the same size. In order to compute the new coloring, 1. A color set C_i, $1 \leq i \leq (k_{min}-2)$ has to exist with exactly one vertex $w \in C_i$ adjacent to v and 2. A color set C_j, $i < j \leq k_{min}-1$ has to exist such that no vertex in C_j is adjacent to w. If both conditions are met the new coloring is computed by recoloring both vertices: $c(v) = i$ and $c(w) = j$. An important consideration is that vertex w can never go *backwards* in the coloring (i.e. $j < i$) because of the following proposition:

Proposition 1. *Given any greedy sequential coloring C, every vertex with color number k has to be adjacent to at least one vertex in every color set* $C_1, C_2, \ldots, C_{k-1}$.

Proof. By contradiction. If this is not so, a vertex v and a color set C_j have to exist such that $c(v) = i$, $j < i$ and $N(v) \cap C_j = \phi$. But if this were the case, vertex v would have been assigned color number j in the first place by the greedy coloring. □

Listing 3 describes how to integrate recoloring in the class coloring framework. DO_RECOL substitutes template function DO in Listing 2. It does not make use of the forbidden colors list F so it has been removed from the parameter list (the empty set is assumed as default value). If the recoloring attempt succeeds, DO_RECOL returns SUCCESS and FAIL otherwise.

It is worth noting that the framework calls DO_RECOL for every candidate vertex *once all color sets below* k_{min} *have been computed*. This is an important difference with the original recoloring proposed in MCS because it is now possible that a previous recoloring opens a color set for vertex v directly. Step 11 checks for that circumstance.

An example of recoloring appears in Fig. 1. The initial SEQ coloring (starting from vertex 1) is of size 3. The repair heuristic manages to

Listing 3. Recoloring procedure in the class coloring framework

```
DO_RECOL (v, k_min, C_1, C_2, ..., C_{k_min-1}, C_k)                    //v ∈ C_k
1. for k_1 := 1 to k_min − 2
2.     if |C_{k_1} ∩ N(v)| = 1 then w ← C_{k_1} ∩ N(v)                  //w ∈ C_{k_1}
3.         for k_2 := k_1 + 1 to k_min − 1                              //according to proposition 1
4.             if C_{k_2} ∩ N(w) = φ then                               //standard double-move recoloring
5.                 C_{k_1} ← (C_{k_1} \ {w}) ∪ {v}                      //removes w from C_{k_1} and adds v to C_{k_1}
6.                 C_{k_2} ← C_{k_2} ∪ {w}                              //adds w to C_{k_2}
7.                 C_k ← C_k \ {v}                                      //removes v from C_k
8.                 return SUCCESS
9.             endif
10.        endfor
11.    elseif C_{k_1} ∩ N(v) = φ then                                   //special one-move recoloring
12.        C_{k_1} ← C_{k_1} ∪ {v}
13.        C_k ← C_k \ {v}
14.        return SUCCESS
15.    endif
16.endfor
17.return FAIL
```

find the minimum coloring of size 2 by assigning vertex 5 color number 1 (green) and vertex 1 color number 3 (cyan).

5. Infra-chromatic bounds

In [13], recoloring was reported to be an important pruning strategy for the more difficult, dense instances. However, the limitation of the chromatic number is still there since the repair mechanism results in a new coloring.

To overcome this limitation Li and Quan in [17] recently proposed an encoding of MCP into MaxSAT assuming the input graph to be colored (definition 1, proposition 2).

Definition 1. [17]. Let G be a graph partitioned into independent sets (color sets). Then the independent set based MaxSAT encoding of MCP is defined as follows: (1) each vertex v_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 (v_i, v_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. [17]. *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 give a maximum clique in G.

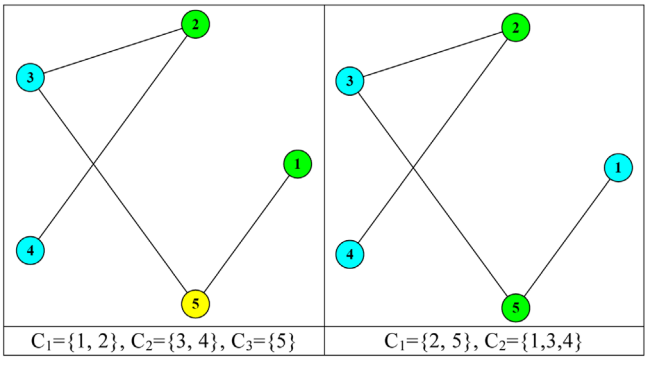

Fig. 1. A recoloring example ($\chi(G)=\omega(G)=2$). (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

For the graph depicted in Fig. 1 (left), the corresponding MaxSAT encoding ϕ is:

- Hard clauses: $\overline{x_1} \vee \overline{x_2}$, $\overline{x_1} \vee \overline{x_3}$, $\overline{x_1} \vee \overline{x_4}$, $\overline{x_2} \vee \overline{x_5}$, $\overline{x_3} \vee \overline{x_4}$, $\overline{x_4} \vee \overline{x_5}$.
- Soft clauses: $x_1 \vee x_2$ (in green), $x_3 \vee x_4$ (in cyan), x_5 (in yellow).

The objective is to find an assignment of values 1 (for true) and 0 (for false) to variables x_i such that all hard clauses and the maximum number of soft clauses are satisfied (equal to 1). The hard clauses oblige that at most one variable in every soft clause can be satisfied (only one vertex from every independent set can be in a (maximum) clique).

Hereafter we will call a color *unique* if only one vertex is colored in it. Fig. 2 provides an example over a 5-cycle graph G given an optimal coloring. It shows graph G partitioned into $\chi(G)=3$ colors (independent sets). We want to check if there is a maximum clique with one vertex in each color set. The figure also shows the MaxSAT encoding.

From the graph perspective, the process of finding the maximum clique starts by picking vertex 5 with a unique color and removing non-adjacent vertices 2, 3 from the remaining two color sets. In the next iteration, vertex 1 is selected because it has now a unique color (green) and non-adjacent vertex 4 is removed thus reducing the last color set (cyan) to the empty set. In the MaxSAT

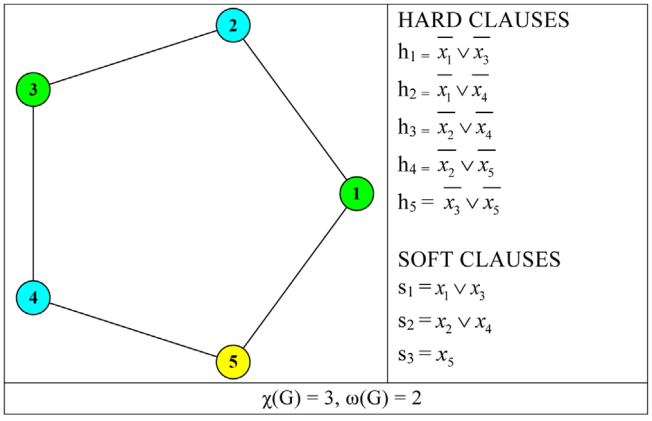

Fig. 2. An example of a 5-cycle graph.

domain, it implies that the subset of all soft clauses cannot be satisfied simultaneously, or no clique with one vertex in each color set (green, cyan and yellow) can exist, therefore $\omega(G) \leq \chi(G)-1$ (the infra-chromatic bound is $\chi(G)-1=2$). This example shows that the new bound can be less than the chromatic number (thus the name chosen in the paper). A subset of soft clauses which cannot be satisfied simultaneously is called an *inconsistent subset of soft clauses*. In general the new infra-chromatic bound of a graph G is equal to $k-s$ from proposition 3.

Proposition 3. [17]. *If a graph G can be partitioned into k independent sets, and there are s disjoint inconsistent subsets of soft clauses in the independent set based MaxSAT encoding, then* $\omega(G) \leq k-s$.

Encoding at every node the graph subproblem to MaxSAT and computing the number of disjoint inconsistent subsets, as suggested in [17] also introduces overhead. The remaining part of the paper describes (and evaluates) a new procedure to obtain inconsistent subsets *in the original graph notation*.

6. The new algorithm

The proposed new algorithm DO_INFRA-CHROMATIC substitutes procedure DO in the class coloring framework. After the first k_{min} color sets are obtained we consider every uncolored vertex v and create a separate color set for it. Then DO_INFRA-CHROMATIC is called for vertex v to search for two color sets (soft clauses in MaxSAT notation) *below* k_{min}, inconsistent with the color set containing only this vertex v. If the attempt succeeds these two color sets are tagged as *forbidden* and we prohibit adding other vertices to the color set containing only vertex v. Thus all these three colors do not take part in next iterations of this procedure, because the inconsistent sets of colors we search for should be disjoint (proposition 3). Otherwise, if the attempt fails, we color the vertex v according to class coloring SEQ.

In contrast to the MaxSAT upper bound, which also detects disjoint inconsistent subsets of more than three colors, we search for disjoint inconsistent subsets of *only three colors*: the unique color of the considered vertex v and the two colors inconsistent with it. Moreover, these two colors are searched only in the first $k_{min}-1$ colors and the first of these two colors should have only one vertex adjacent to vertex v. Not only does it reduce the overhead, but also minimize the rate of growth of the list of colors tagged as forbidden.

The following definitions are used:

- C_{k_1}, C_{k_2}: two color sets of vertices with color numbers k_1 and k_2 respectively, both below threshold k_{min};
- C_k^v: a color set of just one vertex v with color number k;
- an ordered color set tuple $(C_{k_1}, C_{k_2}, \ldots, C_{k_m})$: a tuple of color sets with color numbers in strict ascending order $(k_1 < k_2 < \cdots < k_m)$.

Listing 4 describes the new procedure. The outline is similar to recoloring in listing 3 but the filtering process is completely different. There are two loops that enumerate every pair of color sets below k_{min} to see if they are inconsistent with $C_k^v = \{v\}$. The inner loop is unrolled in two (steps 4–11, and steps 12–19) so as to check inconsistency first in ordered tuples which are more probable.

The inconsistency condition now takes into account the neighbor set of v (steps 6 or 14). It has a very natural interpretation:

Proposition 4. *If there is no vertex in color set* C_{k_2} adjacent to both vertex $w \in C_{k_1}$ and vertex v, then the tuple $(C_{k_1}, C_{k_2}, C_k^v)$ is an inconsistent subset.

Proof. By contradiction. The only possible 2-clique with vertices in C_k^v and C_{k_1} is $\{v, w\}$ since $|C_{k_1} \cap N(v)| = 1$ (step 3) and $C_k^v = \{v\}$, therefore any 3-clique has to be adjacent to both. If this condition is not met then a 3-clique cannot exist and the 3 color sets have to be inconsistent.□

Listing 4. The new infra-chromatic pruning procedure

```
DO_INFRA-CHROMATIC(v,k_min,C_1, C_2, ···, C_{k_min-1}, C_k,F)   //F is the list of forbidden colors
1. for k_1 := 1 to k_min-1
2.   if k_1 ∈ F then goto step 1                                 //color k_1 inconsistent in the previous call
3.   if |C_{k_1} ∩ N(v)| = 1 then w ← C_{k_1} ∩ N(v)
4.       for k_2 := k_1+1 to k_min -1
5.           if k_2 ∈ F then goto step 4                        //color k_2 inconsistent in the previous call
6.           if C_{k_2} ∩ N(w) ∩ N(v) = φ then                  //infra-chromatic pruning condition
7.             F ← F ∪ {k_1} ∪ {k_2}                            //tags the colors as forbidden
8.             C_k ← C_k\{v}                                    //removes v as candidate vertex
9.             return SUCCESS
10.          endif
11.      endfor
12.      for k_2 := 1 to k_1-1
13.          if k_2 ∈ F then goto step 12
14.          if C_{k_2} ∩ N(w) ∩ N(v) = φ then
15.            F ← F ∪ {k_1} ∪ {k_2}
16.            C_k ← C_k\{v}
17.            return SUCCESS
18.          endif
19.      endfor
20.  endif
21.  endfor
22.return FAIL
```

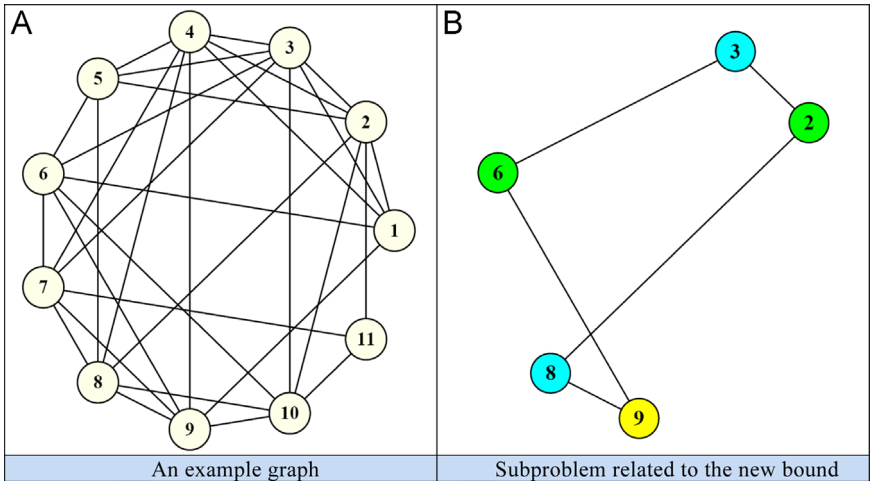

Fig. 3. An example of a graph for a demonstration of the new algorithm.

Table 1
The main steps of the new algorithm. The starred cell marks the step in which the new infra-chromatic pruning is effective.

Clique S	Candidates U	Colors $c(v)$	Comments
	1 2 3 4 5 6 7 8 9 10 11	1 2 3 4 5 6 7 8 8 8 8	
11	2 7 10	1 1 2	Vertex 11 has neighbors 2, 7, 10.
11 10	2	1	Vertex 10 has neighbor 2.
11 10 2			Found a maximal clique, S_{max}={11, 10, 2}
11	2 7	1 1	All branches are pruned since $\|S\|+c(v) \le \|S_{max}\|$ $(1+1 \le 3)$.
	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 8 8	
10	2 3 6 8 9	1 2 1 2 *	Vertex 10 has neighbors 2, 3, 6, 8, 9. **Infra-chromatic pruning condition holds for vertex 9** All branches are pruned since $1+2 \le 3$
	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 8	
9	1 4 6 7 8	1 2 2 1 3	Vertex 9 has neighbors 1, 4, 6, 7, 8. Infra-chromatic pruning condition is ruled out for vertex 8 because vertices 4 and 7 are adjacent.
9 8	4 7	1 2	Vertex 8 has neighbors 4, 7
9 8 7	4	1	Vertex 7 has neighbor 4.
9 8 7 4			Found a maximal clique, S_{max}={9, 8, 7, 4}.
9 8	4	1	This branch is pruned since $2+1 \le 4$.
9	1 4 6 7	1 2 2 1	All branches are pruned since $1+2 \le 4$.
	1 2 3 4 5 6 7 8	1 2 3 4 5 6 7 8	
8	2 4 5 7	1 2 3 1	Vertex 8 has neighbors 2, 4, 5, 7. All branches are pruned since $1+3 \le 4$.
	1 2 3 4 5 6 7	1 2 3 4 5 6 7	
7	3 4 6	1 2 2	Vertex 7 has neighbors 3, 4, 6. All branches are pruned since $1+2 \le 4$.
	1 2 3 4 5 6	1 2 3 4 5 6	
6	1 3 5	1 2 1	Vertex 6 has neighbors 1, 3, 5. All branches are pruned since $1+2 \le 4$.
	1 2 3 4 5	1 2 3 4 5	
5	2 3 4	1 2 3	Vertex 5 has neighbors 2, 3, 4. All branches are pruned since $1+3 \le 4$.
	1 2 3 4	1 2 3 4	All branches are pruned since $0+4 \le 4$.

Once a color gets into an inconsistent subset, it is tagged and added to the forbidden colors list F (steps 7 or 15). Steps 2, 5 and 13 check that the enumeration of color sets do not include any member of F in successive calls. If an inconsistent subset of colors is not found, the procedure returns FAIL (step 22). Otherwise, the procedure returns SUCCESS (steps 9 or 17). Vertex v is then removed from the color set where it has been originally placed by the framework (step 8 or 16), so that the main algorithm will not branch on it. It is important to note that we do not need to branch on it, because we can always change the order of color sets to $\{C_1, C_2, \cdots, C_{k_{min}-1}, C_k{}^v, C_{k_{min}}, \ldots\}$ and compute the upper bound for a graph $G' = G[C_1 \cup C_2 \cup \cdots \cup C_{k_{min}-1} \cup C_k{}^v]$. The upper bound for a graph G'equals to $k_{min}-1$ (proposition 3) that is why the pruning condition (2) is valid for vertex v.

DO_INFRA-CHROMATIC integrates well with the proposed class coloring framework, but it would be much more awkward with classical SEQ. In the latter case, a call to DO_INFRA-CHROMATIC when SEQ assigns a new color number to a vertex greater than or equal to k_{min}, would reason with just *partial colorings of the color sets* below k_{min} since remaining uncolored vertices could still be assigned there. Tagging a color set and later changing its contents makes the procedure incorrect. In the class coloring framework, every color set below k_{min} (susceptible to be tagged) is computed in full prior to any call to DO_INFRA-CHROMATIC.

It is worth noting that inconsistency between color sets is not symmetrical, i.e, the tuple $(C_{k_1}, C_{k_2}, C_k^v)$ might be consistent but $(C_{k_2}, C_{k_1}, C_k^v)$ might not. In listing 4, as mentioned previously, the search develops in two stages. First ordered color set pairs are checked (steps

Table 2
Comparative performance of infra-chromatic pruning over structured instances. The best times and minimum number of steps (recursive procedure calls) for each row are written in bold. The time limit was fixed at 24 h.

Name					*dfmax* [23]	BBMCI [13]		BBMCR [13]		BBMCX_O		BBMCX		MaxCLQ [18]	
	n	p	ω	ω_0	time (s)	steps	time (s)	steps	time (s)	steps	time (s)	steps	time (s)	steps	time (s)
brock200_1	200	0.75	21	16	9.92	93,134	0.249	52,363	0.276	40,010	0.252	**34,472**	**0.240**	38,887	0.447
brock200_2	200	0.50	12	8	0.019	1034	0.016	630	**0.007**	462	0.008	**372**	0.008	1265	0.411
brock200_3	200	0.61	15	9	0.146	3322	0.009	2,178	0.014	1784	0.018	**1,490**	0.018	2481	0.501
brock200_4	200	0.66	17	13	0.620	19,329	0.047	12,206	0.058	6634	0.046	**5,541**	**0.044**	11,534	0.544
brock400_1	400	0.75	27	19	fail	46,984,162	264.8	26,204,657	253.3	18,564,480	197.7	**15,650,405**	**194.7**	19,991,548	267.1
brock400_2	400	0.75	29	18	8300.5	17,854,215	112.7	10,071,509	106.5	7,785,132	89.23	**6,321,421**	**87.94**	8,395,282	124.9
brock400_3	400	0.75	31	17	9406.3	34,034,323	177.4	19,162,515	173.2	16,317,420	156.0	**13,650,086**	**155.5**	16,632,868	208.6
brock400_4	400	0.75	33	18	6618.1	18,456,530	105.6	11,084,744	107.7	8,127,835	**87.00**	**6,954,205**	88.87	9,875,411	129.7
brock800_1	800	0.65	23	14		423,852,560	4762.5	253,929,268	4796.5	193,366,098	3720.0	**153,169,111**	**3496.9**	386,532,897	5930.1
brock800_2	800	0.65	24	15		385,559,845	4404.8	230,492,690	4231.7	174,818,047	3332.7	**140,594,271**	**3140.1**	315,456,602	5202.8
brock800_3	800	0.65	25	14		233,969,561	3315.1	144,250,320	2725.3	107,958,807	2298.9	**87,863,265**	**2063.7**	191,276,571	3263.6
brock800_4	800	0.65	26	15		148,615,536	1990.9	94,335,513	1976.4	69,892,586	1494.5	**56,675,286**	**1429.8**	134,517,995	2408.9
phat300-1	300	0.24	8	7		398	0.002	298	0.002	255	**0.002**	**36**	0.003	424	0.049
phat300-2	300	0.49	25	23		1245	0.008	727	0.008	519	**0.007**	**470**	0.008	1041	0.049
phat300-3	300	0.74	36	31	480.8	176,741	0.918	99,856	0.984	70,279	**0.809**	**64,414**	0.863	96,544	1.51
phat500-1	500	0.25	9	6		1849	0.010	1009	0.012	832	0.011	**654**	0.012	4387	0.196
phat500-2	500	0.51	36	29	80.61	30,947	0.299	17,686	0.322	12,620	**0.249**	**11,301**	0.250	24,680	0.646
phat500-3	500	0.75	50	41	fail	5,052,787	59.70	2,554,363	57.75	1,713,729	43.59	**1,562,368**	**43.32**	2,308,431	51.53
phat700-1	700	0.25	11	8	0.133	6341	0.048	3307	0.048	2635	0.042	**1833**	**0.041**	15,614	3.92
phat700-2	700	0.50	44	38	3013.4	176,610	2.80	98,306	2.57	68,557	2.00	**62,066**	1.99	152,541	4.69
phat700-3	700	0.75	62	55		58,764,014	1343.6	29,882,680	1159.6	19,490,446	**837.4**	**17,733,598**	883.9	39,268,710	1142.6
phat1000-1	1000	0.25	10	7	0.660	36,002	0.263	28,865	0.295	22,699	0.254	**19,018**	**0.249**	50,308	2.08
phat1000-2	1000	0.49	46	39		6,191,494	145.8	3,366,459	124.1	2,451,788	97.17	**2,198,710**	**95.17**	4,721,336	131.5
phat1500-1	1500	0.25	12	8	6.10	143,796	2.60	128,010	2.75	102,007	2.77	**92,264**	**2.53**	341,308	13.02
phat1500-2	1500	0.51	65	54		375,365,052	10775.1	188,146,229	9637.6	117,483,284	5061.0	**104,443,234**	**5007.1**	242,104,729	10434.8
hamming8-4	256	0.64	16	16	1.27	5780	0.015	3212	**0.013**	3234	0.014	2763	0.014	**2,250**	0.050
johnson16-2-4	120	0.77	8	8	0.703	126,354	0.048	125,163	0.064	118,238	0.065	106,538	0.062	**72,345**	0.629
keller4	171	0.65	11	8	0.299	3036	0.011	2261	0.012	2135	**0.010**	1903	0.011	**1569**	0.017
keller5	776	0.75	27	16		5,780,745,261	50474.0	3,558,896,980	37677.1	3,552,052,314	41007.1	3,186,668,048	39389.1	**267,489,494**	**6477.2**
MANN_a27	378	0.99	126	125		18,309	0.248	4679	**0.157**	4679	0.178	4679	0.227	**2475**	0.207
MANN_a45	1035	1.00	345	342		1,081,028	112.9	118,384	**26.85**	118,384	31.25	118,384	41.64	**77,218**	29.51
san200_0.7_1	200	0.70	30	17		710	0.011	240	0.02	326	**0.011**	285	0.015	**107**	0.012
san200_0.7_2	200	0.70	18	12		339	0.008	239	0.014	229	**0.007**	**205**	0.012	294	0.025
san200_0.9_1	200	0.90	70	45	fail	2643	0.028	439	0.036	331	0.035	**291**	0.050	440	**0.017**
san200_0.9_2	200	0.90	60	38	fail	6235	0.066	37,045	0.361	3354	0.088	**3003**	0.095	8044	0.132
san200_0.9_3	200	0.90	44	31		4784	0.039	3555	0.050	1642	0.052	**1524**	0.052	12,583	0.232
san400_0.7_1	400	0.70	40	21	fail	18,098	0.187	7184	**0.137**	9290	0.181	7829	0.187	**4872**	0.166
san400_0.7_2	400	0.70	30	15	fail	4370	0.083	2237	**0.060**	3600	0.099	4219	0.120	**778**	0.083
san400_0.7_3	400	0.70	22	13		40,928	0.399	44,134	0.509	34,962	0.484	**34,041**	0.508	40,461	0.563
san400_0.9_1	400	0.90	100	42		6291	0.249	654,939	7.85	1419	**0.238**	**1271**	0.248	23,734	1.034
san1000	1000	0.50	15	8	fail	8593	0.395	**5,531**	**0.312**	15,907	0.890	15,831	0.887	23,433	0.943
sanr200_0.7	200	0.70	18	14	2.12	40,051	0.097	24,203	0.115	18,916	0.102	**15,823**	0.102	22,604	0.223
sanr200_0.9	200	0.90	42	36		3,496,533	13.17	1,373,072	10.91	892,258	8.53	852,648	9.59	**388,858**	**5.47**
sanr400_0.5	400	0.50	13	9	1.46	60,335	0.237	38,751	0.263	31,096	0.238	**24,993**	0.242	76,786	0.894
sanr400_0.7	400	0.70	21	16	1647.8	15,226,278	71.50	9,031,197	70.27	6,769,696	63.52	**5,666,958**	**57.82**	8,632,873	105.6
gen200_p0.9_44	200	0.90	44	33	fail	56,256	0.276	31,290	0.249	16,960	0.199	15,347	0.202	**9609**	**0.149**
gen200_p0.9_55	200	0.90	55	37	5454.9	109,474	0.426	67,532	0.484	39,104	0.362	36,668	0.405	**9049**	**0.185**
gen400_p0.9_55	400	0.90	55	44		3,754,881,677	19611.2	1,820,376,757	25406.1	1,781,334,874	23972.5	**1,606,799,254**	25170.0	n.a.	fail
gen400_p0.9_65	400	0.90	65	41		8,070,216,132	41499.1	7,292,824,361	**28063.6**	6,699,950,733	81634.0	5,652,743,855	85975.3	**2,183,044,625**	47688.7
gen400_p0.9_75	400	0.90	75	43		15,339,445,936	fail	6,911,492,825	53986.1	6,098,293,591	57179.5	4,993,840,316	61329.3	**583,227,541**	**12979.5**
C125.9	125	0.90	34	31	31.96	11,022	0.033	4705	**0.016**	3107	0.026	3023	0.029	**1462**	0.744
C250.9	250	0.90	44	37	fail	309,822,827	1244.4	119,959,129	992.1	83,669,702	839.6	79,769,017	927.4	**22,810,678**	**348.7**
frb30-15-1	450	0.82	30	26		256,131,721	2048.5	133,821,400	1544.0	121,293,008	1469.7	100,334,507	1338.1	**35,050,106**	**728.4**

Table 2 (*continued*)

Name	n	p	ω	ω_0	*dfmax* [23]	BBMCI [13]		BBMCR [13]		BBMCX_O		BBMCX		MaxCLQ [18]	
					time (s)	steps	time (s)	steps	time (s)	steps	time (s)	steps	time (s)	steps	time (s)
frb30-15-2	450	0.82	30	24		159,314,124	1268.6	93,885,809	1062.1	78,045,483	945.1	65,319,073	**873.4**	**46,951,096**	985.0
frb30-15-3	450	0.82	30	24		99,025,361	761.6	46,641,628	500.7	42,757,624	514.4	35,482,801	**455.4**	**29,798,336**	620.3
frb30-15-4	450	0.82	30	22		302,448,833	2233.8	147,091,590	1648.4	134,831,596	1605.4	110,724,419	1504.2	**60,602,085**	**1241.9**
frb30-15-5	450	0.82	30	22		207,658,092	1621.6	97,797,628	1094.0	91,682,954	1075.1	75,286,301	1041.1	**44,136,994**	**916.8**
dsjc500.5	500	0.50	13	9		262,157	1.09	177,622	1.27	140,795	1.08	**113,810**	**1.05**	296,250	3.69
dsjc1000.5	1000	0.50	15	10		15,251,603	152.2	9,576,727	168.6	7,653,039	137.9	**6,067,730**	**132.0**	23,053,488	344.7

4–11) and later unordered ones (steps 12–19). In Section 7 (the experiments section) performance results for a DO_INFRA-CHROMATIC variant that only checks ordered tuples is also reported.

The main steps of the new algorithm for an example graph (Fig. 3A) are demonstrated in Table 1. The first column (header *Clique S*) is the currently growing clique and is empty for rows which refer to the root node. The maximum degree of the graph is equal to 7, i.e. $\Delta G = 7$. So the initial coloring (listing 1) will be 1, 2, 3, 4, 5, 6, 7, 8, 8, 8, 8 for the vertices 1, 2,…, 11 respectively. First we add the vertex 11 to the current clique S (listing 1, step 5), remove it from U (listing 1, step 4) and find all candidate vertices which are connected with vertex 11 (listing 1, step 6). In our example vertex 11 has neighbors 2, 7, 10 (second row in the table). Then we use class coloring (listing 1, step 12) for the new candidate vertices. The result of class coloring is that colors are 1, 1, 2 for vertices 2, 7, 10 respectively. After it we add vertex 10 to the current clique S and repeat the above-described actions (third row). The list of candidate vertices U is empty when the current clique S consists of vertices 11, 10, 2. It means that S is the maximal clique, because it cannot be enlarged by any other vertex in the graph. We copy S to the best clique *Smax* found so far (listing 1, step 8) and return to the previous level where U is not empty. For $S=\{11\}$ and $U=\{2, 7\}$ (fifth row) all branches are pruned because the condition (2) is satisfied (listing 1, step 3).

After it we add the vertex 10 to empty S (seventh, and critical, row). We color vertices 2, 3, 6, 8, 9 (Fig. 3B). In this case $k_{min}=3$ (listing 1, step 11) thus we use DO_INFRA-CHROMATIC only for vertex 9 (listing 2, step 6). Before using of DO_INFRA-CHROMATIC there are color sets $C_1=\{2, 6\}$ (green) and $C_2=\{3, 8\}$ (cyan). The first color set contains only one vertex 6 which is connected to vertex 9, that is why the condition from step 3 (listing 4) is satisfied. The second color set contains only one vertex 8 which is connected to vertex 9. Vertices 6, 8 are not neighbors. Thus the infra-chromatic pruning condition (listing 4, step 6) is satisfied and the tuple ({2, 6}, {3, 8}, {9}) is an inconsistent subset. As a result, for $S=\{10\}$ and $U=\{2, 3, 6, 8, 9\}$ all branches are pruned (listing 1, step 3). The remaining part of the search is included in the table for completeness.

6.1. Implementation details

The new infra-chromatic procedure has been incorporated in bit parallel BBMC. The main reason is that BBMC already integrates class coloring. Moreover in [20,24] BBMC is reported to be *one of the fastest* algorithms at present together with MCS and MaxCLQ.

In our implementation of DO_INFRA-CHROMATIC, color sets and neighbor sets are all bit strings in which a bit maps to a particular vertex. The forbidden color list F is encoded in another bit string, which maps this time bits to color numbers. The semantics of a 1-bit in F is that the corresponding color set has made part of an inconsistent subset. Details related to the general bit encoding can be found in [12–13].

7. Experiments

The main contribution of this paper, in the opinion of the authors, is that it describes for the first time an efficient way to prune the search space in approximate-color-based algorithms obtaining a bound possibly below the chromatic number of the subproblem at each node. The aim of the tests presented here is to validate the approach.

7.1. Algorithms

For evaluation the following algorithms have been considered:

a) BBMC: the algorithm used as reference for exact MCP. It employs bit strings to encode the domain and implements

Table 3
Comparison of BBMCX and MaxCLQ algorithms on uniform random graphs. The best times and minimum number of steps (recursive procedure calls) for each row are written in bold.

				BBMCX		MaxCLQ[18]	
n	***p***	ω	ω_0	**steps**	**time (s)**	**steps**	**time (s)**
150	0.7	16–17	14.1	**3104**	**0.011**	3794	0.036
150	0.8	23	20	12,163	**0.055**	**9701**	0.093
150	0.9	35–38	31.8	42,015	0.335	**14,907**	**0.170**
150	0.95	51–58	49.2	8962	0.141	**1545**	**0.027**
150	0.98	75–84	76.8	221	**0.004**	**203**	0.010
200	0.7	17–18	13.2	**18,207**	**0.111**	25,683	0.230
200	0.8	25–26	20.1	**174,195**	**1.23**	149,587	1.40
200	0.9	40–42	34.3	1,801,004	19.2	**637,801**	**8.07**
200	0.95	58–66	54.1	2,452,652	39.2	**183,973**	**3.22**
200	0.98	90–103	89	7,988	0.261	**583**	**0.025**
300	0.6	15–16	11.4	**40,573**	**0.268**	79,908	0.753
300	0.7	20–21	14.6	**434,737**	**3.38**	656,847	6.10
300	0.8	28–29	21	12,416,791	116	**10,694,884**	**113**
500	0.4	10–11	7.5	**12,929**	**0.117**	35,332	0.599
500	0.5	13	9.7	**83,514**	**0.837**	256,876	2.72
500	0.6	17	12	**1,188,261**	**13.1**	2,695,081	28.7
500	0.7	22–23	16.3	**47,259,958**	**550**	68,702,580	733
500	0.994	266–271	257.2	88,283	49.7	**2807**	**0.375**
1000	0.2	7–8	6.1	1340	**0.054**	**1166**	0.788
1000	0.3	9–10	6.5	**31,924**	**0.490**	106,262	2.64
1000	0.4	12	8	**280,309**	**5.87**	1,466,971	18.9
1000	0.5	15	10.3	**6,057,372**	**134**	23,010,641	300
3000	0.1	6–7	4.3	**2491**	**0.246**	98,463	19.7
3000	0.2	9	5.8	**196,614**	**5.64**	823,163	44.7
5000	0.1	7	4.4	**6399**	**1.55**	438,717	88.7
5000	0.2	9–10	6.1	**1,147,990**	**89.0**	16,912,464	752
10000	0.1	7–8	5	**368,101**	**26.2**	2,914,178	889
15000	0.1	8	5.1	**1,517,329**	**149**	8,187,418	3574

class coloring. Our implementation of new BBMCX is based on BBMC (in particular we include the improvements suggested in BBMCI [13]).

b) BBMCR: similar to a) but includes a recoloring scheme [13].
c) BBMCX: the default new algorithm which adapts DO_INFRACHROMATIC (as described in Section 6) to BBMCI.
d) BBMCX_O: similar to BBMCX but only checks for consistency in color set tuples which preserve ascending color order. This variant aims at a good compromise between efficient pruning and computational effort.
e) MaxCLQ: the algorithm described by Li and Quan in [18] which improves the algorithm from [17]. The implementation of MaxCLQ was kindly provided by Li and Quan.

All algorithms have been implemented in C/C++ (VC 2010 compiler) and optimized using a native code profiler. The machine employed for the experiments was an Intel(R) Core(TM) i7 CPU 950 @ 3.07 GHz with a 64-bit Win7 O.S. and 6 GB of RAM, the same as in the original BBMCI report. We note that a new version of BBMCI has recently been released [25], so times may slightly differ from those found in the original paper.

In all experiments, the time limit was set to 24 h and only time spent searching the graph is recorded. An initial greedy lower bound for $\omega(G)$ has been precomputed in all cases and assigned to $|S_{max}|$ at the root node (header ω_0 in the table report). This allows a better comparison between variants by removing noise from early branching. The time for this precomputation is never greater than 1 ms. For finding the initial lower bound we use the following scheme:

1. Initialize a list of candidates: $U \leftarrow V$.
2. Add vertex $v \in U$ with the highest degree in G to S_{max}.
3. Update the list of candidates: $U \leftarrow U \cap N(v)$.
4. Repeat steps 2 and 3 until the list of candidates is empty.

The report also includes performance of algorithm *dfmax* [26], commonly used for calibration, over a subset of instances. Times of *dfmax* over DIMACS machine benchmark graphs *r300.5*, *r400.5* and *r500.5* were, for our machine, 0.189, 1.155 and 4.369 s respectively.

Finally, it is important to highlight that none of the algorithms reported employs any form of multicore parallelism. Bit optimization techniques are implemented in a single thread.

7.2. Data sets

The algorithms have been evaluated over structured data sets and BBMCX and MaxCLQ algorithms have been evaluated over uniform random graphs for a more thorough analysis. The majority of the structured data sets are taken from the popular DIMACS benchmark (presented at the Second DIMACS Implementation Challenge[1]), and a small subset belongs to the BHOSHLIB benchmark[2] (in particular the *frb30-15* collection). Some instances which were too easy for all the algorithms (e.g. the *cfat* family), as well as those which were too hard for the time limit of 24 h (i.e. *C.500.9*, *MANN_a81*, other instances from BHOSHLIB etc.) have been left out of the report.

7.3. Results

Table 2 reports performance and tree size of infra-chromatic pruning over the instances considered. Column headers n and p stand for the number of vertices and the density respectively. ω is the maximum clique and ω_0 is our greedy initial lower bound. The best times and minimum number of steps for each line in the table are in bold.

[1] http://cs.hbg.psu.edu/txn131/clique.html.
[2] http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm.

From the perspective of tree size, BBMCX shows an average 54% reduction compared to BBMCI in all but four cases. Recoloring is known to prune the search space especially in more difficult, dense, graphs, but it is limited by the chromatic number of the graph subproblems. Thus BBMCX also visits less nodes than BBMCR in the vast majority of cases (all except 6) averaging a 23% reduction on tree size when this is the case.

Regarding performance, BBMCX is faster than BBMCI in many instances (43 out of 58) and shows some of the best times, to the best of our knowledge, reported in literature for a stand-alone computer (e.g. the *dsjc* family, *brock800_1* and others). To prove this, some calibrated times for leading MCS solver based on the report in Table 3 of [15] are enumerated. Based on that report, a time ratio of 1.92 is inferred in favor of our machine comparing *dfmax* performance over the corresponding machine instances. MCS corrected times for *phat700_3*, *brock800_1* and *C250.9* are 1247.7, 4875.4 and 1689.8 s respectively whereas BBMCX takes 883.9, 3496.9 and 927.4. A similar analysis can be made for other algorithms.

The majority of instances in which BBMCX performs worse than the reference algorithm BBMCI belong to the *san* family. Specifically, BBMCX performs extremely poor in *san 1000* where even the size of the search tree is bigger than its counterpart. No explanation for this anomaly can be given at present. In the remaining cases outside *san* (*brock200_3*, *johnson16-2-4*, *sanr200_0.7* and *sanr400_0.5*) either the instances are fairly easy or times are comparable. Regarding BBMCR, the behavior is similar except for the very dense MANN family, *gen400_p0.9_65* and *gen400_p0.9_75*, in which recoloring outperforms BBMCX.

BBMCX_O shows that the extra effort involved in searching for inconsistent *unordered* color set tuples in BBMCX is approximately balanced by the reduction in tree size. BBMCX_O performs comparably to reference BBMCX, the difference in time is not greater than 15% in favor of one or the other in the vast majority of cases. Exceptionally, BBMCX_O significantly outperforms BBMCX in *san200_0.7_2* as well as both instances of the MANN family. The first case turns out to be very easy for both algorithms so it has minor relevance. However the performance in MANN would seem to indicate that, for very dense graphs, it is better to consider only pairs of color subsets in strict ascending order.

BBMCX shows an average 69% increase of the tree size compared to MaxCLQ but for 36 instances the tree size of BBMCX is smaller than the tree size of MaxCLQ. BBMCX performs faster than MaxCLQ in 42 instances, although for *keller5*, *gen400_p0.9_75*, *C250.9* MaxCLQ gives 5, 4, 2 times speedup respectively. It is worth noting that Table 2 does not give us clear understanding which of the two algorithms is better. That's why we have done additional tests to compare BBMCX and MaxCLQ.

Table 3 reports average times of BBMCX and MaxCLQ algorithms over 10 uniform random graphs with fixed n and p, where n and p denote the number of vertices and the probability that an edge between a pair of vertices exists respectively. It shows that MaxCLQ performs faster than BBMCX only for dense graphs ($p \geq 0.9$) and comparably for $p = 0.8$. MaxCLQ seems more effective in this case because MaxSAT bound is tighter than infra-chromatic bound. So MaxCLQ prunes more vertices than BBMCX and for the denser graphs it proves to be better to spend more time on calculating a better bound. Another possible explanation for MaxCLQ's performance is that it branches on vertices with low degree (not maximum color like BBMCX) and this may prove to be a better local minimization heuristic for such graphs. For densities below 0.8, BBMCX clearly outperforms its counterpart.

8. Concluding remarks and future work

This paper proposes a new maximum clique branch-and-bound algorithm that is able to prune the search space with a new infra-chromatic upper bound at each step of the search. The relevance of this result is that it is more effective to use infra-chromatic bound (limited MaxSAT bound) for graphs with the density less than 0.8.

The paper covers experiments over standard DIMACS and BHOSHLIB benchmarks that show the importance of the contribution. The pruning ability of the new procedure is unquestionable. Moreover, times reported for a number of families considered (e.g. *brock*, *dsjc*) would seem to be the best in literature for a stand-alone computer to the best of our knowledge.

Future work will include comparison with other algorithms, such as an improved version of MaxCLQ, denoted as incMaxCLQ, which has been very recently reported in [23]. Another line of research at present is to combine infra-chromatic pruning with other known techniques such as recoloring.

Releases for the new algorithms BBMCX and BBMCX_O are now publicly available [25] (compiled binaries work under Win7 64-bit OS). The bit string framework as well as the implementation of BBMCI may be found in Biicode (www.biicode.com) repository [27].

Acknowledgments

Pablo San Segundo is funded by the Spanish Ministry of Economy and Competitiveness (ARABOT: DPI 2010-21247-C02-01) and supervised by CACSA. Alexey Nikolaev and Mikhail Batsyn are supported by Russian Federation Grant 14-41-00039. The authors would like to thank Chu-Min Li and Zhe Quan for the source code of their MaxCLQ algorithm.

References

[1] Bomze I, Budinich M, Pardalos P, Pelillo M. The maximum clique problem. Handb Comb Optim 1999;4:1–74.
[2] Butenko S, Wilhelm WE. Clique-detection models in computational biochemistry and genomics. Eur J Oper Res 2006;173:1–17.
[3] San Segundo P, Rodriguez-Losada D, Matia F, Galan R. Fast exact feature based data correspondence search with an efficient bit-parallel MCP solver. Appl Intell 2010;32(3):311–29.
[4] San Segundo P, Rodriguez-Losada D. Robust global feature based data association with a sparse bit optimized maximum clique algorithm. Robot IEEE Trans 2013;99:1–7.
[5] Hotta K, Tomita E, Takahashi H. A view invariant human FACE detection method based on maximum cliques. IPSJ Trans Math Model Appl 2003:57–70.
[6] Bron C, Kerbosch J. Algorithm 457: finding all cliques of an undirected graph. Commun ACM 1973;16(9):575–7.
[7] Carraghan R, Pardalos P. An exact algorithm for the maximum clique problem. Oper Res Lett 1990;9(6):375–82.
[8] Fahle, T. Simple and fast: improving a branch-and-bound algorithm for maximum clique. In: Proceedings of the European symposium on algorithms 2002. LNCS 2461; 2002. p. 485–498.
[9] Régin, J-C. Using constraint programming to solve the maximum clique problem. In: Proceedings of the principles and practice of constraint programming. LNCS 2833; 2003. p. 634–648.
[10] Konc J, Janečič D. An improved branch and bound algorithm for the maximum clique problem. MATCH Commun Math Comput Chem 2007;58:569–90.
[11] Östergård PRJ. A fast algorithm for the maximum clique problem. Discret Appl Math 2002;120(1):97–207.
[12] San Segundo P, Rodriguez-Losada D, Jimenez A. An exact bit-parallel algorithm for the maximum clique problem. Comput Oper Res 2011;38(2):571–81.
[13] San Segundo P, Matia F, Rodriguez-Losada D, Hernando M. An improved bit parallel exact maximum clique algorithm. Optim Lett, 7; 2013. p. 467–79.
[14] Etsuji Tomita, Tomokazu Seki. An efficient branch-and-bound algorithm for finding a maximum clique. In: Cristian S Calude, Michael J Dinneen, Vincent Vajnovszki, editors. Proceedings of the 4th international conference on discrete mathematics and theoretical computer science (DMTCS'03), Berlin, Heidelberg: Springer-Verlag; 2003. p. 278–89.
[15] Tomita E, Sutani Y, Higashi T, Takahashi S, Wakatsuki M. A simple and faster branch-and-bound algorithm for finding a maximum clique. In Lecture notes in computer science, 5942. Edited by Rahman MS, Fujita S.; 2010. p. 191–203.
[16] Wood DR. An algorithm for finding a maximum clique in a graph. Oper Res Lett 1997;21:211–7.
[17] Li, CM, Quan, Z. An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem. In: Proceedings of the24th AAAI Conference on AI, AAAI-10. 2010. p. 128–133.

[18] Li, CM, Quan, Z. Combining graph structure exploitation and propositional reasoning for the Maximum Clique Problem. In: Proceedings of the 2010 22th IEEE international conference on tools with artificial intelligence, IEEE. 2010. p. 344–351.
[19] Batsyn M, Goldengorin B, Maslov E, Pardalos P. Improvements to MCS algorithm for the maximum clique problem. J Comb Optim, 27; 2014. p. 397–416.
[20] Prosser P. Exact algorithms for maximum clique: a computational study. Algorithms 2012;5(4):545–87.
[21] Balas E, Yu CS. Finding a maximum clique in an arbitrary graph. SIAM J Comput 1986;15(4):1054–68.
[22] Mycielski J. Sur le coloriage des graphes. Colloq Math 1955;3:161–2.
[23] Li, CM, Fang, ZW, Xu, K. Combining MaxSAT reasoning and incremental upper bound for the maximum clique problem. In: Proceedings of the IEEE 25th international conference on tools with artificial intelligence. ICTAI; 2013. p. 939–946.
[24] Wu Q, Hao J-K. A review on algorithms for maximum clique problems. Eur J Oper Res 2015;242(3):693–709.
[25] ⟨https://www.dropbox.com/sh/i9jllk459z3lubf/XZA1tjMJ7H⟩.
[26] ⟨ftp://dimacs.rutgers.edu/pub/dsj/clique/dfmax.chttp://intelligentcontrol.es/arabot/sites/default/files/frontpage/bbmc1.0.zip⟩.
[27] ⟨https://www.biicode.com/pablodev/copt⟩.