A Branch-and-Cut Algorithm for Graph Coloring *

Isabel Méndez Díaz - Paula Zabala
e-mail: {imendez, pzabala}@dc.uba.ar

Departamento de Computación
FCEyN - Universidad de Buenos Aires - Argentina

Abstract

In a previous work, we proposed a new integer programming formulation for the graph coloring problem which, to a certain extent, avoids symmetry. We studied the facet structure of the 0/1-polytope associated with it. Based on these theoretical results, we present now a Branch-and-Cut algorithm for the graph coloring problem. Our computational experiences compare favorably with the well-known exact graph coloring algorithm DSATUR.

Keyword: Graph Coloring; Integer Programming; Branch-and-Cut algorithms

1 Introduction

Given an undirected graph $G = (V, E)$ with V the set of vertices and E the set of edges, let $\mid V \mid= n$ and $\mid E \mid= m$. A *coloring* of G is an assignment of colors to each vertex such that the endpoints of any edge have different colors. A *k-coloring* of G is a coloring that uses k colors. The *chromatic number* of G is the smallest number of colors needed to color G and it is denoted by $\chi(G)$. The graph coloring problem *(GCP)* is to determinate $\chi(G)$. *GCP* arises in many applications such as scheduling, timetabling, electronic bandwidth allocation and sequencing.

GCP is known to be NP-hard for arbitrary graphs [12]. Even though, the practical importance of the problem makes necessary to devise algorithms with acceptable computational times for solving medium to moderate instances arising in real-world applications. A lot of work has been done in order to develop efficient algorithms, mainly by using heuristic techniques. The most usual approach is to find a partial coloring on a small subgraph that is extended vertex by vertex until the whole graph is colored. Also metaheuristic techniques like simulated annealing and tabu search were developed for *GCP* [7, 8, 10]. Relatively few methods for solving the problem exactly can be found in the literature. A comprehensive list of papers about coloring algorithms can be found in http://web.cs.ualberta.ca/ joe/Coloring.

An important number of exact methods are based on implicit enumeration. DSATUR is one of the most well-known exact algorithms. This vertex sequential algorithm was developed by Brelaz [3]. DSATUR is an implicit enumeration algorithm where each node of the tree correspond to a partial coloration of the graph. If UB is an upper bound of the number of colors required for the graph, a node using at least UB colors can be fathomed. Otherwise, a branching rule is applied to generate new nodes by choosing an uncolored vertex i. If k is the number of colors used by the partial coloring, for each feasible color from $1, \ldots, k$ and color $k + 1$, a new node is created assigning it to i. The algorithm terminates when there are no nodes left. The branch vertex i is chosen as the vertex adjacent to the largest number of differently colored vertices. In case of tie, a vertex with highest degree in the uncolored subgraph is chosen. This dynamic reordering of the vertices plays an important role to reduce the number of nodes of the search tree. The node selection strategy used by DSATUR is depth-first search. Alternative selection vertex strategies were proposed by other authors (see for instance, [13, 20, 21]).

In spite of *GCP* is related with the maximum clique and the maximum stable set problems and these are also NP-hard problems, instances of *GCP* seem more *difficult* to be solved exactly. Maximum clique and maximum stable set instances of hundreds of vertices can be solved within a reasonable amount of time [2, 8, 14, 17], however exact known algorithms for *GCP* are inefficient on graphs with as few as seventy vertices.

Like most optimization problems on graphs, *GCP* can be formulated as a linear integer programming problem. LP-based Branch-and-Cut algorithms are currently the most important tool to deal with these models computationally. However, the amount of research effort spend to solve the *GCP* by this method is not comparable with what has been dedicated to other problems, like TSP or maximum stable set.

*This research was partially supported by UBACYT Grant X036 and PID CONICET Grant 644/98

Few work using Branch-and-Cut techniques is found in the literature of the graph coloring. In [1], Aardal et al. propose a Branch-and-Cut algorithm for the frequency assignment problem using a vertex packing formulation. *GCP* can be thought as a special case of it. Using a different approach to solve linear integer problems, Mehrotra and Trick [15] developed a column generation algorithm based on the classical independent set formulation.

The performance of a Branch-and-Cut algorithm depends on a combination of many factors. Preprocessing, search and branching strategies, lower and upper bounds, LP-relaxation and cutting planes are the main components to take into account in an implementation. There are general methodologies, but those that take advantage of the particular structure of each problem have proved to be the most successful. In this sense, the use of cutting planes coming from a polyhedral study of the feasible solution set allowed to solve instances of hard combinatorial optimization problems [2, 17, 18, 19].

Since in *GCP* the colors are indistinguishable, there are many symmetrical colorings with the same number of colors, i.e. permutations of the colors define an exponential number of equivalents colorings. If the integer programming formulation exhibits the same property, it turns out that the Branch-and-Cut method has a poor performance even in small instances. The trouble comes from the fact that many subproblems in the enumeration tree have the same optimal value because the indistinguishability of the variables.

In [16] we presented an approach of a new integer programming formulation that reduces the number of symmetrical feasible solutions and we derivated families of facet-defining inequalities. Now, we propose a Branch-and-Cut algorithm based on these theoretical results. We develop separation procedures for some of the inequalities and introduce strategies that reject symmetry on the generation phase of the search tree.

We close this section by introducing all the notation and definitions used throughout the paper.

Let $V' \subset V$, $G[V'] = (V', E')$ is the induced subgraph of G by V' if $E' = \{\{u,v\} : \{u,v\} \in E$ and $u, v \in V'\}$. $V' \subset V$ is a clique in G if $\forall u, v \in V', \{u,v\} \in E$. $V' \subset V$ is a stable set or independent set in G if $\forall u, v \in V'$, $\{u,v\} \notin E$. A clique (stable set) K in G is maximal if there is no clique (stable set) $K' \neq K$ in G with $K \subset K'$. The stability number of G, $\alpha(G)$, is the maximum size of a independent set in G. A clique partition of the graph G is a partition $(K_1, \ldots, K_k)$ of V such that K_i is a clique in 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 is a cycle if in addition $\{v_1, v_k\} \in E$. The neighborhood of v is $N(v) = \{u : u \in V$ and $\{u,v\} \in E\}$. A graph G is bipartite if $\chi(G) \leq 2$. The rest of the paper is organized as follows. In Section 2, we present the coloring polytope and some polyhedral results. We describe the details of the Branch-and-Cut algorithm in Section 3. Section 4 contains our computational results on the DIMACS benchmark and random graphs. The paper closes with final remarks in Section 5.

2 The coloring polytope

The classical IP formulation for *GCP* is:

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

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

$$x_{ij} + x_{kj} \leq w_j \quad \forall\, \{i,k\} \in E,\ 1 \leq j \leq n \tag{2}$$

$$x_{ij} \in \{0,1\} \quad \forall\, i \in V,\ 1 \leq j \leq n \qquad w_j \in \{0,1\} \quad 1 \leq j \leq n$$

where $x_{ij} = 1$ if color j is assigned to vertex i and $x_{ij} = 0$ otherwise. The n binary variables w_j $(1 \leq j \leq n)$ indicate whether color j is used in some vertex, i.e. $w_j = 1$ if $x_{ij} = 1$ for some vertex i. The polytope associated to this formulation is denoted by SCP.

Constraints (1) assert that each vertex must receive exactly one color, and constraints (2) say that every pair of adjacent vertices must not share the same color and that $w_j = 1$ when some vertex has color j.

In [4] we studied the polytope SCP of this classical formulation. We developed a Branch-and-Cut using these results, but it was not very successful. We believe this is due to the existence of too many symmetrical solutions. In [16] we proposed three new IP formulations that try to avoid this problem. In what follows, we summarize polyhedral results for *CP*, the polytope associated to one of them.

The coloring polytope is

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

.

The added constraints state that color j can be assigned to a vertex only if color $j-1$ has been already assigned. For any feasible k-coloring, its symmetrical k-colorings that use colors with label greater than k are eliminated.

We now point out the main properties of the polytope associated to this formulation. Details about these results can be found in [16], as well as other facets that we do not use in this Branch-and-Cut code.

We know that the set of vertices colored with the same color is a stable set, so its size is less than or equal to the stability number. Besides that, the coloring properties of substructures of a graph give information for the coloring of the whole graph. Combining both ideas the next result follows .

Proposition 2.1 *Let $G' = G[V']$, $V' \subset V$, be an induced graph of G. Consider $\alpha(G')$ y $\alpha(G)$ the stability number of G' and G respectively. Then*

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

is a valid inequality for $\mathcal{CP}$. If

- $\alpha(G') < \alpha(G)$.
- $\forall v \in V \backslash V'$, $G[V' \cup \{v\}]$ *has a* $(\alpha(G') + 1)$*-independent set.*
- *exists I, maximum independent set of G', such that $G[V \backslash I]$ is not a clique.*
- *there is some $\chi(G)$-coloring on the face.*

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

These inequalities become useful on substructures with known stability number. In our algorithm, we use the ones that correspond to cliques and cycles. The above result on these special subgraphs is pointed out in the next two propositions. In [16] we also studied properties of paths and complement of cycles.

Proposition 2.2 *Let $1 \leq j_0 \leq n-1$ and K be a maximal clique of G. The **clique** inequality,*

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

is a facet-defining inequality for $\mathcal{CP}$.

Proposition 2.3 *Let C_k be a cycle of G of size k and $1 \leq j_0 \leq n - \lfloor k/2 \rfloor$. The **cycle** inequality*

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

is a valid inequality for $\mathcal{CP}$.

Using similar arguments on the maximum stable set of the neighborhood of a vertex, we derivate the following result:

Proposition 2.4 *Let $v \in V$ and $r = \alpha(G[N(v)])$ with $r \geq 2$. The **neighborhood** inequality,*

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

is a valid inequality for $\mathcal{CP}$.

Let us consider now ***clique*** inequalities. If we add up a set of these constraints, clearly we obtain a valid inequality that is dominated by each of the constraints used in this operation. But, if we can strengthen it while preserving its validity, we will obtain a new inequality that is not dominated by any ***clique*** constraint. To strengthen the inequality we use the trivial fact that no more than k colors are used to color a set of size k.

This concept allows us to derivate the following inequalities:

Proposition 2.5 *Let $P_k = v_1, ..., v_k, k \geq 3$, be a path and consider $\{c_1, ..., c_k\}$ a set of k colors ($c_k \geq c_i \quad i = 1, \ldots, k-1$). The **path** inequality*

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

is a valid inequality for $\mathcal{CP}$.

Proposition 2.6 *Let $\{v_1, \ldots, v_p\}$ be a clique of size p of G, k be a color $p \leq k \leq n-1$ and $Col \subseteq \{1, \ldots, k-1\}$ with $|Col| = p-1$. Then the **p-color clique** inequality*

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

is a valid inequality for $\mathcal{CP}$.

Finally, we mention an inequality that follows from the way we eliminate symmetrical solutions. If a color j_0 is not used in a feasible solution, colors with label greater than j_0 are not used either. Besides that, any vertex does not use more than one color. Both observations are put together in the following result.

Proposition 2.7 *The **block color** inequality*

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

is a valid inequality of $\mathcal{CP}$.

3 Branch-and-Cut algorithm

In this section we describe how the above theoretical results are used to implement a Branch-and-Cut algorithm.

Given an integer programming problem, the idea of a Branch-and-Cut method is recursively partition the solution set into subsets and solve the problem over each subset. This procedure generates an enumeration tree where offsprings of a node correspond to the partition of the set associated with the parent node. In each node of the tree, a linear relaxation of the problem is considered by dropping integrality requirements and adding valid inequalities which cut off the fractional solution. To reduce the number of nodes of the tree, it is important to have good lower and upper bounds, good rules to partition the feasible set, good strategies to search on the tree and a good strengthening of the linear relaxations. In what follows we describe the different aspects of our implementation that take into account this factors.

3.1 Preprocessing

Since even for moderately sized coloring problems, the number of variables and inequalities is rather large in our model. We would like to use preprocessing techniques to eliminate variables and constraints in order to keep the linear program reasonably sized.

A simple heuristic algorithm finds a maximal clique which size, *n_cli*, is used as a lower bound of the chromatic number. All the variables related to these vertices are fixed in the model. Then, we eliminate the vertices having a no-adjacent vertex on the clique that is adjacent to any vertex of its neighborhood. Finally, all vertices with degree less than *n_cli-1* are deleted. From any optimal coloring of the new graph follows an optimal coloring of the original graph.

Besides that, we generate a feasible initial coloring applying a partial enumeration heuristic based on DSATUR. This solution gives an upper bound of the chromatic number (denoted by $\hat{\chi}$) and allows us to eliminate variables of the model.

A limit of 5 seconds is specified for both heuristics but in most instances the complete run time was less than 20% of this limit.

3.2 Improving the linear program relaxation

The model has $m\hat{\chi}$ constraints (2). Since this size is difficult to handle for large and dense graphs, we replace the constraints (2) by

$$\sum_{i\in N(k)} x_{ij} + \mu x_{kj} \leq \mu w_j$$

where $N(k) = \{i \in V : i \text{ is adjacent to } k\}$ and μ is the cardinal of a clique partition of $N(k)$. In this way we handle $n\hat{\chi}$ constraints instead of $m\hat{\chi}$.

In spite of this procedure relaxes the polytope, the computational experience shows it works better than the original formulation. Note that these constraints are a weak version of the ***neighborhood*** inequalities.

Finally, in order to strength the linear programming relaxation, we add the following constraints

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

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

3.3 Branching rules

In our computational experiments we try various branching strategies. The classical rule of branching on a fractional variable, where it is set to 1 in one subproblem and set to 0 in the other is very asymmetrical. The generated search tree is unbalanced because setting a variable to 1 means to fix a color to a vertex, while setting it to 0 means that one color is not considered to be assigned to the vertex. We did not have good success with this strategy.

To avoid this behavior, we first choose a vertex of the graph. Then, for each feasible color for the vertex out of the used colors in the subproblem, a new subproblem is created. In addition, a subproblem is created with the vertex receiving the next color.

Following the idea of Brélatz we choose a fraccionable vertex adjacent to the largest number of differently colored vertices. In case of ties, we consider two alternative tie-breaking rule:

- **VB1:** the vertex with highest degree in the uncolored subgraph
- **VB2:** the vertex that produces the largest decrease in the number of colors available for the remaining uncolored vertices

The first rule is due to Brélatz [3] and the second is a modification proposed by Sewell [21].

The above branching strategies specify how to split the set of feasible solutions of the current subproblem. We have to determine in what order the subproblems will be examined. We use a depth first search rule in choosing the node to evaluate, but we consider four different ways to add the new nodes of the tree to the list of active subproblems:

- **O1:** by increasing order of color labels
- **O2:** first the new color and then by increasing order of color labels
- **O3:** by increasing order of the number of vertices that have been already colored with each color.
- **O4:** by decreasing order of the number of vertices that have been already colored with each color.

For "small" graphs, the complete enumeration of feasible colorings is more efficient than a Branch-and-Cut algorithm. Then, when the number of still uncolored vertices is "small", it was useful to implement the implicit enumeration scheme. This level is a parameter of our implementation. We fix it to 60 for graphs with more than 60 vertices, otherwise the complete enumeration begins on level 2 of the Branch-and-Cut tree.

3.4 Cutting plane generation

In a Branch-and-Cut framework, some key decisions have to be taken: when cutting planes need to be generated, how many iterations of a cutting plane algorithm and how many cuts should be generated at each iteration. An appropriate balance between branching and cutting is necessary because small enumeration tree do not always correspond to smaller computing times. We use the following input parameters: the skip factor (number of nodes of the enumeration tree that are enumerated before cutting plane phase is applied), rounds per node (iterations of a cutting plane algorithm) and the maximal number of cuts added per iteration.

The problem of identifying violated inequalities is called the separation problem. We now describe the identification procedure of violated cutting planes that are implemented in our Branch-and-Cut code.

3.4.1 Clique and p-color clique inequalities

Initially, we considered the alternative of generating a list of cliques before starting the algorithm. Then, by a sequential checking of the list, we looked for a violated inequality. This is a classical approach but our preliminary computations showed it was not good enough. The ***clique*** inequalities play an important role on our algorithm and this procedure did not find many violated inequalities.

So, we developed a simple greedy heuristic procedure. For each color j_0, the greedy criterion is to go for violated ***clique*** inequalities and it makes sense to do so by considering the list of fractional and zero variables in order decreasing $x^*_{ij_0}$ value, where x^* denotes the current fractional solution.

If x_{ij_0} is a fractional variable, we initialize a clique with vertex i. Then, it will be grown into a bigger clique trying to add other vertices follow the order of x^*. We do several trials bounded by an input parameter. In trial k, we choose the fractional variable $x_{i'j_0}$ such that vertex i' is the *k-th* adjacent vertex to i in the list. We add this vertex to the clique and then look in order in the rest of the list.

To avoid any additional computational effort, the clique found is also used to try a violated ***p-color clique*** inequality. For each color j, with $1 \leq j \leq j_0 - 1$, we compute S_j where $S_j = \sum_{i \in clique} x_{ij} - w_j$. If nc is the clique size, to have more chances to find a violated inequality, we choose the first $nc - 1$ colors in order of decreasing S_j values.

3.4.2 Block color inequalities

The ***block color*** inequalities are handled by brute-force. We enumerate all n^2 inequalities and find those that are violated by the fractional current solution.

3.4.3 Path inequalities

For each fractional variable w_k, we associate to each edge $(u, v) \in E$, the weight $c_{uv} = max_{j=1,\dots,k-1}\{x_{uj} + x_{vj} - w_j\} + \sum_{j=k}^{n}(x_{uj} + x_{vj})$. Using a greedy procedure, we compute for each vertex $v \in V$, the weightest path in G. A path with weight greater than w_k corresponds to a violated ***path*** inequality. To avoid inequalities with similar support, the procedure has an upper bound to the number of times a vertex belongs to a path.

4 Computational experiments

We report in this section the computational experience with our Branch-and-Cut code. The code was implemented in C++ using the ABACUS framework [11] and CPLEX 6.0 LP solver [6]. We have performed the experiments on a Sun ULTRA workstation and the times are reported in seconds.

In our computational experiments, we use DIMACS benchmark instances drawn from http://mat.gsia.cmu.edu/COLOR02 and random graphs. $G(n, p)$ is a random graph of n vertices and an edge between each pair of vertices with independent probability p. This class of graphs is used extensively in testing graph coloring algorithms.

The Table 1 shows DIMACS instances. We give the number of vertices, the number of edges, the size of a maximal clique and the chromatic upper bound obtained by the initial heuristics. The last column corresponds to the chromatic number ("?" means unknown).

Problem	vertices	edges	n_cli	$\hat{\chi}$	χ	Problem	vertices	edges	n_cli	$\hat{\chi}$	χ
DSJC125_1	125	736	4	5	?	school1	385	19095	**14**	**14**	**14**
DSJC125_5	125	3891	9	20	?	school1_nsh	352	14612	**14**	**14**	**14**
DSJC125_9	125	6961	32	49	?	zeroin.i.1	211	4100	**49**	**49**	**49**
DSJC250_1	250	3218	4	9	?	zeroin.i.2	211	3541	**30**	**30**	**30**
DSJC250_5	250	15668	11	36	?	zeroin.i.3	206	3540	**30**	**30**	**30**
DSJC250_9	250	27897	37	88	?	anna	138	493	**11**	**11**	**11**
DSJC500_1	500	12458	5	15	?	david	87	406	**11**	**11**	**11**
DSJC500_5	500	62624	12	63	?	homer	561	1629	**13**	**13**	**13**
DSJC500_9	500	1124367	47	161	?	huck	74	301	**11**	**11**	**11**
DSJR500_1	500	3555	**12**	**12**	**12**	jean	80	254	**10**	**10**	**10**
DSJR500_1C	500	121275	72	87	?	games120	120	638	**9**	**9**	**9**
DSJR500_5	500	58862	117	131	?	miles1000	128	3216	41	42	42
DSJC1000_1	1000	49629	6	26	?	miles1500	128	5198	71	73	73
DSJC1000_5	1000	249826	14	116	?	miles250	128	387	**8**	**8**	**8**
DSJC1000_9	1000	449449			?	miles500	128	1170	**20**	**20**	**20**
fpsol2_i_1	496	11654	55	65	65	miles750	128	2113	**31**	**31**	**31**
fpsol2_i_2	451	8691	29	30	30	queen10_10	100	2940	10	12	?
fpsol2_i_3	425	8688	29	30	30	queen11_11	121	3960	11	14	11
inithx.i.1	864	18707	**54**	**54**	**54**	queen12_12	144	5192	12	15	?
inithx.i.2	645	13979	**31**	**31**	**31**	queen13_13	169	6656	13	16	13
inithx.i.3	621	13969	**31**	**31**	**31**	queen14_14	196	8372	14	17	?
latin_squ_10	900	307350	90	129	?	queen15_15	225	10360	15	18	?
le450_15a	450	8168	15	17	15	queen16_16	256	12640	16	20	?
le450_15b	450	8169	15	17	15	queen5_5	25	160	**5**	**5**	**5**
le450_15c	450	16680	15	24	15	queen6_6	36	290	**6**	**7**	**7**
le450_15d	450	16750	15	23	15	queen7_7	49	476	**7**	**7**	**7**
le450_25a	450	8260	**25**	**25**	**25**	queen8_12	96	1368	**12**	**12**	**12**
le450_25b	450	8263	**25**	**25**	**25**	queen8_8	64	728	8	9	9
le450_25c	450	17343	**25**	**28**	**25**	queen9_9	81	1056	9	11	10
le450_25d	450	17425	25	28	25	myciel3	11	20	2	4	4
le450_5a	450	5714	5	9	5	myciel4	23	71	2	5	5
le450_5b	450	5734	5	9	5	myciel5	47	236	2	6	6
le450_5c	450	9803	**5**	**5**	**5**	myciel6	95	755	2	7	7
le450_5d	450	9757	5	10	5	myciel7	191	2360	2	8	8
mulsol.i.1	197	3925	49	49	49	mug88_1	88	146	3	4	4
mulsol.i.2	188	3885	**31**	**31**	**31**	mug88_25	88	146	3	4	4
mulsol.i.3	184	3916	**31**	**31**	**31**	mug100_1	100	166	3	4	4
mulsol.i.4	185	3946	**31**	**31**	**31**	mug100_25	100	166	3	4	4
mulsol.i.5	185	3973	**31**	**31**	**31**	abb313GPIA	1557	46546	8	10	?

Problem	vertices	edges	n_cli	$\hat{\chi}$	χ	Problem	vertices	edges	n_cli	$\hat{\chi}$	χ
ash331GPIA	662	4185	3	4	?	2-FullIns_5	852	12201	4	7	?
ash608GPIA	1216	7844	3	4	?	3-FullIns_3	80	346	5	6	?
ash958GPIA	1916	12506	3	5	?	3-FullIns_4	405	3524	5	7	?
will199GPIA	701	6772	5	7	?	3-FullIns_5	2030	33751	5	8	?
1-Insertions_4	67	232	2	5	?	4-FullIns_3	114	541	6	7	?
1-Insertions_5	202	1227	2	6	?	4-FullIns_4	690	6650	6	8	?
1-Insertions_6	607	6337	2	7	?	4-FullIns_5	4146	77305	6	9	?
2-Insertions_3	37	72	2	4	4	5-FullIns_3	154	792	7	8	?
2-Insertions_4	149	541	2	5	4	5-FullIns_4	1085	11395	7	9	?
2-Insertions_5	597	3936	2	6	?	wap01	2368	110871	41	46	?
3-Insertions_3	56	110	2	4	4	wap02	2464	111742	40	45	?
3-Insertions_4	281	1046	2	5	?	wap03	4730	286722	40	56	?
3-Insertions_5	1406	9695	2	6	?	wap04	5231	294902	40	50	?
4-Insertions_3	79	156	2	4	?	wap05	905	43081	50	51	?
4-Insertions_4	475	1795	2	5	?	wap06	947	43571	40	44	?
1-FullIns_3	30	100	3	4	?	wap07	1809	103368	40	46	?
1-FullIns_4	93	593	3	5	?	wap08	1870	104176	40	47	?
1-FullIns_5	282	3247	3	6	?	qg_order30	900	26100	**30**	**30**	**30**
2-FullIns_3	52	201	4	5	?	qg_order40	1600	62400	40	42	40
2-FullIns_4	212	1621	4	6	?	qg_order60	3600	212400	60	63	60

Table 1: DIMACS Instances

4.1 Reducing the problem size

We start our computations with the reduction techniques described in Section 3. The removal of vertices is highly effective for DIMACS instances. The graph reduction is more important on graphs with low density even though there are instances of different densities. This procedure is useless on random graphs. It can be explained by the regular property of the vertex grades. Table 2 shows DIMACS instances before and after reduction. We report the density, the original number of vertices, the number of vertices after reduction, $\hat{n}$, and the percentage reduction. The CPU time for performing the reductions is insignificant in the total time.

Problem	%Dens.	n	$\hat{n}$	% Red.	Problem	% Dens.	n	$\hat{n}$	% Red.
DSJR500_1	3	500	109	78	anna	5	138	17	88
DSJR500_1C	97	500	410	18	david	11	87	11	87
DSJR500_5	47	500	491	2	homer	1	561	38	93
fpsol2_i_1	9	496	171	66	huck	11	74	11	85
fpsol2_i_2	9	451	164	64	jean	8	80	13	84
fpsol2_i_3	10	425	163	62	games120	9	120	119	1
inithx.i.1	5	864	115	87	miles1000	39	128	50	61
inithx.i.2	7	645	182	72	miles1500	63	128	85	34
inithx.i.3	7	621	172	72	miles250	5	128	15	88
latin_square_10	76	900	129	86	miles500	14	128	28	78
le450_15a	8	450	409	9	miles750	26	128	37	71
le450_15b	8	450	413	8	abb313GPIA	4	1557	1400	10
le450_25a	8	450	271	40	ash331GPIA	2	662	661	1
le450_25b	8	450	302	33	ash608GPIA	1	1216	1215	1
le450_25c	17	450	436	3	ash958GPIA	1	1916	1915	1
le450_25d	17	450	436	3	will199GPIA	3	701	697	1
mulsol.i.1	20	197	49	75	1-FullIns_3	22	30	21	30
mulsol.i.2	22	188	100	47	1-FullIns_4	14	93	63	32
mulsol.i.3	23	184	101	45	1-FullIns_5	8	282	189	33
mulsol.i.4	23	185	102	45	2-FullIns_3	15	52	40	23
mulsol.i.5	23	185	102	45	2-FullIns_4	7	212	160	25
school1	26	385	358	7	2-FullIns_5	3	852	640	25
school1_nsh	24	352	328	7	3-FullIns_3	11	80	65	19
zeroin.i.1	18	211	63	70	3-FullIns_4	4	405	325	20
zeroin.i.2	16	211	57	73	3-FullIns_5	2	2030	1625	20
zeroin.i.3	17	206	56	73	4-FullIns_3	8	114	84	26

Problem	% Dens.	n	$\hat{n}$	% Red.	Problem	% Dens.	n	$\hat{n}$	% Red.
4-FullIns_4	3	690	576	17	wap03	3	4730	4701	1
4-FullIns_5	1	4146	3456	17	wap04	2	5231	5204	1
5-FullIns_3	7	154	79	49	wap05	11	905	665	27
5-FullIns_4	2	1085	931	14	wap06	10	947	787	17
wap01	4	2368	1771	25	wap07	6	1809	1655	9
wap02	4	2464	2174	12	wap08	6	1870	1696	9

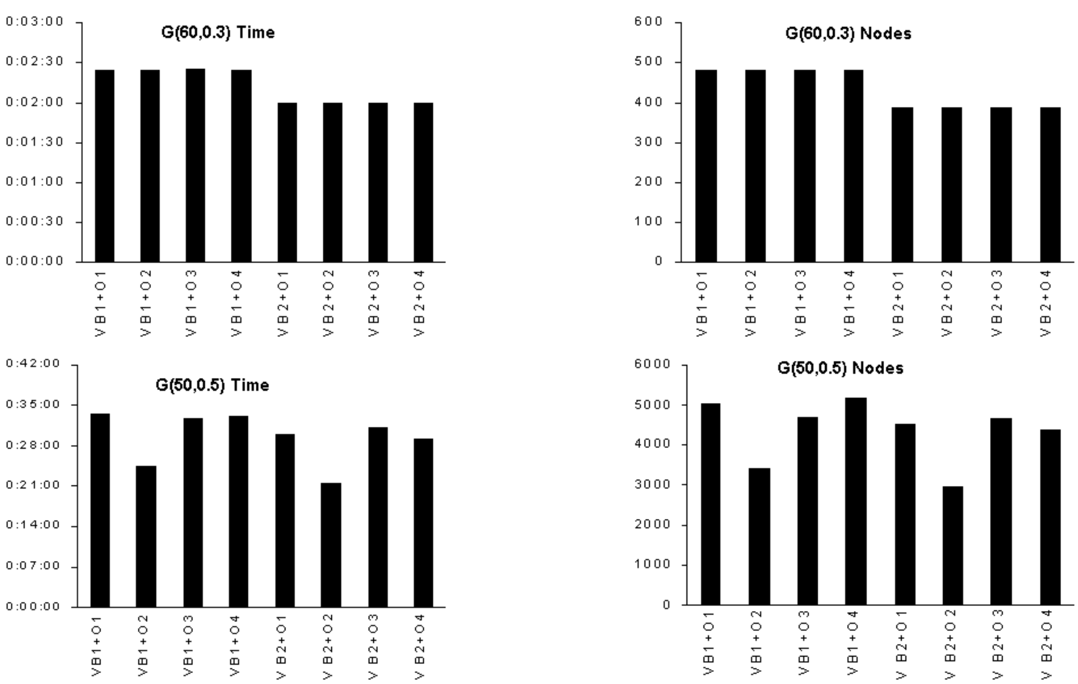
Table 2: Reduction

4.2 Branching strategies

The branching strategies can be combined with the four different ways to add the new nodes to the list of active subproblems. So, we have eight rules to generate and search on the tree.

Our experience shows that the branching rules performance does not change if we add cutting planes during the algorithm. So, to report the results as simple as possible, they were tested on a Branch-and-Bound version of our code. We run on random graphs with 50 and 60 vertices with edge probabilities 0.3, 0.5, 0.7 and 0.9. In Figure 1 we present the results over averages of 5 instances.

If we fix the order to add the nodes to list, **VB2** is generally better than **VB1**. In [21], Sewell proposed an enumerative algorithm using **VB2+O1** rule. He reports that this algorithm produces fewer subproblems (on average) than DSATUR algorithm (**VB1+O1**) but it requires more CPU time because the tie-breaking rule computation is more expensive. We think this is not our case because the percentage of the total CPU time used by it in a Branch-and-Bound scheme is not significative.



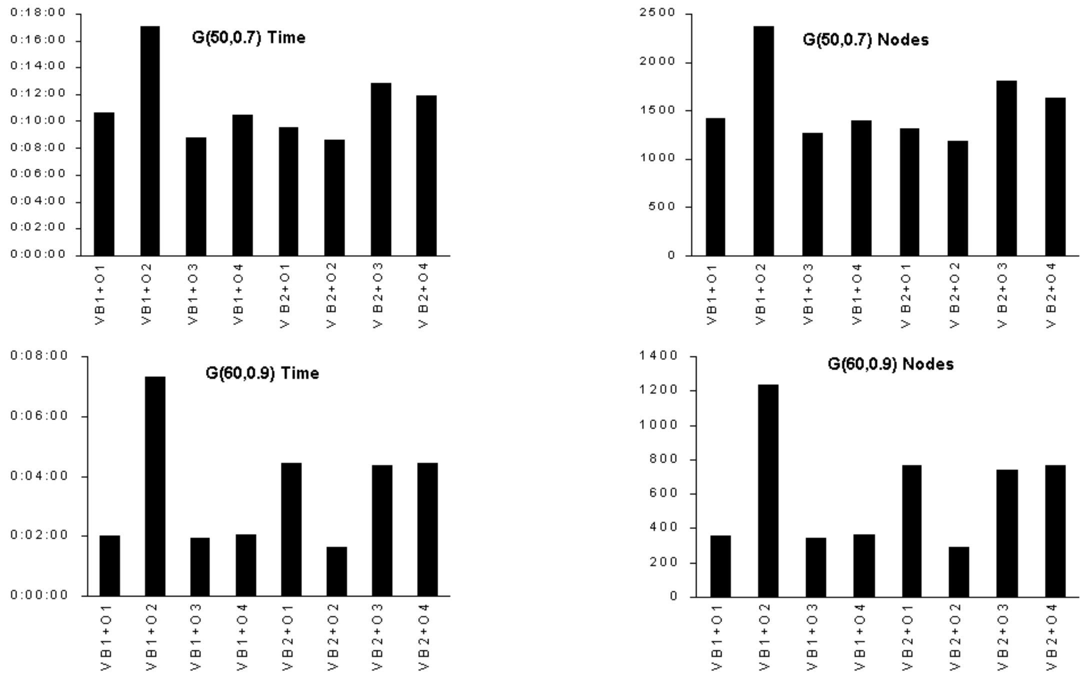


Figure 1: Branching Strategies

It is clear from the figures that the combination **VB2+O2** is the best. This conclusion follows from the number of subproblems as well as the CPU time.

To end the evaluation of branching strategies, we compare the classical 0-1 dichotomy variable selection with our worst and best combination strategy. Figure 2 confirms that the first one is not competitive.

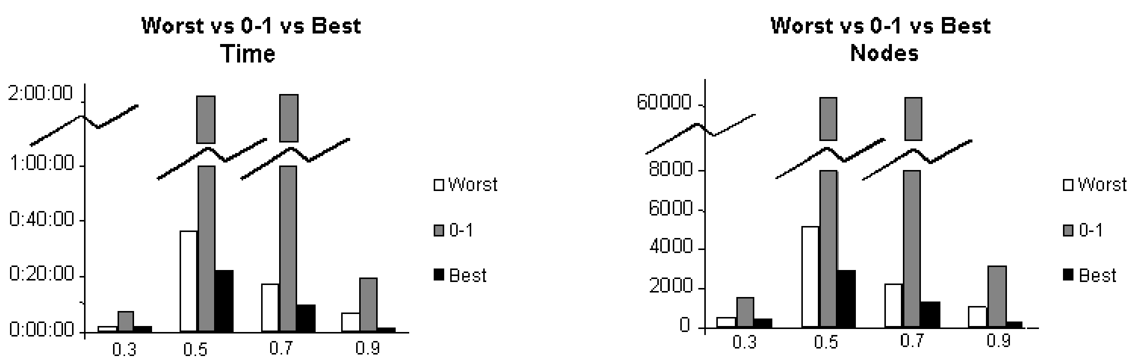


Figure 2: Worst vs 0-1 vs Best Strategies

4.3 Cutting planes

An indirect way of evaluating the quality of a cutting plane is to observe the increase produced in the lower bound when it is added to the LP-relaxation. Larger increases mean better constraints because they define deeper cuts in the relaxation polytope. However, a right balance between different aspects has to be considered.

If the added cuts are dense, they increase memory requirements and may slow down the solution of the LP's. Besides that, if the separation routine for a class of inequalities is computationally expensive in relation to the lower bound increase when they are added to the LP's, it is not worthwhile to include them in the algorithm.

We have conducted experiments to determine a good cut combination scheme. A pure cutting plane algorithm with each of the cut families combination was applied for 50 rounds on eight random instances of 125 vertices with low (less than 30%), eight with medium (between 40 and 60%) and eight with high (greater than 70 %) density. Tables 3, 4 and 5 reports the CPU time and the round where the best lower bound is achieved, the values of the column labeled ILP are the initial LP-value. The computing time is mainly spent within the LP-solver. We do not report separation procedure times, since they represent a small fraction of the total time (never more than 10%) even though it is worth to mention that ***cycle*** and ***p-color clique*** separation were the most expensive procedures. The references for each table row are:

C_1	clique	C_7	clique+path+p-color clique	C_{13}	clique+block color+path+cycle
C_2	clique+block color	C_8	clique+block color+path+p-color clique	$C14$	clique+block color+p-color clique+cycle
C_3	clique+path	C_9	clique+cycle	$C15$	clique+path+p-color clique+cycle
C_4	clique+p-color clique	C_{10}	clique+block color+cycle	$C16$	clique+block color+path+p-color clique+cycle
C_5	clique+block color+path	C_{11}	clique+path+cycle	$C17$	block color+path+p-color clique+cycle
C_6	clique+block color+p-color clique	C_{12}	clique+p-color clique+cycle		

	I1			I2			I3			I4		
	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP
	6	11	0	6	10	0	6	11	0	5	11	0.074
	LB	time	round	LB	time	round	LB	time	round	LB	time	round
C_1	1	108	9	1	85	3	1	95	9	2	187	11
C_2	1	108	9	1	85	3	1	96	9	2	187	11
C_3	1	108	9	1	85	3	1	95	9	2	202	10
C_4	1	109	9	1	86	3	1	96	9	2	221	10
C_5	1	108	9	1	85	3	1	95	9	2	203	10
C_6	1	109	9	1	87	3	1	96	9	2	221	10
C_7	1	109	9	1	84	3	1	96	9	2	260	10
C_8	1	109	9	1	84	3	1	96	9	2	230	11
C_9	1	142	9	1	266	4	1	131	9	2	208	10
C_{10}	1	142	9	1	266	4	1	131	9	2	209	10
C_{11}	1	141	9	1	265	4	1	130	9	2	236	11
C_{12}	1	142	9	1	265	4	1	130	9	2	234	10
C_{13}	1	142	9	1	265	4	1	130	9	2	237	11
C_{14}	1	142	9	1	266	4	1	130	9	2	233	10
C_{15}	1	142	9	1	268	4	1	131	9	2	261	9
C_{16}	1	142	9	1	268	4	1	131	9	2	261	9
C_{17}	0	27	1	1	64	5	0	23	1	1	48	1
	I5			I6			I7			I8		
	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP
	6	12	0	7	13	0	7	12	0	6	12	0.062
	LB	time	round	LB	time	round	LB	time	round	LB	time	round
C_1	2	347	23	1	131	10	1	154	12	2	213	12
C_2	2	348	23	1	131	10	1	154	12	2	213	12
C_3	2	406	23	1	134	10	1	155	12	2	178	11
C_4	2	423	24	1	134	10	1	155	12	2	204	11
C_5	2	406	23	1	131	10	1	153	12	2	210	11
C_6	2	424	24	1	134	10	1	155	12	2	204	11
C_7	2	407	22	1	132	10	1	153	12	2	194	9
C_8	2	407	22	1	133	10	1	153	12	2	194	9
C_9	2	407	26	1	119	8	1	195	13	2	250	13
C_{10}	2	407	26	1	119	8	1	195	13	2	250	13
C_{11}	2	424	21	1	120	8	1	195	13	2	285	11
C_{12}	2	451	24	1	120	8	1	195	13	2	290	12
C_{13}	2	425	21	1	120	8	1	195	13	2	240	11
C_{14}	2	451	24	1	120	8	1	195	13	2	290	12
C_{15}	2	447	22	1	120	8	1	195	13	2	261	11
C_{16}	2	447	22	1	120	8	1	195	13	2	262	11
C_{17}	1	129	3	0	26	1	0	21	1	1	38	1

Table 3: Cutting plane for low density graphs

These tables clearly show that the good performance of the cutting plane algorithm is mainly due to the presence of ***clique*** inequalities. If this family inequality is excluded, it is detrimental to the lower bound improvement.

In order to get a more direct comparasion, the Figure 3 gives a summary of the above results. For each cut combination, the figure shows the average over the eight instances of the ratio of the difference between the CPU time for this cut combination and the CPU time for the best cut combination for that instance over the best cut combination.

There is not a clear computational winner among the combinations considered. However, since combinations without ***p-color clique***, for low and medium density graphs, and ***cycle*** inequalities, for high density graphs, are generally superior, not only in CPU time but also in the number of rounds to achieve the same lower bound, we decide not to include these inequalities. The scheme using ***clique***, ***block color***

	I1			I2			I3			I4		
	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP
	8	16	0	8	15	0	8	16	0	8	16	0
	LB	time	round	LB	time	round	LB	time	round	LB	time	round
C_1	2	484	21	2	515	22	1	531	27	2	452	22
C_2	2	484	21	2	517	22	1	533	27	2	451	22
C_3	2	446	20	2	499	20	1	526	26	2	471	21
C_4	2	498	18	2	542	20	1	580	23	2	462	20
C_5	2	447	20	2	503	20	1	533	26	2	477	21
C_6	2	503	18	2	550	20	1	589	23	2	468	20
C_7	2	554	19	2	549	19	1	556	23	2	521	22
C_8	2	555	19	2	559	19	1	555	23	2	521	22
C_9	2	534	20	2	749	22	1	607	25	2	567	23
C_{10}	2	532	20	2	760	22	1	602	25	2	566	23
C_{11}	2	502	21	2	588	20	1	579	25	2	501	21
C_{12}	2	570	20	2	659	18	1	648	23	2	537	21
C_{13}	2	502	21	2	594	20	1	582	25	2	500	21
C_{14}	2	570	20	2	659	18	1	648	23	2	534	21
C_{15}	2	539	19	2	686	20	1	628	23	2	557	22
C_{16}	2	542	19	2	685	20	1	629	23	2	557	22
C_{17}	0	27	1	1	294	20	0	27	1	1	277	15
	I5			I6			I7			I8		
	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP
	7	16	0.057	9	20	0.271	11	24	0.667	11	24	0.698
	LB	time	round	LB	time	round	LB	time	round	LB	time	round
C_1	3	935	32	3	1118	25	4	2337	43	4	2244	39
C_2	3	930	32	3	997	27	4	2919	38	4	2148	35
C_3	3	954	34	3	917	23	4	3121	35	4	2268	38
C_4	3	1128	32	3	1434	18	4	2986	42	4	2691	37
C_5	3	954	34	3	1036	20	4	2440	41	4	2161	38
C_6	3	1131	32	3	1135	18	4	2896	41	4	2293	37
C_7	3	1293	34	3	1326	10	4	2365	38	4	2426	38
C_8	3	1294	34	3	1060	19	4	2693	30	4	2723	35
C_9	3	941	31	3	951	25	4	2690	30	4	2749	36
C_{10}	3	947	31	3	807	21	4	2059	35	4	2612	37
C_{11}	3	1076	36	3	1131	25	4	2204	42	4	3421	36
C_{12}	3	1170	35	3	1125	18	4	2645	39	4	2664	35
C_{13}	3	1076	36	3	975	24	4	2216	38	4	2333	38
C_{14}	3	1174	35	3	1283	19	4	2913	41	4	3102	36
C_{15}	3	1066	33	3	1034	20	4	2481	40	4	2551	38
C_{16}	3	1067	33	3	1106	19	4	3320	40	4	2691	34
C_{17}	1	55	1	2	519	7	2	1178	3	2	1571	11

Table 4: Cutting plane for medium density graphs

	I1			I2			I3			I4		
	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP
	14	31	1.646	15	29	1.138	14	30	1.859	16	30	0.746
	LB	time	round	LB	time	round	LB	time	round	LB	time	round
C_1	5	2500	27	5	3005	46	5	2108	22	4	1360	26
C_2	5	2502	27	5	2848	50	5	2470	25	4	1666	22
C_3	5	2755	29	5	4072	49	5	1752	25	4	1678	26
C_4	5	3990	28	5	3039	47	5	2236	25	4	1742	23
C_5	5	3083	30	5	3322	47	5	1976	27	4	1685	25
C_6	5	3993	28	5	4979	46	5	2254	26	4	2194	24
C_7	5	3379	32	4	2159	14	5	2039	24	4	1698	22
C_8	5	3375	32	5	4233	48	5	1926	25	4	2148	25
C_9	5	2599	29	5	3213	47	5	2141	25	4	1200	23
C_{10}	5	2602	29	5	3140	43	5	1954	24	4	1744	23
C_{11}	5	2674	29	4	2639	15	5	2199	24	4	1866	22
C_{12}	5	2965	30	4	1976	15	5	2878	22	4	2002	22
C_{13}	5	2863	27	5	2864	47	5	2981	26	4	1463	23
C_{14}	5	2967	30	5	4690	49	5	2973	26	4	1960	24
C_{15}	5	3697	29	4	1926	16	5	2873	25	4	1873	25
C_{16}	5	3696	29	5	3767	50	5	2843	25	4	2584	23
C_{17}	2	125	1	2	84	1	3	2330	6	2	1198	5
	I5			I6			I7			I8		
	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP	n_cli	$\hat{\chi}$	ILP
	29	49	4.886	31	48	4.469	32	48	4.184	33	48	4.067
	LB	time	round	LB	time	round	LB	time	round	LB	time	round
C_1	8	2067	33	8	1430	50	8	1244	38	8	1043	40
C_2	8	1989	41	8	1209	40	8	1576	43	8	1037	45
C_3	8	2395	36	8	1694	50	8	1724	42	8	963	38
C_4	8	2349	36	8	1388	43	8	1784	39	8	1142	42
C_5	8	1710	36	8	1248	40	8	1372	38	8	1025	39
C_6	8	2334	33	8	1548	43	8	1978	40	8	1009	39
C_7	8	2258	30	8	1501	39	8	1643	45	8	1454	45
C_8	8	3188	37	8	1631	46	8	1511	40	8	1348	45
C_9	8	2046	37	8	1442	46	8	1444	34	8	936	36
C_{10}	8	1697	32	8	1376	42	8	1229	35	8	969	38
C_{11}	8	2488	35	8	1729	42	8	1614	43	8	1261	46
C_{12}	8	2209	35	8	1735	48	8	1866	40	8	1392	42
C_{13}	8	2103	34	8	1514	38	8	1510	43	8	982	39
C_{14}	8	3288	39	8	2011	47	8	2047	41	8	1417	44
C_{15}	8	2313	35	8	2386	47	8	1849	42	8	1364	48
C_{16}	8	2634	36	8	1772	43	8	2166	43	8	1489	42
C_{17}	5	95	1	5	79	1	5	66	1	5	50	1

Table 5: Cutting plane for high density graphs

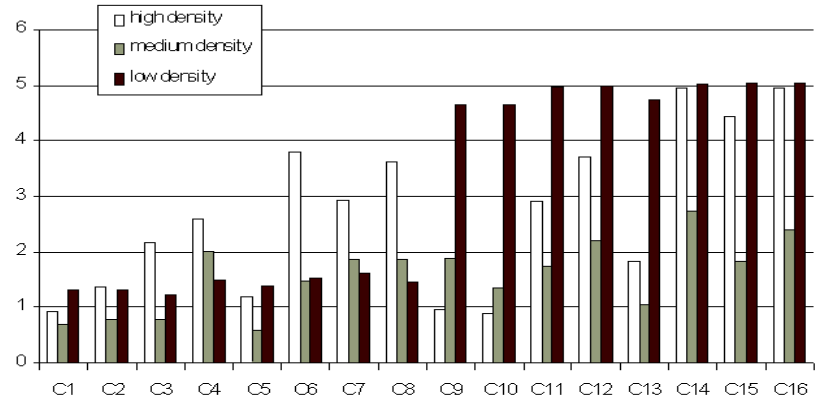
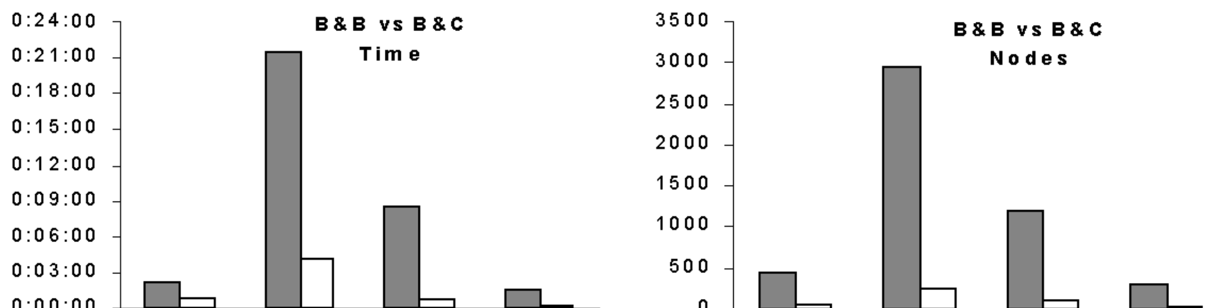


Figure 3: Average GAP

and ***path*** inequalities is the best for medium density graphs, and its behavior is good enough for the other densities. Because the medium density graphs are the hardest to color, we choose this combination for the next experiments.

Graph density seems not to be a crucial factor but we note that the cuts performance improves as the difference between the size of the maximum clique and the chromatic number increases. It is difficult to carry any further conclusions.

Finally, another experiment considered if it is worthwhile to include cuts. We compare a Branch-and-Bound version of our code with a Branch-and-Cut that uses ***clique***, ***block color*** and ***path*** inequalities. The next figure reports results with skip factor equal to 1, three rounds per node, a maximum of 2000 cuts per iteration and a limit of 20 rounds of the cutting plane algorithm on the root node.



	0.3	0.5	0.7	0.9
B&B	0:02:09	0:21:29	0:08:35	0:01:34
B&C	0:00:51	0:04:08	0:00:46	0:00:13

	0.3	0.5	0.7	0.9
B&B	428	2938	1187.2	282
B&C	47	244.2	95.6	15

Figure 4: Branch-and-Bound vs Branch-and-Cut

According to the Figure 4, the Branch-and-Cut method is capable of solving all test instances faster and producing fewer subproblems than Branch-and-Bound.

4.4 Comparison of LP-relaxations

In [16] we proposed two other models that include a smaller number of equivalent solutions. The polytopes associated to these models are:

$$CP' = SCP \cap \{(x,w) : \sum_{i\in V} x_{ij} \geq \sum_{i\in V} x_{ij+1}\}$$

$$CP" = SCP \cap \{(x,w) : x_{ij} = 0 \text{ if } j \geq i+1 \text{ and } x_{ij} \leq \sum_{k=j-1}^{i-1} x_{kj-1} \ \ \forall\, 2 \leq j \leq i \ \ \forall\, i \in V\}$$

The properties of *CP'* and *CP"* depend on some characteristics of the graph (e.g. order of the vertices), and it makes the study more difficult. However, since *CP'* and *CP"* are included in *CP*, the valid inequalities for *CP* are valid for *CP'* and *CP"* as well.

In order to compare the three linear relaxations we run experiments with a cutting plane algorithm. In addition, we also use the LP-relaxation of *SCP* to show the importance of eliminating symmetrical solutions.

To simplify the comparison we only use ***clique*** inequalities given their important role in a cutting plane algorithm, as we showed in 4.3. They are valid for all the polytopes [4].

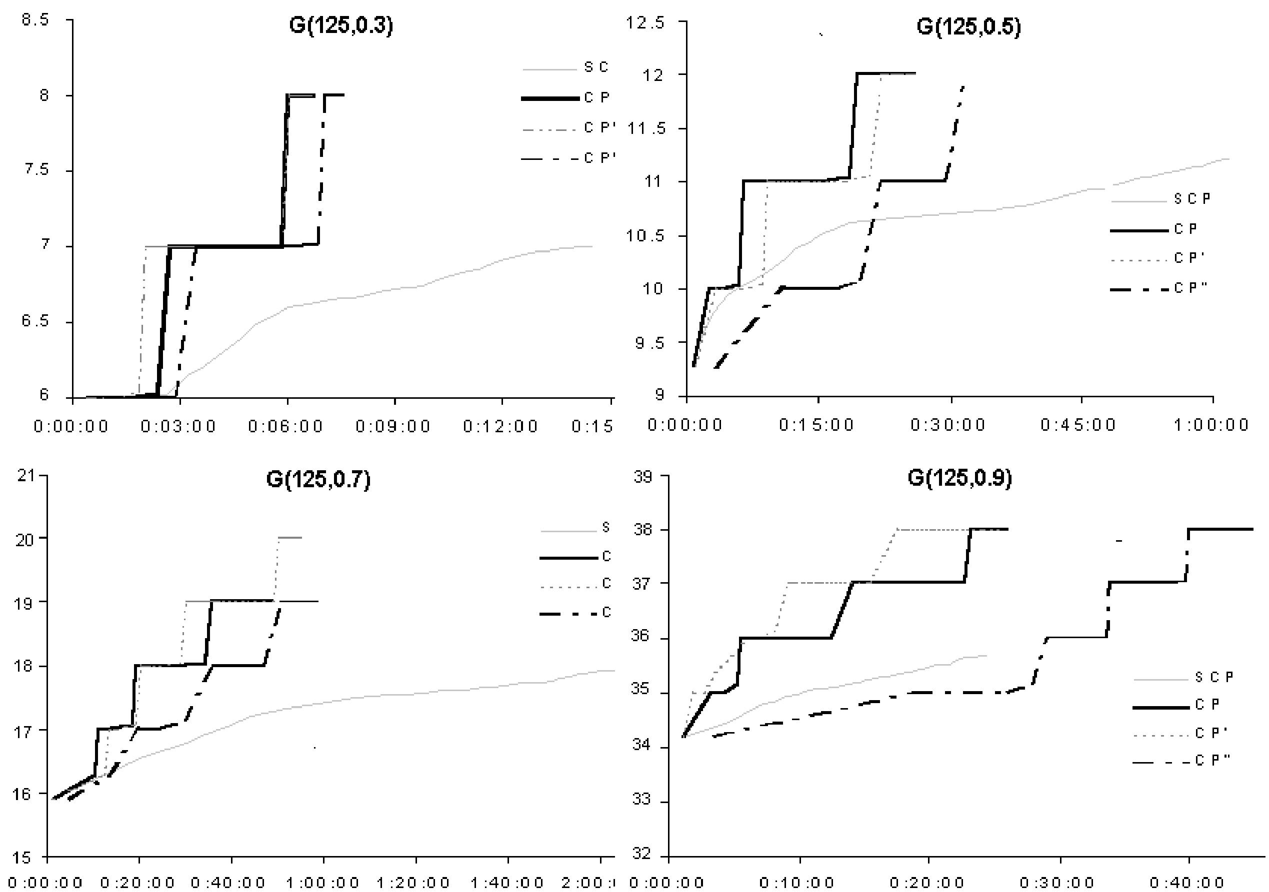


Figure 5: LP-relaxations

In the Figure 5, we represent the evolution of the objective function for the LP-relaxations. It is easy to see that *SCP* has the worst performance and a weaker lower bond is reached. The slow progress in the lower bound and the time required by the LP-solver is due to the large amount of symmetrical solution included in *SCP*.

Relaxations of *CP*, *CP'* and *CP"* have similar behavior in improving the lower bound. But, LP-solver takes longer on *CP"* relaxation as it contains more constraints.

Problem	n	m	n_cli	$\hat{\chi}$	χ	Branch-and-Cut	DSATUR
DSJC125_1	125	736	4	5	5	2.00	0.10
fpsol2_i_1	496	11654	55	65	65	29.00	0.30
fpsol2_i_2	451	8691	29	30	30	9.00	0.20
fpsol2_i_3	425	8688	29	30	30	9.00	0.20
miles1000	128	3216	41	42	42	0.02	0.10
miles1500	128	5198	71	73	73	0.14	0.10
ash331GPIA	662	4185	3	4	4	2719.0	1.70
3-FullIns_3	80	346	5	6	6	1.00	*****
4-FullIns_3	114	541	6	7	7	3.00	*****
5-FullIns_3	154	792	7	8	8	3.00	*****
mug88_1	88	146	3	4	4	485.00	*****
mug88_25	88	146	3	4	4	1690.0	*****
mug100_1	100	166	3	4	4	4029.0	*****
mug100_25	100	166	3	4	4	5498.0	*****
3-Insertions_3	56	110	2	4	4	10.00	12.70
1-FullIns_4	93	593	3	5	5	703.00	*****
2-FullIns_3	52	201	4	5	5	3.00	2558
queen8_8	64	728	8	10	9	96.00	46.00

Table 6:

As a general rule, the larger the gap between $n_c li$ and χ, the better the performance of *CP'* over *CP*. However *CP'* was not useful in a Branch-and-Cut framework. Because our branching strategies can not be applied, we run some experiments with other branching strategies. But, they show that even when LP-relaxation of *CP'* is better, it can not compensate the time due to the largest size of the tree.

4.5 Final results

After this performance, now we are ready to investigate the effectiveness of our code in DIMACS instance. Results of a preliminary version of our code were presented in [5]. We use a CPU time limit of two hours. In the instances that our code was unable to solve within this limit, we report the best lower and upper bounds. We compare our code with DSATUR algorithm. We use the source code available at Trick's page where it is incorporated the modification suggested by Sewell (http://mat.gsia.cmu.edu/COLOR/solvers/trick.c). Complete enumeration of feasible solutions in graphs with less than 50 vertices is very fast so we do not consider them in our report.

There are instances where the lower and the upper bound obtained with the initial heuristics are equal, so the Branch-and-Cut was not used for these graphs.

We begin showing the results of our experiments in Table 6. These are the instances that our Branch-and-Cut code was able to solve within the cpu time limit. Asterisks indicate DSATUR exceeded the time limit. The first 10 instances were solved at the root node after some cutting plane iterations because the gap between lower and upper bound was closed. The last 8 instances required to explore the Branch-and-Cut tree.

On Table 7 we report the lower and the upper bound obtained on instances where the CPU time limit was exceeded.

5 Final remarks

Our algorithm solves instances that DSATUR was not able to. In many cases, DSATUR finds the optimal solution very early in the enumeration process but requires too much time to conclude that there is not better solution. Branch-and-Cut was able to obtain the optimal certification faster than DSATUR. The improvement of the initial lower bound allows us to prove that the solution obtained by the initial heuristic was optimal.

Problem	n	m	n_cli	$\hat{\chi}$	χ	Branch-and-Cut		DSATUR	
						Lower	Upper	Lower	Upper
DSJC125_5	125	3891	9	20	?	13	20	9	19
DSJC125_9	125	6961	32	47	?	42	47	29	45
DSJC250_1	250	3218	4	9	?	5	9	4	9
DSJC250_5	250	15668	11	36	?	13	36	9	35
DSJC250_9	250	27897	38	88	?	47	88	34	85
DSJR500_1c	500	121275	72	87	?	76	88	70	88
queen9_9	81	2112	9	11	10	9	10	9	10
myciel6	95	755	2	7	7	5	7	2	7
myciel7	191	2360	2	8	8	5	8	2	8
1-Insertions_5	202	1227	2	6	?	4	6	2	6
1-Insertions_6	607	6337	2	7	?	4	7	2	7
2-Insertions_4	149	541	2	5	?	4	5	2	5
2-Insertions_5	597	3936	2	6	?	3	6	2	6
3-Insertions_4	281	1046	2	5	?	3	5	2	5
3-Insertions_5	1406	9695	2	6	?	3	6	2	6
4-Insertions_3	79	156	2	4	4	3	4	2	4
4-Insertions_4	475	1795	2	5	?	3	5	2	5
1-FullIns_5	282	3247	3	6	?	4	6	3	6
2-FullIns_4	212	1621	4	6	?	5	6	4	6
2-FullIns_5	852	12201	4	7	?	5	7	4	7
3-FullIns_4	405	3524	5	7	?	6	7	5	7
3-FullIns_5	2030	33751	5	8	?	6	8	5	8
4-FullIns_4	690	6650	6	8	?	7	8	6	8

Table 7: Bounds

Moreover, for instances not solved within the time limit, Branch-and-Cut reduces significantly the initial gap between the lower and upper bounds provided by n_cli and $\hat{\chi}$. The advantage of our approach is that provide good lower bounds. So, if the upper bound is good enough, the algorithm has good chances to find the optimal solution.

Our results suggest that our algorithm is a promising solution strategy and it has potential improvements. There is still place to new cutting plane generation, methods for new valid inequalities and others schemes to prune the search tree. They will be subject for further research.

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] E. Balas, S. Ceria, G. Cornuéjols, G. and G. Pataki, Polyhedral Methods for the Maximum Clique Problem, in Cliques, Coloring, and Satisfiability, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, Johnson, D. and Trick, M.(eds.), 1996, Vol. 26, 11-27.

[3] D. Brélaz, New methods to color the vertices of a graph, Communications of the ACM 22, 1979, pp. 251-256.

[4] P. Coll, J. Marenco, I. Méndez Díaz and P. Zabala, Facets of the graph coloring polytope, Annals of Operations Research, Vol. 116, 2002, pp.79-90.

[5] Computational Symposium on Graph Coloring and its Generalizations, Ithaca, september 2002.

[6] CPLEX Linear Optimization 6.0 with Mixed Integer & Barrier Solvers, ILOG, 1997-1998.

[7] D. de Werra, Heuristics for Graph Coloring, Computing, Suppl. 7, 1990, pp. 191-208.

[8] D. Johnson and M. Trick (eds.), Cliques, Coloring, and Satisfiability, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, Vol. 26, 1996.

[9] F. Glover, M. Parker and J. Ryan, Coloring by tabu branch and bound, in Coloring, Cliques, Coloring, and Satisfiability, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, Johnson, D. and Trick, M.(eds.), Vol.26, 1996, pp. 285-308.

[10] A. Herz and D. de Werra, Using tabu serch techniques for graph coloring, Computing, Vol. 39, 1987, pp. 345-351.

[11] M. Junger and 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, pp. 1325-1352

[12] R. Karp, Reducibility among combinatorial problems, Complexity of computer computations, eds. R. Miller and J. Thatcher, 1972, pp. 85-104.

[13] M. Kubale and B. Jackowski, A generalized implicit enumeration algorithm for graph coloring, Communications of the ACM 28, No. 4, 1985, pp. 412-418.

[14] C. Mannino and A. Sassano, An Exact Algorithm for the Maximum Estable Set Problem, Computational Optimization and Applications, 3, 1994, pp. 243-258.

[15] A. Mehrotra and M. Trick, A column generation approach for graph coloring, INFORMS J. on Computing, Vol. 8, No. 4, 1996, pp. 344-353.

[16] I. Méndez Díaz and P. Zabala, A Polyhedral Approach for Graph Coloring, Electronics Notes in Discrete Mathematics, Vol.7, 2001.

[17] G. Nemhauser and G. Sigismondi, A strong cutting plane/branch-and-bound algorithm for node packing, J. Ops. Res. Soc., 43(5), 1992, pp. 443-457.

[18] M.W. Padberg, On the facial structure of Set Packing Polyhedral, Math. Prog. 5, 1973, pp. 199-215.

[19] A. Sassano,On the facial structure of the Set Covering Polytope, Math. Prog. 44, 1989, pp.181-202.

[20] T. J. Sager and S. Lin, A pruning procedure for exact graph coloring, ORSA Journal on Computing 3, No. 3, 1991, pp. 226-230.

[21] E. Sewell, An improved algorithm for exact graph coloring, Cliques, Coloring, and Satisfiability, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, Vol. 26, D. Johnson and M. Trick (eds.), 1996, pp. 359-373.