Available online at www.sciencedirect.com

SCIENCE DIRECT®

Discrete Applied Mathematics 154 (2006) 826–847

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

# A Branch-and-Cut algorithm for graph coloring[☆]

Isabel Méndez-Díaz, Paula Zabala

*Departamento de Computación, FCEyN, Universidad de Buenos Aires, Argentina*

Received 4 February 2003; received in revised form 9 April 2004; accepted 31 May 2005
Available online 17 November 2005

**Abstract**

In this paper a Branch-and-Cut algorithm, based on a formulation previously introduced by us, is proposed for the Graph Coloring Problem. Since colors are indistinguishable in graph coloring, there may typically exist many different symmetrical colorings associated with a same number of colors. If solutions to an integer programming model of the problem exhibit that property, the Branch-and-Cut method tends to behave poorly even for small size graph coloring instances. Our model avoids, to certain extent, that bottleneck. Computational experience indicates that the results we obtain improve, in most cases, on those given by the well-known exact solution graph coloring algorithm Dsatur.
© 2005 Elsevier B.V. All rights reserved.

*Keywords:* Graph coloring; Integer programming; Branch-and-Cut algorithms

## 1. Introduction

Assume that an undirected graph $G=(V,E)$ is given with a set $V$ of vertices and a set $E$ of edges. A *coloring* of $G$ is an assignment of colors to the vertices in $V$ where different colors are assigned to endpoints of any edge in $E$. Accordingly, a *k-coloring* of $G$, where $k \geqslant 2$, is a coloring that uses exactly $k$ colors. The least possible value for $k$, denoted $\chi(G)$, is the *chromatic number* of $G$ and the graph coloring problem (*GCP*) is to determine $\chi(G)$.

*GCP* has been extensively studied in the graph theory literature [15]. Interest on the problem also arises from applications in scheduling, timetabling, electronic bandwidth allocation and sequencing [9,19,21].

*GCP* is known to be NP-hard for arbitrary graphs [18]. However, the practical importance of the problem makes it desirable to obtain solution algorithms capable of solving, in *acceptable* computational times, at least medium size *GCP* instances. Considerable effort has been devoted so far to derive these algorithms and most of the existing ones are based on heuristic techniques. Among existing *GCP* heuristics, the most commonly used strategy is to derive a coloring for a small subgraph of $G$ and then extend it, vertex by vertex, to the whole graph $G$ (increasing, if necessary, the number of colors involved). Metaheuristic techniques such as simulated annealing and tabu search have also been applied to *GCP* [11,16,13,14]. Very few exact solution algorithms exist for the problem. A comprehensive list of references on coloring algorithms can be found in [10].

[☆] This research was partially supported by UBACYT Grant X212 and PICT Grant 01109112.
*E-mail addresses:* imendez@dc.uba.ar (I. Méndez-Díaz), pzabala@dc.uba.ar (P. Zabala).

0166-218X/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2005.05.022

*GCP* is related to Maximum Clique and Maximum Stable Set problems. However, for similar dimensions, *GCP* appears more *difficult* to solve exactly than these problems (which are also known to be NP-hard). Typically, Maximum Clique and Maximum Stable Set instances with hundreds of vertices are solved to proven optimality in low CPU times [3,16,22,28]. Contrary to that, optimal solutions for some *GCP* instances, defined on graphs with as few as 70 vertices, remain unknown.

Like most optimization problems in graphs, *GCP* can be formulated as a linear integer programming problem. Based on these formulations, Branch-and-Cut algorithms are currently the best exact solution algorithms available for the problem [2,4,5]. However, in comparison with the Traveling Salesman Problem (TSP) or the Maximum Stable Set Problem, substantially less research effort has been devoted to Branch-and-Cut algorithms for *GCP*.

In [1], Aardal et al. propose a Branch-and-Cut algorithm for (a vertex packing model of) the frequency assignment problem. *GCP* can be seen as special case of that. Following a different approach, Mehrotra and Trick [24] developed a column generation algorithm for *GCP*. The algorithm is based on the classical independent set formulation of the problem.

The performance of a Branch-and-Cut algorithm is dependent on a combination of various factors. Among these, the most important ones are preprocessing, search and branching strategies, lower and upper bounds, LP-relaxation and the type of cutting planes used. In particular, cutting planes that take advantage of problem specific polyhedral structures have proven quite successful [3,28,30,32] and are used in this study.

Since colors in *GCP* are indistinguishable, many symmetrical colorings typically exist for a same given number of colors. If feasible solutions of an integer programming *GCP* formulation also suffer from that symmetry drawback, a Branch-and-Cut algorithm, based on that formulation, tends to behave poorly (even for small instances of *GCP*). The main reason for that is the fact that many subproblems in the enumeration tree have the same optimal value.

In a previous work [25,27], we introduced an integer programming formulation of *GCP* with a reduced number of symmetrical feasible solutions. Furthermore, some families of facet-defining inequalities were derived for the corresponding polytope. Based on the resulting reinforced formulation of the problem, a cutting plane algorithm, capable of attaining very good quality lower bounds, was proposed and tested. In this paper, we extend that cutting plane algorithm into a Branch-and-Cut one. A detailed discussion is presented of implementation aspects of the proposed Branch-and-Cut algorithm. We also suggest and test several options for preprocessing and branching strategies (developed in an attempt to avoid enumerating symmetrical solutions). After extensive computational testing, it appears clear that the use of our cutting planes helps to substantially reduce the size of the enumeration tree.

This paper is organized as follows. In Section 2, the coloring polytope is discussed and some associated polyhedral results are presented. Details of our Branch-and-Cut algorithm are presented in Section 3. Section 4 contains computational results for the DIMACS *GCP* benchmark and also for some randomly generated instances of the problem. The paper is closed in Section 5 with some concluding remarks.

Definitions and notation used throughout this paper are presented next. Given a graph $G=(V,E)$, as defined before, $G[V']=(V',E')$, for $V' \subset V$ and $E'=\{\{u,v\}:\{u,v\}\in E \text{ and } u,v\in V'\}$, is the subgraph of $G$ induced by the vertices in $V'$. A clique of $G$ is a subset of vertices $V'\subset V$ such that, for any pair of vertices $u,v\in V'$, there exists an edge $\{u,v\}\in E$. A stable (or independent) set of $G$ is a subset of vertices $V'\subset V$ such that for any pair of vertices $u,v\in V'$, an edge $\{u,v\}$ is not contained in $E$. A clique (resp. stable set) $K$ is maximal if no clique (resp. stable set) $K'\neq K$ exists in $G$ with $K\subset K'$. The stability number of $G$, $\alpha(G)$, is the maximum size of an independent set in $G$. A clique partition of graph $G$ is a partition $(K_1,\ldots,K_k)$ of $V$ such that $K_i$ is a clique of G for $i=1,\ldots,k$. A sequence $v_1,\ldots,v_k$ of pairwise distinct vertices is a path in $G$ if $\{v_1,v_2\},\ldots,\{v_{k-1},v_k\}\in E$. A path defines a cycle if, in addition, $\{v_1,v_k\}\in E$. A hole is a cycle such that $E=\{\{v_1,v_2\},\ldots,\{v_{k-1},v_k\}\{v_k,v_1\}\}$. The neighborhood of $v$ is defined as $N(v)=\{u:u\in V \text{ and } \{u,v\}\in E\}$. Finally, a graph $G$ is called bipartite if $\chi(G)\leqslant 2$.

We expect the reader to be familiar with integer programming and polyhedral theory. The book [34] contains all the background material needed.

## 2. The coloring polytope

Given a graph $G=(V,E)$, let $|V|=n$ and $|E|=m$. Since no more than $n$ distinct colors are required to color any graph, let $\{x_{ij}:i\in V, 1\leqslant j\leqslant n\}$ be a set of binary 0–1 variables where $x_{ij}=1$ if vertex $i$ is colored with color $j$ and $x_{ij}=0$ otherwise. Additionally, let $\{w_j:1\leqslant j\leqslant n\}$ be a set of binary 0–1 variables that control the use or not of each of the $n$ colors available.The classical IP formulation for *GCP* is:

$$\min \quad \sum_{j=1}^{n} w_j \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_{ij} = 1 \quad \forall i \in V \tag{2}$$

$$x_{ij} + x_{kj} \leqslant w_j \quad \forall \{i,k\} \in E, \ 1 \leqslant j \leqslant n \tag{3}$$

$$x_{ij} \in \{0,1\} \quad \forall i \in V, \ 1 \leqslant j \leqslant n, \quad w_j \in \{0,1\}, \ 1 \leqslant j \leqslant n. \tag{4}$$

Constraints (2) guarantee that every vertex is assigned exactly one color. Constraints (3) state that every pair of adjacent vertices must not share the same color. Constraints (3) additionally enforce that $w_j = 1$ only when at least one vertex is colored with $j$.

In [7] we studied the polytope $\mathscr{SCP}$ associated with this classical formulation. Based on the results of that study, we implemented a Branch-and-Cut algorithm that did not prove very efficient. The existence of far too many symmetrical solutions was one of the main reasons of that lack of success. In [26,27] we proposed three new IP formulations that try to overcome this symmetry bottleneck. The polytopes associated with each of these formulations were studied. In that process, advantages and disadvantages of each formulation were highlighted.

Polyhedral results for the coloring polytope $\mathscr{CP}$ are summarized next. That polytope is the one associated with the most promising formulation investigated in [26,27].

Polytope $\mathscr{CP}$ is defined as

$$\mathscr{CP} = \mathscr{SCP} \cap \{(x,w) : w_j \leqslant \sum_{i \in V} x_{ij} \ \forall 1 \leqslant j \leqslant n \text{ and } w_j \geqslant w_{j+1} \ \forall 1 \leqslant j \leqslant n-1\}.$$

Inequalities that are added to the classical formulation in $\mathscr{CP}$, ensure that color $j$ is only assigned to a given vertex if color $j-1$ is already assigned to another one. In doing so, for any feasible $k$-coloring, symmetrical $k$-colorings that use colors with labels higher than $k$ are therefore eliminated.

The main properties of $\mathscr{CP}$ are presented next. Additional details could be found in [25,26], where all facets we identified for polytope $\mathscr{CP}$ are presented. Some of the facets in [25,26] are not used in the Branch-and-Cut algorithm proposed here.

For an optimal coloring of $G$, consider the set of all vertices colored with a given specific color. Clearly, that set must be a stable set and the cardinality of it cannot exceed the stability number of $G$. Adapting that property to a subgraph $G' = (V', E')$ of $G$ is quite straightforward and thus

$$\sum_{v \in V'} x_{vj_0} \leqslant \alpha(G') w_{j_0}, \tag{5}$$

where $\alpha(G')$ is the stability number of $G'$, is valid for all $j_0 = 1, \ldots, n$.

Any $(n - \alpha(G') + 1)$-coloring that assigns color $j_0$ to a maximum independent set of $G'$ satisfies the above inequality as an equality, so it must be a face of $\mathscr{CP}$. However, if $\alpha(G') \geqslant 2$, inequality (5) is not facet-defining since no $(n)$-coloring exists satisfying the inequality as a strict equality. One can thus strengthen (5) by considering the ordering of the colors to obtain

$$\sum_{v \in V'} x_{vj_0} + \sum_{v \in V} \sum_{j=n-\alpha(G')+1}^{n} x_{vj} \leqslant \alpha(G') w_{j_0} + w_{n-\alpha(G')+1}. \tag{6}$$

Any $(n - \alpha(G') + r)$-coloring, where $r \geqslant 1$, has $\alpha(G') - r$ vertices sharing a color with other vertex. Therefore, there must be at most $\alpha(G') + 1$ vertices colored with $r + 1$ different colors and this implies that (6) is a valid inequality. We call (6) the *Independent Set* inequality. Conditions for (6) to be facet-defining are discussed next.

**Proposition 1.** *Let $G' = G[V']$, $V' \subset V$, be an induced subgraph of $G$ and consider the stability numbers $\alpha(G')$ and $\alpha(G)$, respectively of $G'$ and $G$. Then*

$$\sum_{v \in V'} x_{vj_0} + \sum_{v \in V} \sum_{j=n-\alpha(G')+1}^{n} x_{vj} \leqslant \alpha(G') w_{j_0} + w_{n-\alpha(G')+1} \tag{7}$$

*is a valid inequality for $\mathscr{CP}$ for all $1 \leqslant j_0 \leqslant n - \alpha(G')$. Furthermore, if*

- $\alpha(G') < \alpha(G)$;
- $\forall v \in V \backslash V'$, *subgraph $G[V' \cup \{v\}]$ contains an independent set of cardinality* $(\alpha(G') + 1)$;
- *there exists a maximum independent set $I$ of $G'$, such that $G[V \backslash I]$ is not a clique*;
- *there is some $\chi(G)$-coloring which satisfies the inequality as a strict equality*

*then the inequality is facet-defining for $\mathscr{CP}$.*

Inequalities (6) proved very useful for subgraphs with known stability number. In our algorithm, in particular, we use the ones corresponding to cliques and holes. The two propositions that follow specialize (6) respectively for cliques and holes. In [25,26] the inequalities are also specialized for paths and complement of holes.

**Proposition 2.** *Let $K$ be a maximal clique of $G$ and $1 \leqslant j_0 \leqslant n - 1$. The* Clique *inequality*,

$$\sum_{v \in K} x_{vj_0} \leqslant w_{j_0}$$

*is facet-defining for $\mathscr{CP}$.*

**Proposition 3.** *Let $C_k$ be a hole of $G$ of size $k$ and $1 \leqslant j_0 \leqslant n - \lfloor k/2 \rfloor$. The* Hole *inequality*

$$\sum_{v \in C_k} x_{vj_0} + \sum_{v \in V} \sum_{j=n-\lfloor k/2 \rfloor+1}^{n} x_{vj} \leqslant \lfloor k/2 \rfloor w_{j_0} + w_{n-\lfloor k/2 \rfloor+1}$$

*is valid for $\mathscr{CP}$.*

For $1 \leqslant j_0 \leqslant n - 1$ and $v \in V$, denote $|N(v)| = \delta(v)$. Additionally, consider the subset of constraints (3) defined for vertices in $N(v)$. If constraints in that subset are added together, a valid inequality

$$\sum_{k \in N(v)} x_{kj_0} + \delta(v) x_{vj_0} \leqslant \delta(v) w_{j_0}$$

results. Denoting $r = \alpha(G[N(v)])$, the above inequality can be strengthened to

$$\sum_{k \in N(v)} x_{kj_0} + r x_{vj_0} \leqslant r w_{j_0}.$$

Furthermore, if $1 \leqslant j_0 \leqslant n - r + 1$, feasible solutions that satisfy the inequality as strict equalities have $x_{vj} = 0$ for all $j \geqslant n - r + 2$. Applying a lifting procedure [29] to the inequality, one variable at a time, one obtains the result that follows.

**Proposition 4.** *Let $v \in V$ be such that $r = \alpha(G[N(v)])$ is no less than 2. The* Neighborhood *inequality*,

$$\sum_{u \in N(v)} x_{uj_0} + r x_{vj_0} + \sum_{j=1}^{r-1} j x_{vn-r+j+1} \leqslant r w_{j_0}$$

*is facet-defining for $\mathscr{CP}$ for all $1 \leqslant j_0 \leqslant n - r + 1$.*

An inequality, related with the way in which we eliminate symmetrical solutions, now follows. Notice that if a color labeled $j_0$ is not used in a given coloring of $G$, colors with labels greater than $j_0$ should not be used in that coloring as well. Furthermore, notice that any vertex in $V$ may not have more than one color assigned to it in any coloring of $G$. Putting together these two observations brings us to the next proposition.

**Proposition 5.** *The* Block Color *inequality*

$$\sum_{j=j_0}^{n} x_{i_0 j} \leqslant w_{j_0}$$

*is valid for $\mathscr{CP}$ for all $i_0 \in V$ and $1 \leqslant j_0 \leqslant n-1$.*

Let $P_k = v_1, \ldots, v_k$, for $k \geqslant 3$, be a path and consider a set $\{c_1, \ldots, c_k\}$ of $k$ colors such that $c_k \geqslant c_i$, for $i = 1, \ldots, k-1$. Adding together constraints $x_{v_i c_i} + x_{v_{i+1} c_i} \leqslant w_{c_i}$, for $i = 1, \ldots, k-1$, and *Block Color* inequalities $\sum_{j=c_k}^{n} x_{v_i j} \leqslant w_{c_k}$, for $i = 1, \ldots, k$, results in the following inequality:

$$x_{v_1 c_1} + \sum_{i=2}^{k-1} (x_{v_i c_{i-1}} + x_{v_i c_i}) + x_{v_k c_{k-1}} + \sum_{i=1}^{k} \sum_{j=c_k}^{n} x_{v_i j} \leqslant k w_{c_k} + \sum_{i=1}^{k-1} w_{c_i}.$$

That inequality is clearly valid and is dominated by the set of inequalities used to obtain it. However, the coefficient of $w_{c_k}$ could be strengthened to 1 while preserving validity (since no more than $k$ colors are needed to color a set of $k$ vertices). In doing so, we obtain the result that follows.

**Proposition 6.** *Let $P_k = v_1, \ldots, v_k$, for $k \geqslant 3$, be a path and consider a set $\{c_1, \ldots, c_k\}$ of $k$ colors such that $c_k \geqslant c_i$, for $i = 1, \ldots, k-1$. The* Multicolor Path *inequality*

$$x_{v_1 c_1} + \sum_{i=2}^{k-1} (x_{v_i c_{i-1}} + x_{v_i c_i}) + x_{v_k c_{k-1}} + \sum_{i=1}^{k} \sum_{j=c_k}^{n} x_{v_i j} - w_{c_k} - \sum_{j=1}^{k-1} w_{c_j} \leqslant 0$$

*is valid for $\mathscr{CP}$.*

We now obtain another valid inequality for $\mathscr{CP}$ in a fashion similar to the one proposed above. Let $\{v_1, \ldots, v_p\}$ be a clique of $G$ of size $p$, $k$ be a color label such that $p \leqslant k \leqslant n-1$ and $Col \subseteq \{1, \ldots, k-1\}$ be chosen so that $|Col| = p-1$. Adding together *Clique* inequalities $\sum_{i=1}^{p} x_{v_i j} \leqslant w_j$, for all $j \in Col$, and *Block Color* inequalities $\sum_{j=k}^{n} x_{v_i j} \leqslant w_k$, for $i = 1, \ldots, p$, results in the valid inequality $\sum_{i=1}^{p} \sum_{j \in Col} x_{v_i j} + \sum_{i=1}^{p} \sum_{j=k}^{n} x_{v_i j} \leqslant \sum_{i=1}^{p} w_j + p w_k$. Notice that the coefficient of $w_k$ in the resulting inequality could be strengthened to 1 while preserving validity (since $p$ different colors are required to color the vertices of a clique of size $p$). The proposition that follows thus apply.

**Proposition 7.** *Let $\{v_1, \ldots, v_p\}$ be a clique of $G$ of size $p$, $k$ be a color label such that $p \leqslant k \leqslant n-1$ and $Col \subseteq \{1, \ldots, k-1\}$ be chosen so that $|Col| = p-1$. Then, the* Multicolor Clique *inequality*

$$\sum_{i=1}^{p} \sum_{j=k}^{n} x_{v_i j} + \sum_{i=1}^{p} \sum_{j \in Col} x_{v_i j} \leqslant w_k + \sum_{j \in Col} w_j$$

*is valid for $\mathscr{CP}$.*

The inequalities presented in this section are used in the *GCP* Branch-and-Cut algorithm of Section 3.

## 3. A Branch-and-Cut algorithm for *GCP*

For an integer programming formulation of a given problem, a Branch-and-Bound algorithm partitions the associated solution space into smaller subsets and attempts to optimize the objective function over each of these subsets. In doing so one hopes that the resulting subproblems are *easier* to solve than the original problem. However, if subproblems are still *difficult* to solve, their feasibility regions are further partitioned in a similar way. The process is to recursively continue until either the subproblem in hand is solved to proven optimality or else one could guarantee that the optimal solution value to the original problem dominates that for the subproblem. The scheme is typically represented by an enumeration tree where every tree node has a one-to-one correspondence with the subproblems described above. Furthermore, instead of directly attempting to solve subproblems to optimality, *dual* bounds (i.e. lower bounds for the case of *GCP*) on their optimal solution values are computed. The most commonly used approach to computing dual bounds in Branch-and-Bound is to solve, for each subproblem, the corresponding Linear Programming (LP) relaxation.

A Branch-and-Cut algorithm is a refinement of Branch-and-Bound where LP relaxations for each subproblem are strengthened with globally valid (i.e. valid across the whole enumeration tree) inequalities. To reduce the number of nodes in the Branch-and-Cut enumerating tree, it is obviously important to generate good quality dual bounds on the optimal solution values for the subproblems. Moreover, it is equally important to generate good quality *primal* bounds (i.e. upper bounds for the case of *GCP*) on these values. Similarly, the use of adequate rules to partition feasibility sets and the design of efficient strategies to explore the enumeration tree are key ingredients in a successful implementation of Branch-and-Cut. Our Branch-and-Cut algorithm for *GCP* takes into account all the factors mentioned above. The algorithm is implemented in C++ under the ABACUS framework [17] and uses LP solver CPLEX 6.0 [8].

## 3.1. Preprocessing

The number of variables and inequalities in the model we use may be far too large even for moderately sized coloring problems. The use of preprocessing techniques is therefore essential to attempt to eliminate suboptimal variables and unnecessary constraints.

Preprocessing is initiated with a simple heuristic to find as large a clique $K$ as possible. The size $n\_cli$ of that clique gives a valid lower bound on the chromatic number of $G$. Therefore, every vertex in $K$ may be preassigned a different color.

Some vertices have the property that if they are eliminated from the graph, any coloring of the resulting graph may be extended to a coloring of the original graph without using a new color. Two procedures are applied to identify and process vertices with that property:

- Procedure 1: *Processing vertices adjacent to clique K*
  Let $v \in V \backslash K$ and $w \in K$ be such that $w \notin N(v)$. If $N(v) \subset N(w)$, then a coloring of $G - \{v\}$ can be extended to a coloring of $G$ by assigning to $v$ the same color of $w$.
  Based on this property, we consider vertices in increasing value of their edge degrees and use a sequential procedure to identify and remove vertices meeting the condition above.
- Procedure 2: *Processing vertices by degree*
  Let $v \in V \backslash K$ be such that the degree of $v$ is less than $n\_cli - 1$. It is thus easy to verify that $\chi(G[V \backslash \{v\}])$ equals $\chi(G)$.
  These vertices are recursively deleted until all vertices left have degrees greater than $n\_cli - 2$.

## 3.2. GCP upper bounds

Dsatur [6] is one of best known exact solution algorithms for *GCP*. Dsatur is an implicit enumeration algorithm where each node in the search tree corresponds to a partial coloring of graph $G$. If *UB* is an upper bound on the number of colors in an optimal coloring of $G$, a tree node using at least *UB* colors (in a partial coloring of $G$) could be fathomed. The branching rule used in Dsatur is to generate additional tree nodes, from a given partial coloring of $G$, by coloring a yet uncolored vertex $i$. Let $k$ be the number of colors so far used in the partial coloring being investigated. For each feasible color for $i$, out of the $k$ colors already used in the node, a new tree node is created by assigning the color to $i$. If $k + 1 < UB$, the partial coloring is extended with a new tree node involving vertex $i$ and color $k + 1$. The algorithm terminates when no nodes are left for coloring. Vertex $i$ is chosen as the vertex adjacent to the largest number of differently colored vertices. In case of a tie, a vertex with the highest degree in the uncolored subgraph is chosen. This dynamic reordering of the vertices plays an important role in reducing the number of nodes of the enumeration tree. The node selection strategy used by Dsatur is depth-first search. Alternative vertex selection strategies were proposed by other authors (see for instance, [20,31,33]).

We generate a coloring of $G$ by allowing Dsatur to run for 5 s of CPU time. The solution returned by Dsatur gives an upper bound $\hat{\chi}$ on the chromatic number of $G$ and allows us to eliminate variables $x_{ij}$ and $w_j$ indexed by $j \geqslant \hat{\chi} + 1$.

The implementation of Dsatur that we use incorporates the modification suggested by Sewell [33] and can be found in Michael Trick's home page (http://mat.gsia.cmu.edu/COLOR/solvers/trick.c).

## 3.3. Improving the linear programming relaxation

After the reductions suggested above, our proposed model involves $m\hat{\chi} + n$ constraints. However, typically, a model with that many constraints is difficult to handle for *large* dense graphs. We therefore replace constraints (3) by

$$\sum_{i \in N(k)} x_{ij} + \mu x_{kj} \leqslant \mu w_j \quad \forall\, k \in V \tag{8}$$

where $\mu$ is the cardinality of a clique partition of $N(k)$. In this way we are now faced with $n\hat{\chi} + n$ constraints instead of the $m\hat{\chi} + n$ we had before.

One should notice that the use of the surrogate constraints suggested above does not change the set of feasible integer solutions. However these constraints may lead to weaker LP relaxations. Our computational experience indicates that the use of the surrogate constraints pays off in terms of CPU times. Notice that inequalities (8) are a weak version of the *Neighborhood* inequalities.

Finally, in order to strengthen the resulting LP relaxation, we use inequalities

$$\sum_{j=1}^{\hat{\chi}} w_j \geqslant \sum_{j=1}^{\hat{\chi}} j x_{ij} \quad \forall\, i \in V$$

These inequalities eliminate fractional solutions such as $x_{ij} = 1/\hat{\chi}$ for every $i$, $j$ when $\hat{\chi} \geqslant 3$.

## 3.4. Branching rules

In our initial computational experiments, we tested various branching strategies. The classical rule of branching on a fractional variable by setting it to 1 in one subproblem while setting it to 0 in another subproblem proved very *asymmetrical*. As a result, the enumeration tree it produces tends to be quite unbalanced. That is due to the fact that setting a variable to 1 is equivalent to coloring a particular vertex with the given color while setting it to 0 simply prevents the color from being used for that vertex. The results we got with the classical branching rule were therefore quite disappointing.

We eventually settled for using a branching rule that proved quite effective. First, we select a yet uncolored vertex and then consider all colors so far used in the subproblem under investigation. A new subproblem is then formulated for every feasible assignment of one of these colors to the selected vertex. An additional subproblem which assigns the very first so far unused color to the vertex, is also created.

We follow an idea of Brélatz (see [6] for details) in choosing the branching vertex above. The selected vertex is chosen as the fractional vertex (i.e.variable) adjacent to the largest number of differently colored vertices. In case of a tie, we consider two alternative tie-breaking rules:

VB1: Select a vertex with the maximum degree in the uncolored subgraph;

VB2: Select that vertex which produces the largest decrease in the number of colors available for coloring the remaining uncolored vertices.

The first rule is due to Brélatz [6] and the second one is a modification of a rule proposed by Sewell [33].

The branching strategies suggested above specify ways of splitting the set of feasible solutions of the current subproblem. However, one is still left to determine the order in which subproblems should be investigated. In our algorithm we use a depth-first search rule to select the next tree node to investigate. However, four different alternatives are considered for adding new tree nodes to the list of active subproblems. Namely,

O1: By increasing order of color labels;

O2: First a new color and then by increasing order of color labels;

O3: By increasing order of the number of vertices that have already been colored with each color;

O4: By decreasing order of the number of vertices that have already been colored with each color.

For *small* graphs, the complete enumeration of feasible colorings proved more efficient than our Branch-and-Cut algorithm. Therefore, when the number of vertices still uncolored is *small*, it proved more efficient to implement the complete enumeration scheme. The decision on when to start performing complete enumeration is controlled by a parameter in our implementation. We fixed it to 60 vertices for graphs with more than 60 vertices.

## 3.5. Cutting plane generation

Our previous computational experience [27] indicates that a cutting plane algorithm based on the inequalities presented here is an effective way of strengthening the LP relaxation of $\mathscr{CP}$. In particular, results in [27] suggest that the cutting plane generation strategy that follows is quite attractive. Start by applying separation procedures only for *Clique*, *Block Color* and *Multicolor Path* inequalities. Then, only if no violation of these inequalities is detected, proceed to separate *Hole* and *Multicolor Clique* inequalities. Either way, no more than 1000 constraints should be added per cutting plane generation round.

We give next a brief outline of the separation procedures in [27].

### 3.5.1. Clique inequalities

A simple greedy heuristic is used for the separation of *Clique* inequalities. Given the current LP relaxation solution $(x^*, w^*)$ and a color $j_0$, the entries in $\{x^*_{ij_0} : i \in V \text{ and } x^*_{ij_0} < 1\}$ are ordered in decreasing value of magnitude. The resulting list of ordered entries is then scanned to select vertices to initiate and expand cliques. Selection of vertices is conducted in a greedy fashion.

Let $x^*_{ij_0}$ be fractional and assume that the clique is initialized with vertex $i$. The clique is then attempted to be expanded with vertices associated with entries in the ordered list. Several trials, limited by an input parameter, are performed. In trial $k$, $k \geqslant 1$, a vertex $i'$ is selected to expand the clique, where $i'$ is a the $k$th vertex in the neighborhood of $i$ found while scanning the ordered list (from the highest entry value end). After $i'$ is introduced in the clique, the procedure is recursively repeated with the selection of the next neighbor vertex in the ordered list.

To avoid duplication of work, once a clique is identified, it is used to initialize a separation procedure for the *Multicolor Clique* inequality. For each color $j$, with $1 \leqslant j \leqslant j_0 - 1$, we compute $S_j$ where $S_j = \sum_{i \in clique} x_{ij} - w_j$. If $nc$ is the clique size, in order to increase our chances of finding a violated *Multicolor Clique* inequality, the first $nc - 1$ colors, in decreasing value of $S_j$, are selected to enter the *Multicolor Clique* under construction.

### 3.5.2. Multicolor path inequalities

For every variable $w_k$ with $w^*_k$ fractional, a weight $c_{uv} = \max_{j=1,\dots,k-1} \{x^*_{uj} + x^*_{vj} - w^*_j\} + \sum_{j=k}^{n} (x^*_{uj} + x^*_{vj})$ is computed for every edge $(u, v) \in E$. Under these weights, a path of total weight greater than $w^*_k$ corresponds to a violated *Multicolor Path* inequality.

Using a greedy procedure, the path with the largest weight $P_v$ originating from $v$ is computed. The procedure is initialized with $P_v = v_0$, where $v_0 = v$. Then, for every iteration $j \geqslant 1$, the path is extended with that vertex $v_j$ such that $c_{v_{j-1}v_j} = \max\{c_{v_{j-1}l} : l \in N(v_{j-1}) \text{ and } l \text{ does not belong to } P_v\}$.

To avoid generating inequalities with similar support, the procedure restricts vertex inclusion in a path through the use of an upper bound on the number of paths a vertex may belong to and an upper bound on the length any path may have.

### 3.5.3. Block color inequalities

The *Block Color* inequalities are handled by brute force enumeration. We enumerate all $n^2$ inequalities and find those that are currently violated.

### 3.5.4. Hole inequalities

Let $B = (V_1 \cup V_2, E')$ be a bipartite graph which contains two vertices $v_1$ and $v_2$ for every vertex $v \in V$. Additionally, for every edge $(u, v) \in E$, edges $(u_1, v_2)$ and $(v_1, u_2)$ must belong to $E'$. Then, for a given vertex $v \in V$, it is easy to check that a path in $B$ starting in $v_1$ and ending in $v_2$ induces a cycle in $G$.

In an attempt to find violated ***Hole*** inequalities, we consider a color $j_0$ such that $w^*_{j_0} > 0$ and a vertex set $V' \subset V$ where $v \in V'$ if $x^*_{vj_0}$ is fractional. Then, as described above, a bipartite graph $B'$ is built from the subgraph of $G$ induced by vertex set $V'$. Additionally, a weight $c_{u'_1,v'_2} = c_{v'_1,u'_2} = \max(0, w^*_{j_0} - x^*_{uj_0} - x^*_{vj_0})$ is associated with every edge $(u', v')$

in $B'$. Then, for every vertex $v \in V'$, a shortest path between corresponding vertices $v_1$ and $v_2$ is found in $B'$ by using, say Dijkstra's algorithm [23].

No guarantee exists that the procedure outlined above will always succeed in finding existing violated ***Hole*** inequalities. However, in practice, it proved quite effective in fulfilling that task.

### 3.6. Cut management scheme

Effective management of the pool of violated inequalities is highly important in a Branch-and-Cut algorithm. If far too many violated inequalities are generated and used, time spent in the LP solver may considerably slow down the algorithm. Therefore, ideally, at a given cutting plane generation round, only a limited number of violated inequalities should be used to reinforce LP relaxations. One of the options available here is to generate as many violated inequalities as possible and then select a subset of these for use. One may, for instance, select violated inequalities according to the magnitude of their associated slacks. Alternatively, one may consider the distance of the inequality (the larger the better) from the current fractional LP relaxation solution. However, in the end, we eventually settled for the computationally cheap option of stopping cut generation after a given limited number of violated inequalities are identified. All violated inequalities thus generated are then used to reinforce LP relaxations.

If one keeps adding constraints to LP relaxations and no inequality is eventually dropped from them, LP programs soon become larger than necessary. To reduce memory requirements and cut down on CPU time, ABACUS offers the possibility of eliminating nonbinding cutting planes from LP programs. However, inequalities eliminated in that way may eventually end up being violated again later on in the Branch-and-Cut algorithm. Inequalities eliminated from LP programs are therefore kept in a *pool* so that they may be checked for violation later on. Pool inequalities violation checks are conducted in a very efficient way by ABACUS.

Proceeding as suggested above is very convenient. We use heuristics to solve separation problems and therefore no guarantee exists of identifying a removed inequality when it becomes violated again. Therefore, in our application, the pool of inequalities acts as an additional place to look for cutting planes.

As pointed out before, some key decisions must be made in the design of a Branch-and-Cut algorithm. One must decide, for instance, how many cutting plane rounds should be implemented at every enumeration tree node (before one resorts to branching at the node). Clearly, an appropriate balance between branching and cutting is desirable since small enumeration trees do not always correspond to smaller CPU times. To help us making decisions, we use the following input parameters: a skip factor (i.e. number of tree nodes that are enumerated before one resorts to the use of cutting planes), maximum number of cutting plane rounds per enumeration tree node and maximal number of cuts added per cutting plane round. In the following section, different alternatives for these parameters are tested.

## 4. Computational experiments

In this section we describe our computational experience with *BC-Col*, i.e. the Branch-and-Cut algorithm implemented in this study. Experiments were carried out on a Sun ULTRA1 workstation with CPU running at 140 MHz and 288 Mb of RAM memory. CPU times are reported in seconds.

DIMACS benchmark instances [12] were used in the experiments. Additionally, some randomly generated instances were also used. Let $G(n, p)$ be a graph with $n$ vertices and an independent probability $p$ of an edge existing between two given vertices. That class of graphs is very frequently used for testing graph coloring algorithms. Table 1

describes the DIMACS instances and number of vertices, number of edges and size of maximal cliques are given. The rightmost table column indicates the chromatic number of the corresponding graphs ("?" stands for unknown).

### 4.1. Reducing problem input size

*BC-Col* is initiated with the reduction techniques described in Section 3. Removal of vertices proved highly effective for DIMACS instances. Table 2

shows the DIMACS instances where size reduction was attained. Column entries in that table give graph density, original number of vertices, reduced number, $\hat{n}$, of vertices, and the percentage reduction attained. CPU time spent on reduction tests is minute when compared with the overall CPU time spent by *BC-Col*. Substantial reductions were

Table 1
DIMACS instances

| Problem | Vertices | Edges | $n\_cli$ | $\chi$ |
|---|---|---|---|---|
| DSJC125_1 | 125 | 736 | 4 | ? |
| DSJC125_5 | 125 | 3891 | 9 | ? |
| DSJC125_9 | 125 | 6961 | 32 | ? |
| DSJC250_1 | 250 | 3218 | 4 | ? |
| DSJC250_5 | 250 | 15668 | 11 | ? |
| DSJC250_9 | 250 | 27897 | 37 | ? |
| DSJC500_1 | 500 | 12458 | 5 | ? |
| DSJC500_5 | 500 | 62624 | 12 | ? |
| DSJC500_9 | 500 | 1124367 | 47 | ? |
| DSJR500_1 | 500 | 3555 | 12 | 12 |
| DSJR500_1C | 500 | 121275 | 72 | ? |
| DSJR500_5 | 500 | 58862 | 117 | ? |
| DSJC1000_1 | 1000 | 49629 | 6 | ? |
| DSJC1000_5 | 1000 | 249826 | 14 | ? |
| DSJC1000_9 | 1000 | 449449 | 55 | ? |
| fpsol2_i_1 | 496 | 11654 | 55 | 65 |
| fpsol2_i_2 | 451 | 8691 | 29 | 30 |
| fpsol2_i_3 | 425 | 8688 | 29 | 30 |
| inithx.i.1 | 864 | 18707 | 54 | 54 |
| inithx.i.2 | 645 | 13979 | 31 | 31 |
| inithx.i.3 | 621 | 13969 | 31 | 31 |
| latin_squ_10 | 900 | 307350 | 90 | ? |
| le450_15a | 450 | 8168 | 15 | 15 |
| le450_15b | 450 | 8169 | 15 | 15 |
| le450_15c | 450 | 16680 | 15 | 15 |
| le450_15d | 450 | 16750 | 15 | 15 |
| le450_25a | 450 | 8260 | 25 | 25 |
| le450_25b | 450 | 8263 | 25 | 25 |
| le450_25c | 450 | 17343 | 25 | 25 |
| le450_25d | 450 | 17425 | 25 | 25 |
| le450_5a | 450 | 5714 | 5 | 5 |
| le450_5b | 450 | 5734 | 5 | 5 |
| le450_5c | 450 | 9803 | 5 | 5 |
| le450_5d | 450 | 9757 | 5 | 5 |
| mulsol.i.1 | 197 | 3925 | 49 | 49 |
| mulsol.i.2 | 188 | 3885 | 31 | 31 |
| mulsol.i.3 | 184 | 3916 | 31 | 31 |
| mulsol.i.4 | 185 | 3946 | 31 | 31 |
| mulsol.i.5 | 185 | 3973 | 31 | 31 |
| school1 | 385 | 19095 | 14 | 14 |
| school1_nsh | 352 | 14612 | 14 | 14 |
| zeroin.i.1 | 211 | 4100 | 49 | 49 |
| zeroin.i.2 | 211 | 3541 | 30 | 30 |
| zeroin.i.3 | 206 | 3540 | 30 | 30 |
| anna | 138 | 493 | 11 | 11 |
| david | 87 | 406 | 11 | 11 |
| homer | 561 | 1629 | 13 | 13 |
| huck | 74 | 301 | 11 | 11 |
| jean | 80 | 254 | 10 | 10 |
| games120 | 120 | 638 | 9 | 9 |
| miles1000 | 128 | 3216 | 41 | 42 |
| miles1500 | 128 | 5198 | 71 | 73 |
| miles250 | 128 | 387 | 8 | 8 |
| miles500 | 128 | 1170 | 20 | 20 |
| miles750 | 128 | 2113 | 31 | 31 |
| queen10_10 | 100 | 2940 | 10 | ? |
| queen11_11 | 121 | 3960 | 11 | 11 |
| queen12_12 | 144 | 5192 | 12 | ? |

Table 1 (*continued*)

| Problem | Vertices | Edges | *n_cli* | $\chi$ |
|---|---|---|---|---|
| queen13_13 | 169 | 6656 | 13 | 13 |
| queen14_14 | 196 | 8372 | 14 | ? |
| queen15_15 | 225 | 10360 | 15 | ? |
| queen16_16 | 256 | 12640 | 16 | ? |
| queen8_12 | 96 | 1368 | 12 | 12 |
| queen8_8 | 64 | 728 | 8 | 9 |
| queen9_9 | 81 | 1056 | 9 | 10 |
| myciel6 | 95 | 755 | 2 | 7 |
| myciel7 | 191 | 2360 | 2 | 8 |
| mug88_1 | 88 | 146 | 3 | 4 |
| mug88_25 | 88 | 146 | 3 | 4 |
| mug100_1 | 100 | 166 | 3 | 4 |
| mug100_25 | 100 | 166 | 3 | 4 |
| abb313GPIA | 1557 | 46546 | 8 | ? |
| ash331GPIA | 662 | 4185 | 3 | ? |
| ash608GPIA | 1216 | 7844 | 3 | ? |
| ash958GPIA | 1916 | 12506 | 3 | ? |
| will199GPIA | 701 | 6772 | 5 | ? |
| 1-Insertions_4 | 67 | 232 | 2 | ? |
| 1-Insertions_5 | 202 | 1227 | 2 | ? |
| 1-Insertions_6 | 607 | 6337 | 2 | ? |
| 2-Insertions_4 | 149 | 541 | 2 | 4 |
| 2-Insertions_5 | 597 | 3936 | 2 | ? |
| 3-Insertions_3 | 56 | 110 | 2 | 4 |
| 3-Insertions_4 | 281 | 1046 | 2 | ? |
| 3-Insertions_5 | 1406 | 9695 | 2 | ? |
| 4-Insertions_3 | 79 | 156 | 2 | ? |
| 4-Insertions_4 | 475 | 1795 | 2 | ? |
| 1-FullIns_4 | 93 | 593 | 3 | ? |
| 1-FullIns_5 | 282 | 3247 | 3 | ? |
| 2-FullIns_3 | 52 | 201 | 4 | ? |
| 2-FullIns_4 | 212 | 1621 | 4 | ? |
| 2-FullIns_5 | 852 | 12201 | 4 | ? |
| 3-FullIns_3 | 80 | 346 | 5 | ? |
| 3-FullIns_4 | 405 | 3524 | 5 | ? |
| 3-FullIns_5 | 2030 | 33751 | 5 | ? |
| 4-FullIns_3 | 114 | 541 | 6 | ? |
| 4-FullIns_4 | 690 | 6650 | 6 | ? |
| 4-FullIns_5 | 4146 | 77305 | 6 | ? |
| 5-FullIns_3 | 154 | 792 | 7 | ? |
| 5-FullIns_4 | 1085 | 11395 | 7 | ? |
| wap01 | 2368 | 110871 | 41 | ? |
| wap02 | 2464 | 111742 | 40 | ? |
| wap03 | 4730 | 286722 | 40 | ? |
| wap04 | 5231 | 294902 | 40 | ? |
| wap05 | 905 | 43081 | 50 | ? |
| wap06 | 947 | 43571 | 40 | ? |
| wap07 | 1809 | 103368 | 40 | ? |
| wap08 | 1870 | 104176 | 40 | ? |
| qg_order30 | 900 | 26100 | 30 | 30 |
| qg_order40 | 1600 | 62400 | 40 | 40 |
| qg_order60 | 3600 | 212400 | 60 | 60 |

more systematically attained for low density graphs. However, in some cases, significant reductions were also attained for medium density and high density graphs.

Reduction tests failed for random graphs. An explanation for this is that all vertex degrees are very similar in these graphs.

Table 2
Vertex reduction

| Problem | % Density | $n$ | $\hat{n}$ | % Red. |
|---|---|---|---|---|
| DSJR500_1 | 3 | 500 | 109 | 78 |
| DSJR500_1C | 97 | 500 | 410 | 18 |
| DSJR500_5 | 47 | 500 | 491 | 2 |
| fpsol2_i_1 | 9 | 496 | 171 | 66 |
| fpsol2_i_2 | 9 | 451 | 164 | 64 |
| fpsol2_i_3 | 10 | 425 | 163 | 62 |
| inithx.i.1 | 5 | 864 | 115 | 87 |
| inithx.i.2 | 7 | 645 | 182 | 72 |
| inithx.i.3 | 7 | 621 | 172 | 72 |
| latin_square_10 | 76 | 900 | 129 | 86 |
| le450_15a | 8 | 450 | 409 | 9 |
| le450_15b | 8 | 450 | 413 | 8 |
| le450_25a | 8 | 450 | 271 | 40 |
| le450_25b | 8 | 450 | 302 | 33 |
| le450_25c | 17 | 450 | 436 | 3 |
| le450_25d | 17 | 450 | 436 | 3 |
| mulsol.i.1 | 20 | 197 | 49 | 75 |
| mulsol.i.2 | 22 | 188 | 100 | 47 |
| mulsol.i.3 | 23 | 184 | 101 | 45 |
| mulsol.i.4 | 23 | 185 | 102 | 45 |
| mulsol.i.5 | 23 | 185 | 102 | 45 |
| school1 | 26 | 385 | 358 | 7 |
| school1_nsh | 24 | 352 | 328 | 7 |
| zeroin.i.1 | 18 | 211 | 63 | 70 |
| zeroin.i.2 | 16 | 211 | 57 | 73 |
| zeroin.i.3 | 17 | 206 | 56 | 73 |
| anna | 5 | 138 | 17 | 88 |
| david | 11 | 87 | 11 | 87 |
| homer | 1 | 561 | 38 | 93 |
| huck | 11 | 74 | 11 | 85 |
| jean | 8 | 80 | 13 | 84 |
| games120 | 9 | 120 | 119 | 1 |
| miles1000 | 39 | 128 | 50 | 61 |
| miles1500 | 63 | 128 | 85 | 34 |
| miles250 | 5 | 128 | 15 | 88 |
| miles500 | 14 | 128 | 28 | 78 |
| miles750 | 26 | 128 | 37 | 71 |
| abb313GPIA | 4 | 1557 | 1400 | 10 |
| ash331GPIA | 2 | 662 | 661 | 1 |
| ash608GPIA | 1 | 1216 | 1215 | 1 |
| ash958GPIA | 1 | 1916 | 1915 | 1 |
| will199GPIA | 3 | 701 | 697 | 1 |
| 1-FullIns_4 | 14 | 93 | 63 | 32 |
| 1-FullIns_5 | 8 | 282 | 189 | 33 |
| 2-FullIns_3 | 15 | 52 | 40 | 23 |
| 2-FullIns_4 | 7 | 212 | 160 | 25 |
| 2-FullIns_5 | 3 | 852 | 640 | 25 |
| 3-FullIns_3 | 11 | 80 | 65 | 19 |
| 3-FullIns_4 | 4 | 405 | 325 | 20 |
| 3-FullIns_5 | 2 | 2030 | 1625 | 20 |
| 4-FullIns_3 | 8 | 114 | 84 | 26 |
| 4-FullIns_4 | 3 | 690 | 576 | 17 |
| 4-FullIns_5 | 1 | 4146 | 3456 | 17 |
| 5-FullIns_3 | 7 | 154 | 79 | 49 |
| 5-FullIns_4 | 2 | 1085 | 931 | 14 |
| wap01 | 4 | 2368 | 1771 | 25 |
| wap02 | 4 | 2464 | 2174 | 12 |
| wap03 | 3 | 4730 | 4701 | 1 |

Table 2 (*continued*)

| Problem | % Density | $n$ | $\hat{n}$ | % Red. |
|---|---|---|---|---|
| wap04 | 2 | 5231 | 5204 | 1 |
| wap05 | 11 | 905 | 665 | 27 |
| wap06 | 10 | 947 | 787 | 17 |
| wap07 | 6 | 1809 | 1655 | 9 |
| wap08 | 6 | 1870 | 1696 | 9 |

## 4.2. Branching strategies

We investigated the impact of the different combinations involving the two branching strategies and the four ways of adding new tree nodes to the list of active subproblems, proposed in Section 3.4. Eight different combinations are possible.

Initial experiments indicated that the relative performance of branching rules is not affected by the strengthening of LP relaxations with cutting planes. Branching rules were therefore tested on a Branch-and-Bound version of our code. Testing was conducted on random graphs with 50 vertices (edge probabilities 0.5, 0.7) and 60 vertices (edge probabilities 0.3 and 0.9). In Fig. 1 we report on the average results obtained over five instances for each edge probability considered.

In our experiments, VB2 proved to be, in most cases, a better option (to add nodes to the list of active tree nodes) than VB1. That is in accordance with the results obtained by Sewell [33]. In [33] an enumerative algorithm that combines VB2 and O1 was proposed and compared with Dsatur (VB1 + O1). Typically, Sewell's algorithm enumerates fewer tree nodes than Dsatur but requires more CPU time (since the tie-breaking rule computations in [33] are relatively expensive). Such a limitation does not apply in our case since the percentage of the total CPU time spent on tie-breaking duties in our algorithm is relatively small.

From Fig. 1 it appears clear that the best combination tested is VB2 + O2. That combination dominates the other ones in terms of the number of subproblems explored and total CPU time.

Concluding our evaluation of branching strategies, we compare the classical 0–1 variable selection dichotomy (a fractional variable is set to 1 in one subproblem and set to 0 in another subproblem) with the best and worst strategies in Fig. 1. The comparison was carried out over the same random instances used before. Results in Fig. 2 indicate that the classical strategy is not an attractive option.

## 4.3. Skip factor

A very important efficiency related issue in a Branch-and-Cut algorithm is the decision on to whether or not to generate cutting planes at a given node of the enumeration tree. In an attempt to answer that question, different options were tested where the cutting plane algorithm was only called at every $\beta$ tree nodes explored by *BC-Col*, where $\beta \geqslant 1$. Parameter $\beta$ is called *skip factor*. In that experiment, *BC-Col* was applied to random graphs with 70 vertices and densities 0.3, 0.5, 0.7, and 0.9, $\beta \in \{1, 2, 4, 8\}$, 3 cutting plane rounds per tree node (when that applies), a maximum of 1000 cutting planes generated for every cutting plane round and no more than 20 cutting plane rounds at the root node of the enumeration tree. Fig. 3 gives the average CPU time and the average size of the search tree for the 10 instances considered for every parameter combination tested. Due to the very different magnitudes associated with the results obtained, skip factor 1 results were standardized to 1. The results obtained indicate that *BC-Col* with skip factor 1 is faster than all the other alternatives available. Indeed, *BC-Col* with skip factor 1 was the fastest alternative for every single instance tested.

## 4.4. Cutting plane rounds

An initial *GCP* lower bound is obtained in *BC-Col* through a greedy heuristic that attempts to generate a maximal clique of $G$. Additional lower bounds are obtained through LP relaxations of the formulation being used. Computational experiments in [27] indicate that LP relaxation bounds are greatly improved when the cutting planes described in this

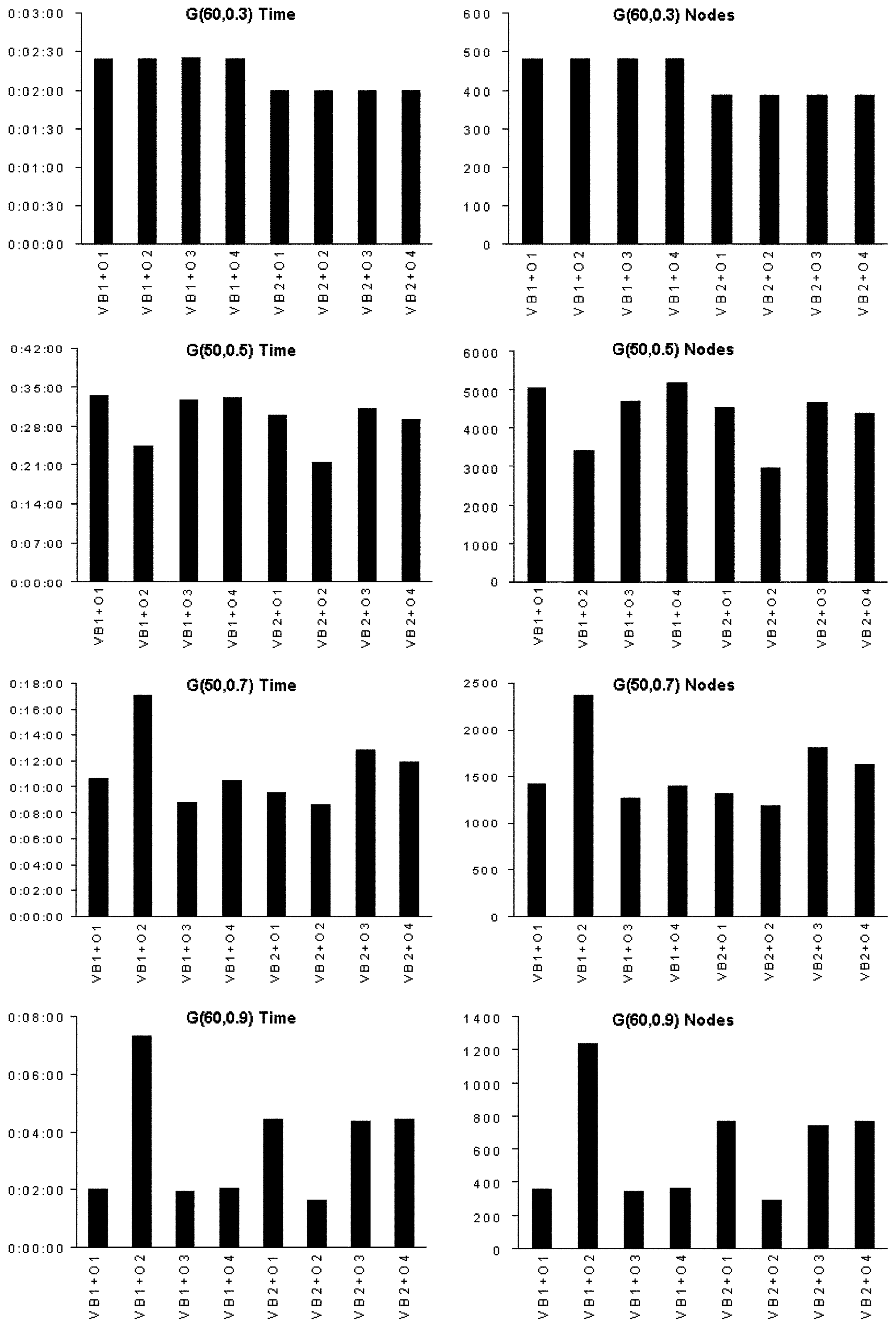


Fig. 1. Branching strategies.

study are used. Furthermore, 20 cutting plane rounds at the root node of the enumeration tree is suggested in [27] as being a good compromise between lower bound improvement and CPU time demands. We follow that suggestion in this study.

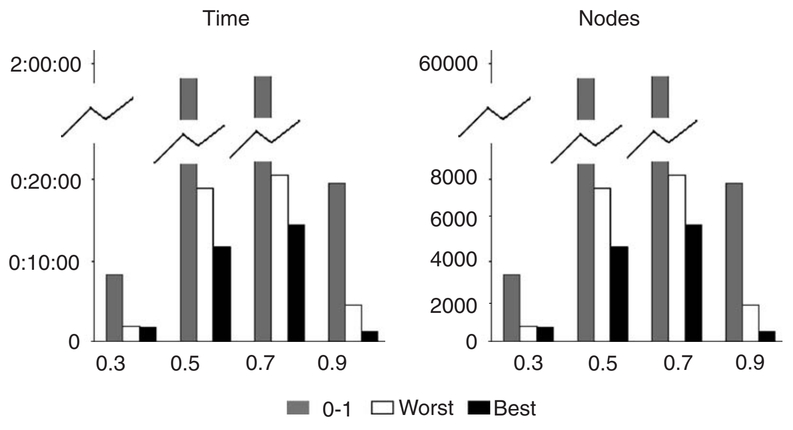


Fig. 2. Worst vs 0–1 vs best strategies.

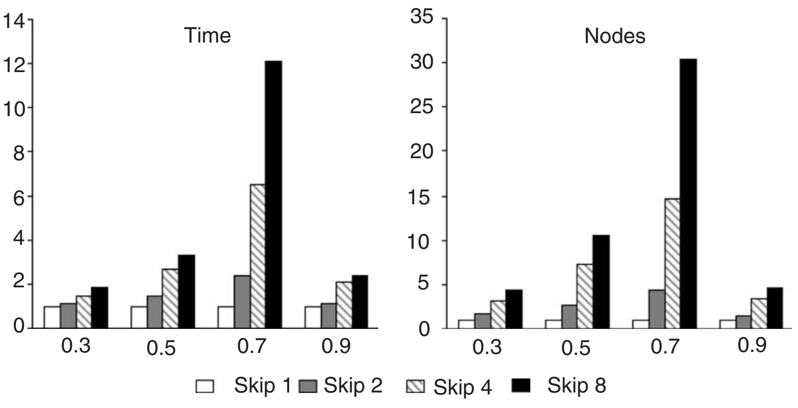


Fig. 3. Skip factor.

We now concentrate in determining the *ideal* number of cutting plane rounds to apply at tree nodes other than the root. For that purpose, we use the *IPC* parameter which controls the number of cutting plane rounds to be performed at these nodes. In general terms, a significant lower bound improvement attained at the root node of the enumeration tree is more important than a corresponding improvement attained at the other tree nodes. One should thus be prepared to spend more CPU time at the root node than at the other tree nodes.

We experimented with *BC-Col* under different *IPC* values. Fig. 4 shows the results obtained for 20 cutting plane rounds at the root node of the enumeration tree, and 1, 2, 4, and 6 rounds, respectively, for the remaining tree nodes. Average results over 10 instances are given, respectively, for densities 0.3, 0.5, 0.7 and 0.9. Due to the very different magnitudes associated with the results obtained, CPU times for 2 cutting plane rounds per tree node (other than the root) were standardized to 1. The results obtained clearly indicate that the more cutting plane rounds are allowed per tree node, the higher is the reduction in the number of tree nodes generated. However, when one goes over 2 cutting plane rounds per tree node (other than the root), the resulting increase in CPU time clearly offsets the reduction attained in the number of tree nodes. We have then settled for 2 cutting plane rounds per tree node (other than the root).

## *4.5. Branch-and-Bound vs BC-Col*

The benefits of using cutting planes (throughout the enumeration tree) may be apparent from our skip factor analysis. However, that point is important enough to be specifically highlighted. We then compare our *BC-Col* algorithm (with skip factor 1) with a Branch-and-Bound algorithm (i.e. *BC-Col* with the cutting plane subroutine switched off). Comparisons

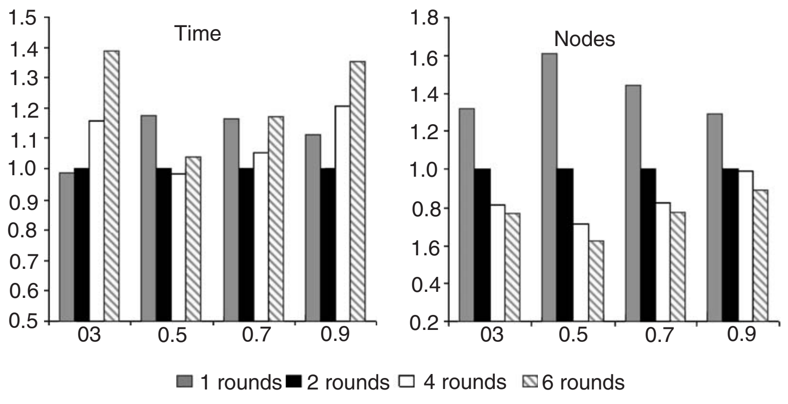
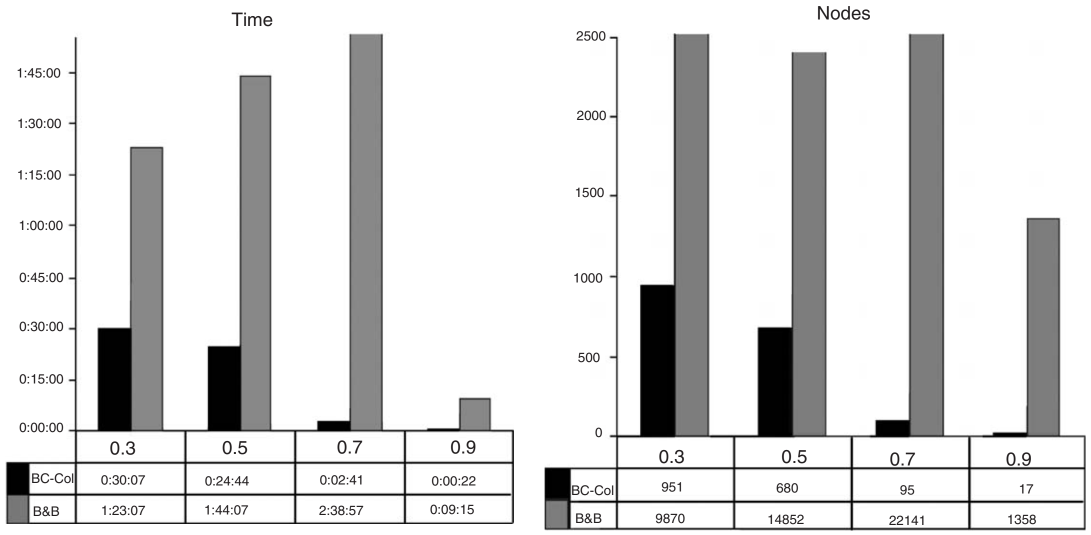


Fig. 4. Rounds per node.



| | 0.3 | 0.5 | 0.7 | 0.9 |
|---|---|---|---|---|
| BC-Col | 0:30:07 | 0:24:44 | 0:02:41 | 0:00:22 |
| B&B | 1:23:07 | 1:44:07 | 2:38:57 | 0:09:15 |

| | 0.3 | 0.5 | 0.7 | 0.9 |
|---|---|---|---|---|
| BC-Col | 951 | 680 | 95 | 17 |
| B&B | 9870 | 14852 | 22141 | 1358 |

Fig. 5. Branch-and-Bound vs *BC-Col*.

were carried out over 10 instances per density considered. Results in Fig. 5 indicate that *BC-Col* attains a better performance than Branch-and-Bound. As one may appreciate from the results obtained, Branch-and-Cut managed to solve *GCP* instances that could not otherwise be solved by Branch-and-Bound within the 2 h CPU time limit imposed. Notice that Branch-and-Bound could only tackle, within the time limit imposed, instances with up to 60 vertices. For solving instances larger than that, the use of cutting planes seems to be essential.

## *4.6. CPLEX vs BC-Col*

After implementing a Branch-and-Cut algorithm, it appears quite natural to compare it with a general purpose IP-solver. For our algorithm, comparisons were carried out with CPLEX [8]. Instances of 50 and 60 vertices with densities of 0.3, 0.5, 0.7 and 0.9, respectively, were used in the experiments. Table 3 shows average results over 10 instances for each of the parameter settings considered. CPU times and the number of tree nodes explored by each algorithm are given. The symbol ***** is used to indicate that the corresponding instance could not be solved within the 2 h CPU time limit imposed. Runs with CPLEX were performed using all the advantages the package offers: preprocessing, clique cuts and cover cuts. The search strategy used was Best-Bound search while the classical 0-1 dichotomy (branching on the variable with maximum infeasibility) was applied.

*BC-Col*'s advantage over CPLEX is larger for instances associated with medium density graphs. For these instances, CPLEX frequently exceeded the time limit imposed.

Table 3
CPLEX vs *BC-Col*

| 0.3 Density | | | | 0.5 Density | | | |
|---|---|---|---|---|---|---|---|
| *BC-Col* | | CPLEX | | *BC-Col* | | CPLEX | |
| Time | Nodes | Time | Nodes | Time | Nodes | Time | Nodes |
| 98 | 100 | 645 | 1236 | 239 | 247 | ***** | ***** |
| 306 | 128 | ***** | ***** | 492 | 348 | ***** | ***** |
| 45 | 36 | 303 | 568 | 310 | 397 | 6468 | 8134 |
| 12 | 7 | 137 | 324 | 33 | 103 | 589 | 1106 |
| 75 | 21 | 1046 | 586 | 155 | 126 | ***** | ***** |
| 97 | 41 | ***** | ***** | 3 | 1 | 4 | 4 |
| 9 | 1 | 7 | 4 | 56 | 56 | ***** | ***** |
| 14 | 16 | 54 | 128 | 164 | 191 | ***** | ***** |
| 86 | 78 | 322 | 492 | 172 | 75 | ***** | ***** |
| 113 | 27 | ***** | ***** | 62 | 41 | ***** | ***** |
| 0.7 Density | | | | 0.9 Density | | | |
| *BC-Col* | | CPLEX | | *BC-Col* | | CPLEX | |
| Time | Nodes | Time | Nodes | Time | Nodes | Time | Nodes |
| 62 | 116 | 1389 | 1979 | 8 | 7 | 22 | 438 |
| 44 | 106 | 71 | 125 | 7 | 5 | 10 | 237 |
| 31 | 75 | 85 | 206 | 29 | 45 | ***** | ***** |
| 39 | 49 | ***** | ***** | 12 | 9 | 678 | 9929 |
| 53 | 132 | 56 | 152 | 8 | 7 | 36 | 903 |
| 37 | 33 | ***** | ***** | 3 | 8 | 5 | 198 |
| 35 | 38 | ***** | ***** | 3 | 5 | 3 | 67 |
| 31 | 44 | ***** | ***** | 24 | 26 | 371 | 2935 |
| 20 | 30 | 7668 | 26277 | 0 | 4 | 0 | 1 |
| 16 | 28 | 1347 | 3773 | 2 | 20 | 1 | 59 |

A comparison of the (CPU time—number of nodes explored) ratios for the two algorithms indicate that CPLEX generates a larger number of tree nodes per unit of CPU time. Indeed, it is not difficult to explain why that ratio is much lower for *BC-Col*. Separation algorithms and LP re-optimizations increase CPU time significantly at every tree node. Furthermore, *BC-Col* attempts to take advantage of the flexibility offered by some of the structures and procedures found in ABACUS. In particular, these structures and procedures help with cutting planes generation and the implementation of search and branching strategies. However, that convenience is offered at a considerable CPU time price. In any case, the results obtained speak for themselves. In spite of CPLEX soundness and implementation efficiency, *BC-Col* proved to be the more effective option.

## *4.7. Final results*

After fine tuning *BC-Col*, we compared it with Dsatur over the DIMACS and the random graph instances. Complete enumeration of feasible solutions for instances associated with graphs of less than 50 vertices is typically very fast to carry out. We therefore do not use this kind of instance in our experiments. A CPU time limit of 2 h is enforced for each algorithm. *BC-Col* is tuned as follows:

- Node selection: Depth-First-Search.
- Branching Rule: VB2 + O2.
- Skip Factor: 1.
- Rounds of cutting planes : 20 rounds at root node and 2 rounds for the rest of the search tree.

We start by commenting on the results obtained for random graph instances (see Table 4 ). An entry * in that table indicates that the time limit imposed was exceeded.

Table 4
*BC-Col* on random graphs

| G(80,40) | | G(70,0.5) | | G(75,0.5) | | G(70,0.7) | | G(75,0.7) | |
|---|---|---|---|---|---|---|---|---|---|
| Dsatur | *BC-Col* | Dsatur | *BC-Col* | Dsatur | *BC-Col* | Dsatur | *BC-Col* | Dsatur | *BC-Col* |
| 939 | 343 | 560 | ***** | 440 | 2135 | 247 | 85 | 3443 | 597 |
| ***** | 3835 | 1814 | 742 | 6027 | 1631 | 767 | 65 | 5171 | 2978 |
| 4209 | 1291 | 125 | 94 | 866 | 5754 | 2197 | 449 | ***** | 1778 |
| 216 | 192 | 481 | 91 | ***** | 6393 | 1559 | 511 | ***** | 2058 |
| ***** | 698 | 1023 | 302 | 2037 | 3545 | 1010 | 362 | 1389 | 545 |
| 6447 | 2308 | 72 | 336 | 580 | 3294 | ***** | 674 | 1443 | 576 |
| 720 | 347 | 383 | 205 | 5825 | 4414 | 290 | 34 | 3200 | 2087 |
| ***** | 4953 | 767 | 203 | 5655 | 4141 | 1208 | 238 | ***** | 1933 |
| 195 | 206 | 4065 | 950 | 995 | 994 | 286 | 57 | ***** | 1914 |
| ***** | 2015 | 268 | 213 | 2181 | 429 | 1057 | 186 | 5975 | ***** |
| G(125,0.1) | | G(80,0.2) | | G(80,0.3) | | G(80,0.9) | | G(90,0.9) | |
| Dsatur | *BC-Col* | Dsatur | *BC-Col* | Dsatur | *BC-Col* | Dsatur | *BC-Col* | Dsatur | *BC-Col* |
| 48 | 221 | 2 | 40 | 308 | 228 | 30 | 10 | 24 | 5 |
| 5240 | 2767 | 1 | 19 | 12 | 99 | 14 | 6 | 4043 | 2146 |
| 1 | 86 | 3 | 66 | 152 | 177 | 284 | 50 | 162 | 69 |
| 61 | 43 | 18 | 51 | 3778 | 1187 | 11 | 2 | ***** | ***** |
| 30 | 64 | 8 | 44 | 15 | 85 | 1173 | 2764 | ***** | ***** |
| 4081 | 830 | 6 | 56 | 101 | 337 | 60 | 13 | ***** | 5496 |
| 120 | 685 | 2 | 70 | 588 | 135 | 6291 | 1114 | ***** | 395 |
| 2199 | 1280 | 1 | 57 | 152 | 153 | 353 | 70 | 1174 | 284 |
| 411 | 180 | 5 | 11 | 8 | 106 | ***** | 1840 | 3890 | 1065 |
| 4 | 63 | 4 | 63 | 182 | 128 | 540 | 160 | ***** | 3307 |

Table 5
Average time on random graphs

| G(80,0.4) | | G(70,0.5) | | G(75,0.5) | | G(70,0.7) | | G(75,0.7) | |
|---|---|---|---|---|---|---|---|---|---|
| Dsatur | *BC-Col* | Dsatur | *BC-Col* | Dsatur | *BC-Col* | Dsatur | *BC-Col* | Dsatur | *BC-Col* |
| *Average time* | | | | | | | | | |
| 2121 | 866 | 956 | 348 | 2734 | 2926 | 958 | 221 | 3437 | 1357 |
| *% of not solved instances* | | | | | | | | | |
| 40 | 0 | 0 | 10 | 10 | 0 | 10 | 0 | 40 | 10 |
| G(125,0.1) | | G(80,0.2) | | G(80,0.3) | | G(80,0.9) | | G(90,0.9) | |
| Dsatur | *BC-Col* | Dsatur | *BC-Col* | Dsatur | *BC-Col* | Dsatur | *BC-Col* | Dsatur | *BC-Col* |
| *Average time* | | | | | | | | | |
| 606 | 1220 | 7 | 48 | 530 | 264 | 973 | 465 | 1859 | 714 |
| *% of not solved instances* | | | | | | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 | 50 | 20 |

From the computational evidence available in the literature, it appears that random graph instances are the most difficult ones for coloring. The available exact solution algorithms only manage to solve exactly instances with as many as 70 vertices. Random graph instances also proved the hardest for *BC-Col* to solve. However, instances with as many as 90 vertices were solved to proven optimality by that algorithm.

Low-density random graph instances (i.e instances with graph density less than 0.3) were solved faster by Dsatur. This is not only due to the good performance of Dsatur over that type of instance but also to the low impact our valid inequalities appear to have on these instances. The importance of *Clique* inequalities in strengthening our LP relaxation bounds have already been pointed out in [27]. For low-density random graphs, initial lower and upper bounds tend to

Table 6
Instances solved by the initial heuristic

| Problem | $\chi$ | Problem | $\chi$ |
|---|---|---|---|
| DSJR500_1 | 12 | zeroin.i.1 | 49 |
| inithx.i.1 | 54 | zeroin.i.2 | 30 |
| inithx.i.2 | 31 | zeroin.i.3 | 30 |
| inithx.i.3 | 31 | anna | 11 |
| le450_25a | 25 | david | 11 |
| le450_25b | 25 | homer | 13 |
| le450_5c | 5 | huck | 11 |
| mulsol.i.1 | 49 | jean | 10 |
| mulsol.i.2 | 31 | games120 | 9 |
| mulsol.i.3 | 31 | miles250 | 8 |
| mulsol.i.4 | 31 | miles500 | 20 |
| mulsol.i.5 | 31 | miles750 | 31 |
| school1 | 14 | queen8_12 | 12 |
| school1_nsh | 14 | qg_order30 | 30 |

Table 7
Optimal results

| Problem | $n\_cli$ | $\hat{\chi}$ | $\chi$ | *BC-Col* | Dsatur |
|---|---|---|---|---|---|
| DSJC125_1 | 4 | 5 | 5 | 0.9 | 0.1 |
| fpsol2_i_1 | 55 | 65 | 65 | 0.6 | 0.1 |
| fpsol2_i_2 | 29 | 30 | 30 | 1.2 | 0.1 |
| fpsol2_i_3 | 29 | 30 | 30 | 1.1 | 0.1 |
| miles1000 | 41 | 42 | 42 | 0.02 | 0.1 |
| miles1500 | 71 | 73 | 73 | 0.1 | 0.1 |
| mug88_1 | 3 | 4 | 4 | 11 | ***** |
| mug88_25 | 3 | 4 | 4 | 184 | 4756 |
| mug100_1 | 3 | 4 | 4 | 60 | ***** |
| mug100_25 | 3 | 4 | 4 | 60 | ***** |
| queen8_8 | 8 | 10 | 9 | 3 | 18 |
| ash331GPIA | 3 | 4 | 4 | 51 | 0.7 |
| ash608GPIA | 3 | 4 | 4 | 692 | 3 |
| will199GPIA | 6 | 7 | 7 | ***** | 1.2 |
| 1-Insertions_4 | 2 | 5 | 5 | 2 | ***** |
| 3-Insertions_3 | 2 | 4 | 4 | 1 | 5 |
| 4-Insertions_3 | 2 | 4 | 4 | 4204 | 4701 |
| 1-FullIns_4 | 3 | 5 | 5 | 0.1 | ***** |
| 2-FullIns_3 | 4 | 5 | 5 | 0.1 | 1014 |
| 3-FullIns_3 | 5 | 6 | 6 | 0.1 | ***** |
| 4-FullIns_3 | 6 | 7 | 7 | 3 | ***** |
| 5-FullIns_3 | 7 | 8 | 8 | 20 | ***** |

be quite close to each other, maximal cliques tend to have low cardinality and $n\_cli$ typically differs little from $\chi(G)$. As a result, limited bound improvements are to be expected when using our valid inequalities. Likewise, no effective pruning is to be expected for these instances.

Medium density graph instances (i.e. instances associated with graphs with densities between 0.3 and 0.7) proved the most difficult for Dsatur. For these instances, initial lower and upper bounds tend to quite distant from each other. Furthermore, the number of nodes in the Dsatur enumeration tree appear to increase exponentially. For these instances, Dsatur could only go up to 70 vertices within the CPU time limit imposed. The results obtained by Dsatur clearly indicate that these instances are hard to solve. *BC-Col* however, proved very effective on that test set, being able to solve, to proven optimality, considerably larger instances than Dsatur. That was particularly true for higher density instances.

Results obtained for high density graph instances (i.e. instances associated with graphs with densities over 0.7) do not show a clear pattern. Some instances could be solved to proven optimality in a few CPU s while others could not

Table 8
Bounds

| Problem | *n_cli* | $\hat{\chi}$ | $\chi$ | *BC-Col* | | | Dsatur | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | Lower | Upper | %Gap | Lower | Upper | %Gap |
| DSJC125_5 | 9 | 20 | ? | 13 | 20 | 35 | 9 | 19 | 52.63 |
| DSJC125_9 | 32 | 47 | ? | 42 | 47 | 10.6 | 29 | 45 | 35.5 |
| DSJC250_1 | 4 | 9 | ? | 5 | 9 | 44.4 | 4 | 9 | 55.5 |
| DSJC250_5 | 11 | 36 | ? | 13 | 36 | 63.8 | 9 | 35 | 74.2 |
| DSJC250_9 | 38 | 88 | ? | 48 | 88 | 45.4 | 34 | 87 | 60.9 |
| DSJC500_1 | 5 | 15 | ? | 5 | 15 | 66.6 | 5 | 15 | 66.6 |
| DSJC500_5 | 12 | 63 | ? | 13 | 63 | 79.3 | 9 | 63 | 85.7 |
| DSJC500_9 | 47 | 161 | ? | 59 | 161 | 63.3 | 43 | 160 | 73.1 |
| DSJR500_1c | 72 | 88 | ? | 78 | 88 | 11.3 | 70 | 88 | 20.4 |
| DSJR500_5 | 117 | 130 | ? | 119 | 130 | 8.4 | 103 | 130 | 20.7 |
| DSJC1000_1 | 6 | 26 | ? | 6 | 26 | 76.9 | 5 | 25 | 80 |
| DSJC1000_5 | 14 | 116 | ? | 15 | 116 | 87 | 10 | 114 | 91.22 |
| DSJC1000_9 | 55 | 301 | ? | 65 | 301 | 78.4 | 53 | 300 | 82.3 |
| latin_squ_10 | 90 | 129 | ? | 90 | 129 | 30.2 | 90 | 129 | 30.2 |
| le450_15a | 15 | 17 | 15 | 15 | 17 | 11.7 | 15 | 17 | 11.7 |
| le450_15b | 15 | 17 | 15 | 15 | 17 | 11.7 | 15 | 16 | 6.2 |
| le450_15c | 15 | 24 | 15 | 15 | 24 | 37.5 | 13 | 23 | 43.4 |
| le450_15d | 15 | 23 | 15 | 15 | 23 | 34.7 | 13 | 23 | 43.4 |
| le450_25c | 25 | 28 | 25 | 25 | 28 | 10.7 | 20 | 28 | 28.5 |
| le450_25d | 25 | 28 | 25 | 25 | 28 | 10.7 | 21 | 27 | 22.2 |
| le450_5a | 5 | 9 | 5 | 5 | 9 | 44.4 | 5 | 9 | 44.4 |
| le450_5b | 5 | 9 | 5 | 5 | 9 | 44.4 | 5 | 9 | 44.4 |
| le450_5d | 5 | 10 | 5 | 5 | 10 | 50 | 5 | 8 | 37.5 |
| queen10_10 | 10 | 12 | ? | 10 | 12 | 16.6 | 10 | 12 | 16.6 |
| queen11_11 | 11 | 14 | 11 | 11 | 14 | 21.4 | 11 | 13 | 15.38 |
| queen12_12 | 12 | 15 | ? | 12 | 15 | 20 | 12 | 14 | 14.2 |
| queen13_13 | 13 | 16 | 13 | 13 | 16 | 18.7 | 13 | 15 | 13.3 |
| queen14_14 | 14 | 17 | ? | 14 | 17 | 17.6 | 14 | 17 | 17.6 |
| queen15_15 | 15 | 18 | ? | 15 | 18 | 16.6 | 15 | 18 | 16.6 |
| queen16_16 | 16 | 20 | ? | 16 | 20 | 20 | 16 | 19 | 15.7 |
| queen9_9 | 9 | 11 | 10 | 9 | 11 | 18.1 | 9 | 10 | 10 |
| myciel6 | 2 | 7 | 7 | 5 | 7 | 28.5 | 2 | 7 | 71.4 |
| myciel7 | 2 | 8 | 8 | 5 | 8 | 37.5 | 2 | 8 | 75 |
| abb313GPIA | 8 | 10 | ? | 8 | 10 | 20 | 6 | 10 | 40 |
| ash958GPIA | 3 | 5 | ? | 4 | 5 | 20 | 3 | 5 | 40 |
| 1-Insertions_5 | 2 | 6 | ? | 4 | 6 | 33.3 | 2 | 6 | 66.6 |
| 1-Insertions_6 | 2 | 7 | ? | 4 | 7 | 42.8 | 2 | 7 | 71.4 |
| 2-Insertions_4 | 2 | 5 | ? | 4 | 5 | 20 | 2 | 5 | 60 |
| 2-Insertions_5 | 2 | 6 | ? | 3 | 6 | 50 | 2 | 6 | 66.6 |
| 3-Insertions_4 | 2 | 5 | ? | 3 | 5 | 40 | 2 | 5 | 60 |
| 3-Insertions_5 | 2 | 6 | ? | 3 | 6 | 50 | 2 | 6 | 66.6 |
| 4-Insertions_3 | 2 | 4 | 4 | 3 | 4 | 25 | 2 | 4 | 50 |
| 4-Insertions_4 | 2 | 5 | ? | 3 | 5 | 40 | 2 | 5 | 60 |
| 1-FullIns_5 | 3 | 6 | ? | 4 | 6 | 33.3 | 3 | 6 | 50 |
| 2-FullIns_4 | 4 | 6 | ? | 5 | 6 | 16.6 | 4 | 6 | 50 |
| 2-FullIns_5 | 4 | 7 | ? | 5 | 7 | 28.5 | 4 | 7 | 42.8 |
| 3-FullIns_4 | 5 | 7 | ? | 6 | 7 | 14.2 | 5 | 7 | 28.5 |
| 3-FullIns_5 | 5 | 8 | ? | 6 | 8 | 25 | 5 | 8 | 37.5 |
| 4-FullIns_4 | 6 | 8 | ? | 7 | 8 | 12.5 | 6 | 8 | 25 |
| 4-FullIns_5 | 6 | 9 | ? | 6 | 9 | 33.3 | 6 | 9 | 33.3 |
| 5-FullIns_4 | 7 | 9 | ? | 8 | 9 | 11.1 | 7 | 9 | 22.2 |
| wap01 | 41 | 46 | ? | 41 | 46 | 10.8 | 39 | 48 | 18.7 |
| wap02 | 40 | 45 | ? | 40 | 45 | 11.1 | 39 | 46 | 15.21 |
| wap03 | 40 | 56 | ? | 40 | 56 | 28.5 | 40 | 55 | 27.27 |
| wap04 | 40 | 50 | ? | 40 | 50 | 20 | 20 | 48 | 58.3 |
| wap05 | 50 | 51 | ? | 50 | 51 | 1.9 | 27 | 51 | 47 |

Table 8 (*continued*)

| Problem | *n_cli* | $\hat{\chi}$ | $\chi$ | *BC-Col* | | | Dsatur | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | Lower | Upper | %Gap | Lower | Upper | %Gap |
| wap06 | 40 | 44 | ? | 40 | 44 | 9 | 33 | 45 | 26.6 |
| wap07 | 40 | 46 | ? | 40 | 46 | 13 | 23 | 46 | 50 |
| wap08 | 40 | 47 | ? | 40 | 47 | 14.8 | 23 | 45 | 48.8 |
| qg_order40 | 40 | 42 | 40 | 40 | 42 | 4.7 | 40 | 42 | 4.7 |
| qg_order60 | 60 | 63 | 60 | 60 | 63 | 4.7 | 60 | 63 | 4.7 |

be solved within the 2 h time limit imposed. No obvious explanation seem to exist for that behavior. Overall, *BC-Col* performance for that type of instance was very good with some instances being solved to proven optimality at the root node of the enumeration tree.

As a summary, Table 5 gives, for each algorithm, the average CPU time for those instances solved within the prescribed time limit. It also indicates the percentage of the instances that could not be solved within that time limit. Here, it is worth mentioning the low percentage of instances that could not be solved by *BC-Col* (as compared to Dsatur). Likewise, one should also point out the good performance of *BC-Col*, which is reflected by the average CPU times quoted.

Next, we present results for the DIMACS instances. For the DIMACS instances in Table 6, the lower and upper bounds obtained with our initial heuristics turned out to be the same. Therefore, no need exists for applying Branch-and-Cut to solve these instances.

Table 7 identifies those DIMACS instances that could be solved by *BC-Col* and/or Dsatur within the prescribed CPU time limit. Finally, Table 8 identifies those instances for which the time limit was exceeded. Entries for that table give the lower and the upper bounds obtained and the corresponding percentage gap between these bounds.

Out of the 110 DIMACS instances, 28 were solved by the initial *BC-Col* heuristics alone. Twenty instances were solved, within the prescribed time limit, by resorting to Branch-and-Cut. Finally, 62 instances could not be solved within the prescribed time limit. However, for 30 of these 62 instances, initial lower bounds were improved by *BC-Col*. Finally, the average percentage gap for *BC-Col* was 29.59% while the corresponding figure for Dsatur was 41.77%.

## 5. Concluding remarks

The algorithm implemented in this study is capable of solving graph coloring instances that are out of the reach of Dsatur. For many of the instances tested, Dsatur tends to find an optimal solution very early on in the enumeration process. It requires, however, far too much CPU time to provide an optimality certificate. On the other hand, Branch-and-Cut proved capable of obtaining optimality certificates faster than Dsatur. Generally speaking, the improvement *BC-Col* produces over the initial lower bounds allows (an eventual) optimality of the initial upper bounds to be quite effectively proven. Moreover, for those instances that could not be solved within the prescribed time limit, *BC-Col* tends to reduces significantly the initial lower and upper bound gaps (given, respectively, by *n_cli* and $\hat{\chi}$). Another advantage of the algorithm proposed here is that it tends to produce, in general, good quality lower bounds.

Our results suggest that *BC-Col* is a promising exact solution algorithm for *GCP*. Furthermore, there still exists a good potential for improving that algorithm. For instance, one could probably design more effective separation algorithms than the ones used here. Additionally, new families of strong valid inequalities may be eventually characterized and incorporated into *BC-Col*. Finally, some additional pruning strategy may prove more effective than the one currently used. We plan to address each of these points in the our future research in the area.

## Acknowledgements

We would like to thank Abilio Lucena for his constructive comments. We are also indebted to two anonymous referees whose remarks have helped us in improving the paper.

## References

[1] K. Aardal, A. Hipolito, C. van Hoesel, B. Jansen, C. Roos, T. Terlaky, A Branch-and-Cut algorithm for the frequency assignment problem, Research Memorandum 96/011, Maastricht University, 1996.
[2] D. Applegate, R. Bixby, V. Chvátal, W. Cook, On the solution of traveling salesman problems, Documenta Mathematica Journal der Deutschen Mathematiker-Vereinigung, International Congress of Mathematicians (1998) 645–656.
[3] E. Balas, S. Ceria, G. Cornuéjols, G.G. Pataki, Polyhedral methods for the maximum clique problem, in: D. Johnson, M. Trick (Eds.), Cliques, Coloring, and Satisfiability, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 26, 1996, pp. 11–27.
[4] F. Barahona, L. Ladányi, Branch-and-Cut Based on the volume algorithm: Steiner trees in graphs and max cut, IBM Research Report RC22221, 2001.
[5] U. Blasum, W. Hochstättler, Application of the Branch and Cut method to the vehicle routing problem, Zentrum für Angewandte Informatik Köln Technical Report zpr2000-386, 2000.
[6] D. Brélatz, New methods to color the vertices of a graph, Comm. ACM 22 (1979) 251–256.
[7] P. Coll, J. Marenco, I. Méndez-Díaz, P. Zabala, Facets of the graph coloring polytope, Ann. Oper. Res. 116 (2002) 79–90.
[8] CPLEX Linear Optimization 6.0 with mixed integer & barrier solvers, ILOG, 1997–1998.
[9] G.J. Chaitin, M. Auslander, A.K. Chandra, J. Cocke, M.E. Hopkins, P. Markstein, Register allocation via coloring, Computer Languages 6 (1981) 47–57.
[10] J. Culberson, Graph coloring bibliography, <http://web.cs.ualberta.ca/∼joe>.
[11] D. de Werra, Heuristics for graph coloring, Computing 7 (1990) 191–208.
[12] DIMACS Instances, <http://mat.gsia.cmu.edu/COLOR02>.
[13] F. Glover, M. Parker, J. Ryan, Coloring by tabu branch and bound, in: D. Johnson, M. Trick (Eds.), Coloring Cliques Coloring and Satisfiability, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 26, 1996, pp. 285–308.
[14] A. Herz, D. de Werra, Using tabu search techniques for graph coloring, Computing 39 (1987) 345–351.
[15] T. Jensen, B. Toft, Graph coloring problems, Wiley-Interscience, Series in Discrete Mathematics and Optimization, Wiley, New York, 1995.
[16] D. Johnson, M. Trick (Eds.), Cliques, Coloring, and Satisfiability, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 26, 1996.
[17] M. Junger, S. Thienel, The ABACUS system for Branch-and-Cut-and-Price algorithms in integer programming and combinatorial optimization, Software Practice and Experience 30 (11) (2000) 1325–1352.
[18] R. Karp, Reducibility among combinatorial problems, in: R. Miller, J. Thatcher (Eds.), Complexity of Computer Computations, 1972, pp. 85–104.
[19] A. Koster, Frequency Assignment, Models and Algorithms, Ph.D. Thesis, Universiteit Maastricht, 1999.
[20] M. Kubale, B. Jackowski, A generalized implicit enumeration algorithm for graph coloring, Commun. ACM 28 (4) (1985) 412–418.
[21] F.T. Leighton, A graph coloring algorithm for large scheduling problems, J. Res. Natl. Bur. Standards 84 (1979) 489–506.
[22] C. Mannino, A. Sassano, An exact algorithm for the maximum stable set problem, Comput. Optimization and Applications 3 (1994) 243–258.
[23] J. McHugh, Algorithmic Graph Theory, Prentice-Hall, Englewood Cliffs, NJ, 1990.
[24] A. Mehrotra, M. Trick, A column generation approach for graph coloring, INFORMS J. Comput. 8 (4) (1996) 344–353.
[25] I. Méndez-Díaz, Problema de Coloreo de Grafos-Un Estudio Poliedral y un Algoritmo Branch-and-Cut, Tesis Doctoral, Universidad de Buenos Aires, Argentina, 2003.
[26] I. Méndez-Díaz, P. Zabala, A polyhedral approach for graph coloring, Electronics Notes in Discrete Mathematics 7 (2001).
[27] I. Méndez-Díaz, P. Zabala, Polyhedral results for graph coloring, TR 02-001 Departamento de Computación, Universidad de Buenos Aires, 2002.
[28] G. Nemhauser, G. Sigismondi, A strong cutting plane/branch-and-bound algorithm for node packing, J. Oper. Res. Soc. 43 (5) (1992) 443–457.
[29] G. Nemhauser, L. Wolsey, Integer and Combinatorial Optimization, Wiley, New York, 1988.
[30] M.W. Padberg, On the facial structure of set packing polyhedral, Math. Prog. 5 (1973) 199–215.
[31] T.J. Sager, S. Lin, A pruning procedure for exact graph coloring, ORSA J. Comput. 3 (3) (1991) 226–230.
[32] A. Sassano, On the facial structure of the set covering polytope, Math. Prog. 44 (1989) 181–202.
[33] E. Sewell, An improved algorithm for exact graph coloring, in: D. Johnson, M. Trick (Eds.), DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 26, 1996, pp. 359–373.
[34] L. Wolsey, Integer Programming, Wiley, New York, 1998.