Exact Solution of Graph Coloring Problems via Constraint Programming and Column Generation

Stefano Gualandi, Federico Malucelli
Dipartimento di Elettronica ed Informazione, Politecnico di Milano, Piazza L. da Vinci 32, Milano
stefano.gualandi@polimi.it,malucell@elet.polimi.it

We consider two approaches for solving the classical minimum vertex coloring problem, that is the problem of coloring the vertices of a graph so that adjacent vertices have different colors, minimizing the number of used colors, namely Constraint Programming and Column Generation. Constraint Programming is able to solve very efficiently many of the benchmarks, but suffers from the lack of effective bounding methods. On the contrary, Column Generation provides tight lower bounds by solving the fractional vertex coloring problem exploited in a Branch-and-Price algorithm, as already proposed in the literature. The Column Generation approach is here enhanced by using Constraint Programming to solve the pricing subproblem and to compute heuristic solutions. Moreover new techniques are introduced to improve the performance of the Column Generation approach in solving both the linear relaxation and the integer problem. We report extensive computational results applied to the benchmark instances: we are able to prove optimality of 11 new instances, and to improve the best known lower bounds on other 17 instances. Moreover we extend the solution approaches to a generalization of the problem known as Minimum Vertex Graph Multicoloring Problem where a given number of colors has to be assigned to each vertex.

Key words: Column Generation, Integer Linear Programming, Constraint Programming, Graph Coloring
History:

---

1 Introduction

Given a graph $G = (V, E)$ and an integer k, a k-coloring of G is a one-one mapping of vertices to colors, such that adjacent vertices are assigned to different colors. The Minimum Graph Coloring Problem (Min–GCP) consists in finding the minimum k such that a k-coloring exists. Such minimum k is known as the chromatic number of G and is denoted by $\chi(G)$, or simply by χ. Min–GCP is NP-hard. The chromatic number is bounded from below by the size of the *maximum clique* of G, known as the clique number $\omega(G)$ which is equal to $\chi(G)$ when G is a perfect graph.

A formulation of Min–GCP based on Column Generation was introduced in Mehrotra and Trick (1996), where the master subproblem is a set partitioning and the pricing subproblem is a maximum weighted stable set problem. Column Generation-based Branch-and-Bound algorithms, known as Branch-and-Price, are reputed, so far, the most efficient exact methods to solve Min–GCP.

In addition to the Branch-and-Price presented in Mehrotra and Trick (1996) other remarkable exact approaches to MIN–GCP are the DSATUR algorithm (Brélaz, 1979) and the Branch-and-Cut (Méndez-Díaz and Zabala, 2006). DSATUR is a sequential vertex coloring algorithm that successively colors the vertices sorted in a predetermined order. The ordering is based on the *saturation degree* of a vertex, that is the number of different colors adjacent to the vertex. The Branch-and-Cut is based on the polyhedral study of the graph coloring polytope presented in Coll et al. (2002), and it applies several families of valid inequalities, such as, for instance, the clique and the neighborhood facet defining inequalities. The same families of inequalities are used in a cutting-plane algorithm in Méndez-Díaz and Zabala (2008). An implementation of a Branch-and-Price-and-Cut approach for MIN–GCP is reported in Hansen et al. (2009), where a family of valid inequalities that do not break the structure of the pricing subproblem is introduced. Despite this interesting idea, the practical impact of this approach is limited. For a recent survey on exact and heuristic methods refer to Malaguti and Toth (2010).

The combinatorial structure of coloring problems makes Constraint Programming (CP) approaches particularly interesting and often efficient and competitive with respect to the mathematical programming ones. CP is an emerging programing paradigm that has proved to be successful for checking if a k-coloring exists, since constraint propagation can be exploited quite effectively (Barnier and Brisset, 2004) by means of the well–known `alldifferent` constraint (Régin, 1994). Recently, the efficiency of CP solvers has significantly improved, however standard CP approaches lack of efficient mechanisms to compute tight lower bounds and to guide the search towards the optimal solution.

The idea of exploiting good lower bounds obtained via mathematical programming within a CP model gives rise to hybrid methods (e.g., see Milano and Wallace (2006)). A promising hybrid approach is the so–called Constraint Programming-based Column Generation (CP–CG) that formulates and solves the pricing subproblem via CP. The CP–CG framework was introduced in Junker et al. (1999) for crew management problems. For a recent survey on CP–CG, see Gualandi and Malucelli (2009), and for a summary of the main references therein contained, see Table 1.

In this paper, we revive the work presented in Mehrotra and Trick (1996) by solving the pricing subproblems with three different methods, more up to date with respect to the original paper. The first method is based on a very efficient algorithm for solving the (weighted) maximum clique problem, presented in Ostergard (2002) and called CLIQUER. CLIQUER can be used both as a heuristic to find a maximal clique of weight at least equal to a given value, and as an exact method to solve the maximum weighted clique problem. Since CLIQUER is very efficient mainly for sparse graphs, but not so efficient for dense graphs, we have exploited a second method for solving the pricing subproblem using a heuristic (weighted) maximum clique algorithm, based on a trust-region method, that was introduced in Busygin (2006) and is called

Application	Refs.
Wireless Mesh Networks	Capone et al. (2010)
Employee Timetabling	Demassey et al. (2006)
Traveling Tournament Problem	Easton et al. (2002)
Airline Crew Assignment	Fahle et al. (2002); Junker et al. (1999); Sellmann et al. (2002)
Constrained Cutting Stock	Fahle and Sellmann (2002)
Airline Planning	Gabteni and Grönkvist (2006); Grönkvist (2004, 2006)
Grouping Cabin Crew	Hansen and Liden (2005)
Two Dimensional Bin Packing	Pisinger and Sigurd (2007)
Vehicle Routing Problem with Time Windows	Rousseau (2004); Rousseau et al. (2004)
Multi-Machine Assignment Scheduling Problem	Sadykov and Wolsey (2006)
Urban Transit Crew Management	Yunes et al. (2000, 2005)

Table 1: Successful applications of the CP–CG framework.

Qualex-MS. The main feature of Qualex-MS is to compute quasi-optimum weighted cliques. Finally, we considered a third approach to solve the pricing subproblem based on Constraint Programming and relying on a weighted version of the cost-based filtering algorithm for the max-clique problem introduced in Fahle (2002).

We unify the three methods in a CP–CG approach where we consider two possible formulations for the pricing subproblem: in the first one, the most negative reduced cost column, i.e. a maximum weighted stable set, is sought, in the second one, the decision problem that finds a maximal stable set with negative reduced cost smaller than a threshold τ is considered. In the column generation algorithm, we alternatively apply the two versions, utilizing one of the three algorithms mentioned above for the clique. In addition, we introduce a so–called *augmented pricing* with the aim of generating integer feasible solutions during the execution of the column generation algorithm. The motivations for having an *augmented pricing* are twofold: first, integer feasible solutions so generated yield columns that can contribute the optimal solution, and second, since in many cases the solution of the column generation algorithm is by itself time-consuming, improving both the upper bound and the lower bound at the same time may be profitable.

The outline of this paper is as follows. Section 2 presents the CP approach, able to solve more than half of the classical Min–GCP benchmarks. Section 3 reviews the Column Generation approach producing tight lower bounds. Section 4 presents a CP–CG approach able to overcome the limitations of both the CP and the Column Generation approaches, yielding some enhancements. Extensive computational results are reported in Section 5. Section 6 extends the Branch-and-Price algorithm to the Minimum Graph Multicoloring Problem (Min–GMP) and reports additional computational results. Finally, Section 7 concludes the paper presenting new challenges for Min–GCP and Min–GMP.

2 Graph Coloring via Constraint Programming

Constraint Programming is a programming paradigm for solving combinatorial problems that combines *expressive* modeling languages with *efficient* solver implementations. An introductory textbook on CP is Apt (2003), while the state-of-the-art on CP is contained in Rossi et al. (2006). The two basic concepts of CP are *constraint propagation* and *constructive search*. Constraint propagation is an efficient inference mechanism aiming at reducing the domains of the problem variables by exploiting the semantics of the problem constraints. The inference mechanism is implemented into *filtering algorithms* that filter out values from the domain of each variable. When constraint propagation is unable to further reduce the domains of variables, the *constructive search* comes into play. Constructive search explores the search tree by tightening the domain of a single variable at a time. In practice, it performs a search in the space of partial solutions. The simplest form of constructive search consists of selecting an undetermined variable, i.e., a variable having more than a value in its domain, and assigning a value to that variable. This form of search is called *labeling*. By iterating constraint propagation and labeling, the CP solver computes the solution(s) of a given problem, or it reports that none exists. In the recent literature several attempts to combine CP with OR can be found with the intent of integrating the pros of the two approaches. We refer the reader to Milano and Wallace (2006) for a recent survey on hybrid methods that combine CP and OR techniques. CP–CG is one of the most successful examples of such combinations.

The CP model of the Min–GCP problem relies on the `alldifferent` constraint, introduced in Régin (1994). This constraint forces a set of integer variables to take different values in a given set. The strength of the `alldifferent` constraint is an efficient propagation algorithm that is able to reduce the domain of each variable by exploiting the bipartite matching theory.

Given a graph $G = (V, E)$, let $\underline{\chi}$ and $\overline{\chi}$ denote a lower and upper bound for $\chi(G)$. Let $K = \{1, \ldots, \overline{\chi}\}$ be the set of available colors (assuming that colors map to natural numbers), and let $x_i \in K$ be a finite domain integer variable denoting the color assigned to vertex $i \in V$. Let x_0 be a finite domain integer variable denoting the number of used colors, hence at the optimal solution $x_0 = \chi(G)$. The CP model of Min–GCP

is as follows:

$$
\begin{aligned}
&\textit{variables:} && \texttt{domain}(x_0) = \{\underline{\chi} \dots \overline{\chi}\}, && && \\
& && \texttt{domain}(x_i) = K, && \forall i \in V, && \\
&\textit{constraints:} && x_i \neq x_j, && \forall \{i,j\} \in E, && (1) \\
& && \texttt{alldifferent}(\{x_i \mid i \in C\}), && \forall C \in \mathcal{C}, && (2) \\
& && x_0 = \texttt{max}(\{x_i \mid i \in V\}), && && (3) \\
&\textit{cost bounding:} && x_0 \leq x_0^*. && && (4)
\end{aligned}
$$

Constraints (1) state that adjacent vertices are assigned to different colors. The redundant constraint (2) imposes that all vertices in a clique C have different colors, for every clique C belonging to a given collection $\mathcal{C}$; the method used to define $\mathcal{C}$ is described below. Constraints (3) impose that no vertex takes a color bigger than x_0. Denoting by x_0^* the cost of the last solution found during the CP search tree, the bounding constraint (4) minimizes x_0, thus the number of the used color.

The CP model of Min–GCP is simple and intuitive, but in order to be solved efficiently, a number of issues, presented in the next paragraphs, must be considered. Note that most of those methods can be profitably applied also in Mathematical Programming approaches.

2.1 Preprocessing

Let us summarize some standard preprocessing techniques for Min–GCP.

1. A maximal clique of G is computed and different colors are assigned to its vertices. To find a maximal clique in G, we use the CP-based pricing subproblem defined in Section 4, where vertex weights are set to one. In addition, the value $\bar{\omega}$ of the maximal clique can be used to reduce the graph size by removing every vertex that has degree smaller than $\bar{\omega} - 1$.

2. A tight upper bound $\overline{\chi}$ on the number of colors is obtained with a heuristic algorithm. We use the coloring heuristic present in the Boost Graph Library (BGL), which is an implementation of the algorithm presented in Coleman and More (1984). A better performance could be obtained by using more sophisticated heuristics as, for instance, that proposed in Leighton (1979) or Hertz and de Werra (1987).

3. The collection of cliques $\mathcal{C}$ used to post the `alldifferent` constraint is a crucial element of the CP model. In Gualandi (2008), several strategies are evaluated to decide how to determine $\mathcal{C}$, but since no strategy dominates the others we have adopted the simplest: to solve several times the graph coloring

problem on the complementary graph $\bar{G}$ with the heuristic in the BGL, randomly shuffling the order of the vertices. Each class of colors in $\bar{G}$ corresponds to a clique of G.

2.2 Upper and lower bounding via CP

When solving model (1)–(3), there are two alternative options for the CP labeling strategy of variable x_0: (i) x_0 is not considered as a branching variable, thus constraint (3) determines the value of x_0 only when all the other variables x_i (with $i > 0$) have been assigned, or (ii) variable x_0 is a branching variable.

In the first case, a Branch-and-Bound using the cost-bounding constraint (4) yields solutions with decreasing costs. We call this strategy *CP–UB*, since it provides a sequence of upper bounds even if the search is stopped before the search tree has been completely explored. In the second case, we can consider the following two-level labeling strategy: in the first level, we assign to variable x_0 the minimum value v' in its domain, then, in the second level, we label all the other x_i variables. In this way, the first solution found by the CP solver is the optimum one, and the search can stop. Note that, in this case, the cost-bounding constraint (4) is not needed. In addition, each time the second level labeling strategy has to backtrack to the first level in order to assign a new value to variable x_0, it means that a k coloring with $k = v'$ does not exist, thus the lower bound increases. We call this second strategy *CP–LB*, since if it is stopped before the search tree has been completely explored, the current value of x_0 is the best lower bound.

2.3 Symmetry Breaking

Min–GCP is a challenging problem partially due to the high number of symmetric solutions. While fixing the colors of a maximal clique breaks statically a certain number of symmetric solutions in the preprocessing, many more symmetries arise during the solution of problem (1)–(4). In general the same criterion can be applied at each labeling phase. Consider the set of variables with an undetermined value and such that their domains have null intersection with the set of assigned colors, and let C' be a clique of the corresponding vertices of G. As it is done in the preprocessing phase, we can avoid to generate branches on the assignment of these variables and we can directly assign them a different color selecting it from their domain. The simplest and less time consuming method is to consider cliques of one vertex, which allows in any case to significantly reduce the number of search nodes in a reasonable time.

Other symmetry breaking strategies have been proposed in the literature (e.g. Gent et al., 2006). They are computational more demanding and more complex to be implemented. Since the evaluation of their utility goes beyond the scope of this work, we decided to leave their use for future investigation.

3 Graph Coloring via Branch-and-Price

MIN–GCP can be also seen as the problem of partitioning the vertices of a graph into the minimum number of stable sets, since all stable set vertices can be assigned to the same color, no matter which. Being a minimization problem, MIN–GCP can be formulated as a set covering where the ground set is given by V and the family of subsets is the collection of maximal stable sets of G. Alternatively, a set packing formulation of MIN–GCP is presented in Hansen et al. (2009), but since it yields the same lower bound than the set covering formulation, we consider here only the former.

Let $\mathcal{S}$ be the collection of all maximal stable sets of $G = (V, E)$, and $\mathcal{S}_v$ be the collection of maximal stable sets containing $v \in V$. Denoting by λ_i the selection variable of stable set i, the set covering formulation of MIN–GCP is:

$$z_{MP} = \min \quad \sum_{i \in \mathcal{S}} \lambda_i \tag{5}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{S}_v} \lambda_i \geq 1, \qquad \forall v \in V, \tag{6}$$

$$\lambda_i \in \{0, 1\}, \qquad \forall i \in \mathcal{S}. \tag{7}$$

In a Column Generation scheme, we consider the so called *restricted master problem*:

$$z_{RMP} = \min \quad \sum_{i \in \bar{\mathcal{S}}} \lambda_i \tag{8}$$

$$\text{s.t.} \quad \sum_{i \in \bar{\mathcal{S}}_v} \lambda_i \geq 1, \qquad \forall v \in V, \tag{9}$$

$$\lambda_i \in [0, 1], \qquad \forall i \in \bar{\mathcal{S}}. \tag{10}$$

where $\bar{\mathcal{S}} \subseteq \mathcal{S}$ is such that the existence of a feasible solution is guaranteed, that is $\bar{\mathcal{S}}$ is a subset of maximal stable sets such that each vertex v of G is contained in at least one set of $\bar{\mathcal{S}}$. Let $\bar{z}_{MP}$ denote the linear relaxation optimal solution value of problem (5)–(7). Note that z_{MP} is equal to the chromatic number $\chi(G)$, and that $\bar{z}_{MP}$ is equal to the fractional chromatic number $\chi_f(G)$. The computation of $\chi_f(G)$ is an important NP-hard problem, and it gives the sharpest known lower bound for χ (Schrjiver, 2008).

Consider the dual of problem (8)-(10) where we denote by π_v the dual variables of constraints (9). Given the optimal dual solution $\bar{\pi}_v$, to check the existence of negative reduced columns to be added to (8)–(10), the following pricing subproblem has to be solved:

$$s^* = \max \quad \sum_{v \in V} \bar{\pi}_v y_v \tag{11}$$

$$\text{s.t.} \quad y_v + y_w \leq 1, \qquad \forall \{v, w\} \in E, \tag{12}$$

$$y_v \in \{0, 1\}, \qquad \forall v \in V. \tag{13}$$

Note that this problem is equivalent to finding the maximum weight stable set: variable y_v is equal to one if vertex v is part of the stable set and constraints (12) avoid that neighbors of v are in the solution. Negative reduced cost columns correspond to solutions of problem (11)–(13) with value strictly greater than 1, thus the actual value of the pricing subproblem is $c^* = 1 - s^*$. The pricing subproblem (11)–(13) is solved in Mehrotra and Trick (1996) with a recursive algorithm that could be used both as an exact and a heuristic method. In the heuristic case, the execution is stopped when a stable set with $s^* > 1.1$ (i.e. $c* < -0.1$) is found.

If the pricing subproblem is solved with an exact method, the value $\frac{z_{RMP}}{1-c^*}$ is a valid lower bound on $\bar{z}_{MP}$, as shown in Lübbecke and Desrosiers (2005). This bound can be used to early terminate the column generation algorithm whenever $\lceil z_{RMP} \rceil = \lceil \frac{z_{RMP}}{1-c^*} \rceil$.

4 Constraint Programming-based Column Generation

The main idea of the CP–CG framework is to use CP to formulate and solve the pricing subproblem. The use of CP is sound since it is not necessary to solve the pricing subproblem to optimality, but it suffices to find a negative reduced cost solution. In order to formulate the pricing subproblem with CP, the `negativeReduceCost` global constraint has been introduced in Fahle et al. (2002). Let Y be a set variable having domain ranging into the vertex set V, and let $\tau \leq 0$ be a parameter. In our context, the $\texttt{negativeReducedCost}(\overline{\pi}, Y, \tau, c)$ symbolic constraint forces the set of vertices in Y to define a column with reduced cost smaller than τ. Fixing the value of τ to 0 is equivalent to look for any negative reduced cost column. This way, the general CP model of the pricing decision subproblem is:

$$\textit{variables/domains:} \quad Y \subseteq V, \tag{14}$$

$$\texttt{domain}(X_i) \in \{0,1\}, \qquad \forall i \in V, \tag{15}$$

$$\texttt{domain}(c) < 0, \tag{16}$$

$$\textit{constraints:} \quad \texttt{negativeReducedCost}(\overline{\pi}, \texttt{Y}, \texttt{0}, \texttt{c}), \tag{17}$$

$$X_i + X_j \leq 1, \qquad \forall \{i,j\} \in E, \tag{18}$$

$$X_i = 1 \Leftrightarrow i \in Y, \qquad \forall i \in V, \tag{19}$$

$$\textit{cost bounding:} \quad c_0 \leq c_0^*. \tag{20}$$

Constraints (17) is implemented as a weighted maximum clique constraint, that forces the vertices in Y to be a clique of the complement $\bar{G}$ of G, i.e., a stable set of G, with cost at least equal to τ. Constraints (18) are equivalent to constraints (12). Constraints (19) link the binary variables X_i with the set variable Y.

The CP pricing can be solved either as a decision problem or as an optimization problem. In the first case, it is enough to solve problem (14)–(19) to obtain an improving column for the restricted master problem. In the second case, the cost variable `c` is used as an *optimization variable*. During the search process, each time the CP solver finds a solution of cost c^*, a *bounding constraints* $c \leq c^* - 1$ is added until optimality is proven. In the context of CP–CG, the use of *optimization constraints* that perform cost-based domain reduction is of particular interest for enhancing the filtering of the `negativeReduceCost` constraints. For further details on the CP solving techniques and on more involved CP search strategies, see Milano and Wallace (2006).

We define some quality measures of columns added to the restricted master problem during the Column Generation process in order to account for their impact in the solution of the linear relaxation and of the integer problem. Let a be a column with negative reduced cost equal to c', and let c^* be the optimum value of the pricing subproblem.

Definition 1 *The* **relative reduced cost** γ *of column a with respect to (8)–(10) is*

$$\gamma = \frac{c'}{c^*}.$$

Definition 2 *Column a is* γ**-LP-good** *if it has a relative reduced cost $\gamma > 0$.*

Solving the optimization version of the pricing subproblem is equivalent to generate 1-LP-good columns.

Let z^0_{RMP} be the optimum of the integer restricted master problem (8)–(9) with 0–1 variables λ_i without column $\boldsymbol{a}$, and let z'_{RMP} be the optimum of the same integer master problem but extended by column $\boldsymbol{a}$.

Definition 3 *The* **integer impact factor** δ *of column $\boldsymbol{a}$ with respect to (8)–(9) with 0–1 variables λ_i is*

$$\delta = \frac{z^0_{RMP} - z'_{RMP}}{z^0_{RMP}}.$$

Definition 4 *Column a is* δ**-IP-good** *if it has an integer impact factor $\delta > 0$.*

In a Column Generation approach γ-LP-good and δ-IP-good columns are important to improve the lower bound and the upper bound, respectively. These issues have been addressed by several papers in the recent literature. Further on, we present the related work while illustrating our enhancing strategies.

4.1 Weighted Maximum Clique Constraint

Constraint (17) plays a crucial role in the solution of the CP pricing subproblem. It might be implemented as the naive sum over set constraint, but it would perform a rather weak propagation. Therefore, we have extended the maximum clique constraint presented in Fahle (2002) to the case where vertices have positive weights and we look for the maximum weighted clique.

The maximum clique constraint presented in Fahle (2002) basically works as follows. Two sets of vertices are maintained: (i) the current partial solution C and (ii) the candidate set P, i.e., the set of vertices that could extend C to a larger clique. A Branch-and-Bound algorithm selects the vertices from P to be added to C; each time a vertex $v \in P$ is moved into C, the candidate set is reduced as $P \leftarrow P \cap N(v) \setminus C$, where $N(v)$ is the set of vertices adjacent to v. An additional filtering algorithm removes from the set P all the vertices having degree smaller than $|C|$. Despite its simplicity, this rule is very effective. Therefore, we have adapted it to the weighted case. Let $w(C)$ be the weight of the current clique. Then, the filtering algorithm for the weighted case is: remove from the set P every vertex having the sum of the weights of its neighbors plus its own weight smaller than $w(C)$.

The maximum weighted constraint is used in a CP problem with an ad hoc labeling strategy that selects from the candidate set P the vertex v maximizing the sum of vertex weights in $N(v) \cap P$. In case of ties, the vertex with highest degree is selected.

4.2 Generating γ-LP-good columns

The aim is to reduce the computation time required for solving the linear relaxation of master problem (5)–(7) Note that the strategies that we are going to present are not tailored to graph coloring and can be used also in other contexts.

4.2.1 *Shuffled* Static Order

In column generation, producing *diverse* columns is a desideratum, since diverse columns improve the convergence rate of the algorithm, and mitigate the degeneracy phenomenon (Lübbecke and Desrosiers, 2005) which unfortunately is quite common when CP is used to approach the pricing problem (Rousseau, 2004). Indeed, during the iterations of the column generation algorithm, the pricing subproblem (14)–(17) does not change, with the exception of coefficients $\overline{\pi}$ in the `negativeReducedCost` constraint. The decision variables are stored in a vector $\mathbf{y}$ having always the same order corresponding to that of matrix A rows. Even if the CP solver uses a dynamic ordering for labeling the variables, for instance the `first-fail` strategy, it will always query them in vector y in the same order, typically from the lowest to the highest index. Therefore, if the dynamic ordering heuristic has ties, independently from the way ties are broken and from how the search tree is visited, variables with lower indices have more chances to be selected appearing more frequently in the generated columns.

In order to avoid this phenomenon, several ideas have been proposed. For example in Sellmann et al. (2002) variables selected more than k times are penalized in the labeling, or in (Rousseau, 2004) the Limited Discrepancy Search (Harvey and Ginsberg, 1995) is used as a CP search strategy to implicitly generate

diverse columns. In contrast to Depth First Search, Limited Discrepancy Search is not a complete search strategy and depends on a parameter, that is, the maximum number of allowed discrepancy whose value may heavily affect the performance and that in practice is set by trial-and-error.

Our idea is to statically random shuffle variables indices of vector $\mathbf{y}$ at each iteration of the column generation, before the CP solver begins the search phase. This *shuffled* static order acts as an implicit random tie-breaking strategy. The effect of the shuffling is impressive: it produces very diverse columns as a consequence of the constraint propagation, and it reduces dramatically the number of iterations of the Column Generation (Gualandi, 2008). Note that the shuffled static order is straightforward to implement, parameter-free and problem independent, moreover it is orthogonal to the search strategy.

The idea of using a random component in the search strategy is not completely new, but the way and the purpose of its use are part of our contribution: the concern is not to speed-up the solution of a single problem instance, but, given a set of instances of the same problem, to generate diverse solutions. An iterative randomized search strategy that randomly selects variable-value pairs during the search is presented in Gomes et al. (1998). The iterative randomized search is applied in Otten et al. (2006) obtaining a considerable speed-up against usual deterministic search strategies. Though efficient, the iterative random search strategy depends heavily on the parameter setting. The idea of the *dual strategy* presented in Gendron et al. (2005) is close to that of the *shuffled* static order. The *dual strategy* selects first the variable having the i-th smallest reduced cost, where i is a pseudo-random number, and it is used for generating multiple columns at the same iteration of the column generation that is with the same dual multipliers $\bar{\pi}$, rather than generating diverse solution in different iterations.

4.2.2 Adaptive Thresholds for the Negative Reduced Cost Constraint

The solution of the pricing (11)–(13) as a decision problem through CP model (14)–(19) aims at generating γ-LP-good columns limiting the computation effort. A solution of the pricing decision problem may have a negative reduced cost slightly smaller than zero, yielding a modest improvement. If we knew the cost c^* of the pricing subproblem optimum solution, we could add to the CP model (14)–(19) the constraint $c = c^*$. Since the value c^* is not a priori known, our idea is to introduce an oracle that gives an estimate τ such that $c^* \leq \tau < 0$. If τ is equal to c^*, the solution of the decision problem is a 1-LP-good column and it has been obtained without the need to prove optimality.

In practice, the idea is to alternate between the decision and the optimization pricing subproblems to take advantage of the respective strengths. The pricing decision subproblem is faster since it can be arrested whenever a solution with reduced cost lower than a given value τ is found. The pricing optimization problem is useful to compute the following lower bound for the restricted master problem (Lübbecke and Desrosiers

(2005)):

$$\kappa(c^*) = \frac{z_{RMP}}{1 - c^*} \leq \overline{z}_{MP}. \tag{21}$$

The underlying intuition of using a threshold τ is that we would like to obtain a new column potentially yielding a lower bound improving the current one without solving the optimization pricing subproblem. The idea of using a threshold is not new: for instance in Mehrotra and Trick (1996), the value of τ is empirically fixed to 1.1. To the best of our knowledge, ours is the first attempt to determine the values of τ analytically, and not experimentally by a trial-and-error hand-tuned strategy.

First threshold: exploiting an a priori known lower bound. Let κ be a given lower bound for $\overline{z}_{MP}$ and let $\kappa(c^*)$ computed as in (21). There are two possibilities:

$$\begin{aligned} \text{either} \qquad & \kappa \leq \kappa(c^*) < \overline{z}_{MP} \\ \text{or} \qquad & \kappa(c^*) < \kappa < \overline{z}_{MP} \end{aligned}$$

In the first case, as $\kappa(c^*)$ would yield a better lower bound than κ, the optimization pricing could be worth solving. While, in the second case, this would be useless, since κ would be anyway the best lower bound. Therefore, if the pricing subproblem finds a solution of cost $\bar{c}$ such that $\kappa(c^*) \leq \kappa(\bar{c}) < \kappa$, we could stop the solution of the pricing subproblem, saving computation time.

Considering the Min–GCP setting, the idea is to look for a maximal stable set that attains a reduced cost $\overline{c} < 0$, and that satisfies the inequality:

$$\kappa(c^*) \leq \kappa(\bar{c}) = \frac{z_{RMP}}{1 - \overline{c}} < \kappa < \overline{z}_{MP} \tag{22}$$

Let z^i_{RMP} be the value of the restricted master problem at the i-th iteration, and let $\overline{s}$ denote the value of the maximal stable set (i.e., $\overline{c} = 1 - \overline{s}$). Using the second and the third term in (22), the value of τ at the i-th iteration is computed as follows:

$$\frac{z_{RMP}}{1 - \overline{c}} < \kappa \quad \rightarrow \quad \overline{c} = 1 - \overline{s} = 1 - \frac{z^i_{RMP}}{\kappa} \quad \rightarrow \quad \overline{s} = \frac{z^i_{RMP}}{\kappa} = \tau(i). \tag{23}$$

At the i-th iteration, if solving the pricing subproblem, we find a stable set of value $s > \tau(i) = \frac{z^i_{RMP}}{\kappa}$, we can stop since, even if we could find a solution of higher cost, we would not get a lower bound better than κ. Therefore, in constraint (17), we can set the value of c equal to $\tau(i)$ computed as in (23), and solve the decision problem (14)–(19).

Second threshold: exploiting the lower bound $\underline{\kappa}(c^*)$. Let j be the current iteration of the column generation algorithm and be i a preceding one, i.e. $i < j$, when the optimization pricing subproblem (14)–(20) has been solved yielding the lower bound $\underline{\kappa}^i(c^*)$ computed by (21). Let z^j_{RMP} be the value of the

Algorithm 1 CP-based Column generation using adaptive thresholds.

In $\bar{\mathcal{S}}$: subset of maximal stable set $\mathcal{S}$
Var $\overline{\pi}$: dual optimal multipliers
Var $\underline{\kappa}(c^*)$: lower bound on $\overline{z}_{MP}$
Var s : maximal stable set with negative reduced cost
Var τ : threshold for the negative reduced cost constraint

1: **repeat**
2: $\{z_{RMP}, \overline{\pi}\} \leftarrow$ `solveRMP(`$\bar{\mathcal{S}}$`)` ▷ solve problem (8)–(10)
3: $\tau \leftarrow$ `computeThreshold` ▷ using either formula (23) or (24)
4: $s \leftarrow$ `solvePricingCP(`$\overline{\pi}, \tau$`)` ▷ solve problem (14)–(20)
5: **if** s is empty **then**
6: $\{c^*, s\} \leftarrow$ `solvePricing(`$\overline{\pi}, 0$`)` ▷ if solved to optimality update $\underline{\kappa}(c^*)$
7: **end if**
8: **if** s is not empty **then**
9: $\bar{\mathcal{S}} \leftarrow \bar{\mathcal{S}} \cup s$
10: **end if**
11: **until** s is empty (**or** $\lceil z_{RMP} \rceil = \lceil \underline{\kappa}(c^*) \rceil$)

restricted master problem at the j-th iteration. Then in order to compute the value of τ, we replace $\underline{\kappa}$ with $\underline{\kappa}^i(c^*) = \frac{z^i_{RMP}}{c^{*i}}$ in formula (23) as follows:

$$\tau(i,j) = \frac{z^j_{RMP}}{\underline{\kappa}^i(c^*)} = \frac{z^j_{RMP}}{z^i_{RMP}} c^{*i}. \tag{24}$$

Note that when the algorithm approaches the optimal solution $z^j_{RMP} \simeq z^i_{RMP}$, $\overline{c}^{*\,i} \simeq 0$, and consequently the value of the estimate τ tends to 0. In addition, if a lower bound $\underline{\kappa}$ is given, the current lower bound is $\underline{\kappa}^i(c^*) = \max\{\underline{\kappa}, \underline{\kappa}^i(c^*)\}$. The main difference between (23) and (24) is that the second is usually more accurate, but it requires to solve the pricing subproblem to optimality. It is not always evident whether the improved accuracy is worth the additional computational effort.

Embedding the estimate τ into column generation. The regular column generation algorithm extended by considering the threshold values is described by Algorithm 1. To guarantee the correctness the following additional steps are necessary:

step 3: to compute the threshold τ according to either (23) or (24);

step 4: to solve the CP-pricing using τ in the `negativeReducedCost` constraint. This is achieved by the procedure `solvePricingCP` that solves problem (14)–(20) with cost vector $\bar{\pi}$ and threshold τ;

step 6: if using the threshold τ yields no solution, the pricing problem has to be solved either by replacing τ with the value 0 in the `negativeReducedCost` constraint or by solving the pricing to optimality. The procedure `solvePricing` either calls the procedure `solvePricingCP` or – for dense graph – calls Cliquer to solve the pricing subproblem to optimality.

4.3 Generating δ-IP-good columns

The motivation for generating δ-IP-good columns is that solving the restricted master problem containing the columns generated by the usual pricing techniques we could obtain a solution far away from the optimal. Indeed, the pricing subproblem focuses on columns with negative reduced costs, thus potentially improving the objective function of the linear relaxation of the master, but it usually gives no clues about the impact on the integer problem.

The issue of generating good columns for either the linear or the integer master problem has been tackled in Gendron et al. (2005), where two labeling strategies for solving the CP pricing subproblem are presented: a *dual strategy* and a *master strategy*. The dual strategy consists in solving the CP pricing subproblem with a static order of the variables based on the values of the dual variables, thus, according to our notation, it focuses on generating γ-LP-good columns. The master strategy consists in adding constraints to the pricing subproblem exploiting the structure of the master problem, thus it is a strategy for generating δ-IP-good columns. Although the overall computation time is comparable with respect to the regular pricing, the master strategy gives smaller integrality gap than the dual strategy. The dual strategy mainly suffers from the *tailing-in* effect, that is the effect occurring in the first iterations of column generation when many negative reduced cost columns are introduced without improvement, while the master strategy suffers from the *tailing-off* effect, that is the very slow convergence towards the optimal solution of the relaxation, due to the introduction of columns with very small negative reduced cost Lübbecke and Desrosiers (2005). The conclusion in Gendron et al. (2005) is that a hybrid method combining the two strategies might be very effective.

We propose to use an additional pricing subproblem, called *augmented pricing*, that takes a γ-LP-good column and generates a set of δ-IP-good columns. The idea is to use two pricing subproblems: the first generates γ-LP-good columns and the second generates δ-IP-good columns. The first subproblem is the regular pricing and can be solved with any of the strategies presented previously. The second subproblem called *augmented pricing* is based on the idea of generating a set of structured columns that together with the column generated by the regular pricing contribute to define a feasible solution. The augmented pricing can be modeled as a CP problem with constraints coming both from the master and the pricing subproblems.

Consider an integer $k = \overline{\kappa}$, and the column a_p generated by the regular pricing subproblem, the augmented pricing problem looks for a $\overline{\kappa}$-partition, thus an integer solution of the master problem, that includes the

stable set described by a_p. The problem can be modeled as follows:

$$\textit{variables:} \qquad Z_i \subseteq \{1,\ldots,n\}, \quad \forall i \in \{1,\ldots,k\} \tag{25}$$

$$\textit{constraints:} \qquad Z_i \in F, \quad \forall i \in \{1,\ldots,k\} \tag{26}$$

$$Z_1 = I(a_p) \tag{27}$$

$$\texttt{partition}([Z_1,\ldots,Z_k],\{1,\ldots,n\}) \tag{28}$$

where $I(a_p)$ gives the set of indices corresponding to the non zero entries of column a_p, $n = |V|$ and F is the collection of stable sets of G.

Embedding the augmented pricing into column generation. The augmented pricing (25)–(28) is a decision problem, since no cost bounding constraint (objective function) is defined. If the upper bound k is equal to the cardinality of the optimal coloring, the augmented pricing looks for an integer solution to the original problem containing the given column. This is a very interesting feature of the augmented pricing: it avoids to solve the integer master problem at the end of column generation, since an integer solution is already available.

If the augmented pricing does not admit a solution, one possibility is to stop the CP–CG algorithm and to return the incumbent integer solution. Another option is to bypass the augmented pricing at every subsequent iteration, and to continue as long as the regular pricing generates a negative reduced cost column.

Algorithm 2 outlines the CP–CG algorithm with the augmented pricing. The differences with a regular column generation are in the steps 5–9. The logical variable *condition* in step 5 is true as long as the augmented pricing does not fail. Step 9 is used to store the incumbent integer solution and to update the upper bound.

5 Computational Results

To evaluate the algorithm performances we consider the graph coloring instances of the last challenge (see `http://mat.gsia.cmu.edu/COLOR04`). The tests are run on DELL Power Edge server, with Linux-Ubuntu server edition (64 bits), two processors with 4 cores, and 4GB of memory and the codes have been compiled with `gnu-g++` compiler (version 4.3). The results on our server of the `dfmax` benchmark used in the DIMACS challenge to compare machines are: **r200.5**: 0.04s **r300.5**: 0.37s, **r400.5**: 2.30s, **r500.5**: 8.74s.

5.1 Problem instances

There are a number of graph instances that are considered as standard benchmarks for Min–GCP used to compare the performance of different algorithms. Most of these instances were introduced in the first Graph

Algorithm 2 Column generation with an augmented pricing.

In $\tilde{A}$: subset of feasible columns of A
Var $\overline{\pi}$: dual optimal values of the RMP
Var $\underline{\kappa}, \overline{\kappa}$: integer lower and upper bounds
Var a_p, A_P : improving columns
Var *condition* : boolean variable
Out S : incumbent integer solution

```
repeat
    {z_RMP, π̄} ← solveRMP(Ã)                          ▷ solve problem (8)–(10)
    a_p ← solveRegularPricing(π̄)
    if a_p ≠ ∅ then
        if condition then                                  ▷ strategy dependent
            A_P ← solveAugmentedPricing(π̄, a_p)
        end if
        if A_P ≠ ∅ then                                    ▷ in this case a_p ∈ A_P
            S ← A_P ; κ̄ ← |A_P| ; Ã ← Ã ∪ A_P
        else
            Ã ← Ã ∪ a_p; condition =false
        end if
    end if
until a_p = ∅ or ⌈z_RMP⌉ = ⌈κ̲⌉
```

Coloring DIMACS challenge in 1994, and other instances were introduced during the second challenge held in a series of workshops between 2002 and 2004. A number of instances from the first challenge are considered *easy*, and do not provide significant information for comparing algorithms.

In this paper, we consider as *easy* all the instances that are solved to optimality by the DSATUR algorithm within a timeout of two hours on our server. We have chosen the DSATUR algorithm, since it is used as subroutine in other approaches, e.g., in Méndez-Díaz and Zabala (2006). Table 2 lists all the easy instances along with their number of nodes $|V(G)|$ and edges $|E(G)|$, density $d\%$, (maximal) clique number $\tilde{\omega}(G)$, and chromatic number $\chi(G)$ (in italic when it is the best-known value). For instances having $\omega(G) < \chi(G)$, proving optimality is a non-trivial task.

We briefly describe the instances that are still challenging for exact methods. The first class of hard instances are the random graphs `DSJc.n.p` introduced by Johnson. These instances correspond to standard random graphs, where an edge between two vertices appears with probability equal to `p`. The `DSJc.n.p` instances have probabilities equal to $p = \{0.1, 0.5, 0.9\}$ and the number of vertices equal to $n = \{125, 250, 500, 1000\}$. Similar graphs are the `Rn.p` graphs introduced by Culberson, that are quasi-random graphs. The `flat` instances were also introduced by Culberson, and are again very challenging. Another set of instances are the Leighton graphs, all with 450 vertices, but with different clique numbers. The most difficult instances are those with clique numbers equal to 15 and 25 (those with $\omega = 5$ are easy). Then, there are the `wap` instances, introduced by Koster, that are derived from frequency assignment problems, and are difficult due

to their size (around 1000 vertices).

Another type of challenging instances are those constructed so as to have an arbitrary large gap between $\omega(G)$ and $\chi(G)$. The most famous are the Mycileski graphs, which have $\omega(G) = 2$ and an arbitrary large chromatic number. The `myciel` instances are quite easy to color. Though, it is very hard to prove optimality. Similarly, the `Insertions` and `FullIns` graphs introduced by Caramia and Dell'Olmo, which are a generalization of the Mycielski graphs, are easy to color, but it is hard to prove optimality.

Finally, there are few "isolated" challenging instances: `abb313GPIA` and `ash608GPIA`, arising in matrix partitioning problems; and some latin square instances `latin_square_10`, `qgorder60` and `qgorder100` that are challenging due to their size.

5.2 CP approach

The CP approach presented in Section 2 is implemented in C++ using the Gecode 3.1 constraint system (Schulte et al., 2009). The maximal clique used in the preprocessing is found with the QUALEX-MS algorithm (Busygin, 2006), a very efficient max-clique heuristic. The coloring heuristic used for the preprocessing is part of the Boost Graph Library.

Table 2 compares the results obtained with the CP–UB and CP–LB approaches with the DSATUR algorithm on the *easy* instances. For each method the table reports the computation time in seconds, and the number of Branch-and-Bound nodes. Note the DSATUR algorithm is extremely fast on small graphs having $\omega(G) = \chi(G)$, but it takes more than 100 seconds on three instances (`2-FullIns_3`, `le450_5d`, and `mug88_25`) and more than 3000 seconds on other three instances (`1-Insertions_4`, `mug88_1`, and `quenn9_9`). CP–UB and CP–LB are never significantly worse than DSATUR both in terms of computation time and number of Branch-and-Bound nodes, even if it is based on a general purpose solver, and perform much better for DSATUR critical instances.

Table 3 reports the results of 11 instances that are solved within four minutes by the CP–LB approach, but that are not solved within two hours by DSATUR. Among these instances, `ash958GPIA` and `wap05` are solved to optimality for the first time (with respect to the recent survey Malaguti and Toth (2010)), and apart from the first two instances, the others are considered as hard instances in Méndez-Díaz and Zabala (2008). For the DSATUR algorithm, the table gives the UB found at the time limit. Note that for three instances the upper bound is not even equal to the optimal solution, while CP–UB and CP–LB always return the optimal solution.

Table 2: Instances solved by the CP–UB and CP–LB approaches within 1 minute. Comparison with the DSATUR algorithm with a timeout of 7200 seconds.

						CP-UB		CP-LB		DSATUR	
Problem	$\|V(G)\|$	$\|E(G)\|$	$d\%$	$\omega(G)$	$\chi(G)$	time	B&B nodes	time	B&B nodes	time	B&B nodes
1-FullIns_3	30	100	0.23	3	4	0.1	26	0.1	25	0.0	38
2-FullIns_3	52	201	0.15	4	5	0.0	73	0.1	83	157	$> 10^6$
1-Insertions_4	67	232	0.1	2	5	13.8	$> 10^6$	13.3	$> 10^6$	5998	$> 10^6$
2-Insertions_3	37	72	0.11	2	4	0.0	5150	0.2	6443	0.0	8859
3-Insertions_3	56	110	0.07	2	4	0.5	438713	1.5	$> 10^6$	1.0	828180
anna	138	493	0.05	11	11	0.0	128	0.0	128	0.0	128
ash331GPIA	662	4181	0.02	3	4	1.9	2647	3.3	299	0.1	2617
david	87	406	0.11	11	11	0.0	76	0.0	77	0.0	77
DSJC125.1	125	736	0.09	4	5	0.2	496	0.2	41	0.0	1429
DSJR500.1	500	3555	0.03	11	12	0.1	495	0.3	479	0.0	489
fpsol2.i.1	496	11654	0.09	65	65	0.2	1693	0.1	838	0.0	432
fpsol2.i.2	451	8691	0.09	30	30	0.3	1103	0.3	418	0.0	422
fpsol2.i.3	425	8688	0.10	30	30	0.3	933	0.3	392	0.0	396
games120	120	638	0.09	9	9	0.0	111	0.0	111	0.0	112
homer	561	1628	0.01	13	13	0.1	1540	0.1	832	0.0	549
huck	74	301	0.11	11	11	0.0	64	0.0	64	0.0	64
inithx.i.1	864	18707	0.05	54	54	0.4	3834	0.3	4526	0.1	811
inithx.i.2	645	13979	0.07	31	31	0.9	1102	0.8	712	0.0	615
inithx.i.3	621	13969	0.07	31	31	0.9	1105	0.9	1676	0.0	591
jean	80	254	0.08	10	10	0.0	70	0.0	70	0.0	71
le450_25a	450	8260	0.08	25	25	0.6	2182	0.4	413	0.0	426
le450_25b	450	8263	0.08	25	25	0.3	747	0.4	414	0.0	426
le450_5c	450	9803	0.1	5	5	1.2	3408	0.8	11	0.1	2310
le450_5d	450	9757	0.1	5	5	18.9	143615	0.8	31	161	$> 10^6$
miles1000	128	3216	0.40	42	42	0.0	91	0.0	86	0.0	87
miles1500	128	5198	0.64	73	73	0.0	61	0.0	61	0.0	56
miles250	128	387	0.05	8	8	0.0	121	0.0	120	0.0	121
miles500	128	1170	0.14	20	20	0.0	110	0.0	112	0.0	109
miles750	128	2113	0.26	31	31	0.0	118	0.0	97	0.0	98
mug88_1	88	146	0.04	3	4	0.2	129478	20.0	$> 10^6$	3335	$> 10^6$
mug88_25	88	146	0.04	3	4	0.8	782956	0.2	20440	764	$> 10^6$
mulsol.i.1	197	3925	0.20	49	49	0.1	248	0.1	689	0.0	149
mulsol.i.2	188	3885	0.22	31	31	0.0	245	0.0	346	0.0	158
mulsol.i.3	184	3916	0.23	31	31	0.0	408	0.0	243	0.0	154
mulsol.i.4	185	3946	0.23	31	31	0.0	270	0.0	304	0.0	155
mulsol.i.5	186	3973	0.23	31	31	0.0	247	0.0	394	0.0	156
myciel3	11	20	0.36	2	4	0.1	16	0.2	16	0.0	27
myciel4	23	71	0.28	2	5	0.0	160	0.2	136	0.0	848
myciel5	47	236	0.22	2	6	0.1	13317	0.3	13166	0.6	378311
qg.order30	900	26100	0.06	30	30	8.3	13654	5.7	2663	0.1	1683
queen5_5	25	160	0.53	5	5	0.0	3	0.0	3	0.0	21
queen6_6	36	290	0.46	6	7	0.2	213	0.1	70	0.0	1865
queen7_7	49	476	0.40	7	7	0.1	685	0.0	39	0.0	6849
queen8_12	96	1368	0.30	12	12	0.1	289	0.0	50	0.0	164
queen8_8	64	728	0.36	8	9	0.8	16231	1.3	46267	4.0	$> 10^6$
queen9_9	81	1056	0.33	9	10	113	$> 10^6$	1.3	55849	3731	$> 10^6$
r1000.1	1000	14378	0.03	20	20	3.2	6439	2.6	978	0.1	981
r125.1	125	209	0.03	5	5	0.0	117	0.0	116	0.3	81
r125.1c	125	7501	0.97	46	46	0.0	19	0.0	19	0.0	121
r125.5	125	3838	0.50	36	36	0.3	1591	1.3	35831	5.7	$> 10^6$
r250.1	250	867	0.03	7	8	0.0	589	0.2	513	10.8	$> 10^6$
r250.1c	250	30227	0.97	63	64	0.4	464	0.4	115	0.0	243
school1	385	19095	0.26	14	14	1.0	1624	0.4	36	0.1	372
will199GPIA	701	6772	0.03	6	7	1.8	5687	3.0	649	0.1	696
zeroin.i.1	211	4100	0.19	49	49	0.0	163	0.0	167	0.0	163
zeroin.i.2	211	3541	0.16	30	30	0.0	180	0.0	180	0.0	182
zeroin.i.3	206	3540	0.17	30	30	0.0	201	0.0	180	0.0	177

Table 3: Instances solved in about one minute by CP–LB. Comparison with CP–UB and DSATUR. Bold entries in column $\chi(G)$ denote instances optimally solved for the first time. DSATUR UB in italics denote instances where it differs from the optimal solution; for the other instances, DSATUR found the optimum but could not certify it within 7200 secs (marked with '-').

						CP-UB		CP-LB		DSATUR	
Problem	$\|V(G)\|$	$\|E(G)\|$	$d\%$	$\tilde{\omega}(G)$	$\chi(G)$	time	B&B nodes	time	B&B nodes	UB	time
1-FullIns_4	93	593	0.14	3	5	0.1	3796	0.1	2399	5	-
3-FullIns_3	80	346	0.11	5	6	0.3	63593	0.2	179	6	-
4-FullIns_3	114	541	0.08	7	7	-	$> 10^6$	0.2	359	7	-
5-FullIns_3	154	792	0.07	7	8	0.0	149	0.1	147	8	-
ash958GPIA	1916	12506	0.01	3	**4**	29.4	19839	54.7	3989	*5*	-
le450_15a	450	8168	0.08	15	15	1.1	7327	16.8	457338	*17*	-
mug100_1	100	166	0.03	3	4	0.3	27653	0.2	95020	4	-
mug100_25	100	166	0.03	3	4	0.2	65986	0.2	24554	4	-
qg.order40	1600	62400	0.05	40	40	51.6	18117	52.3	8743	*42*	-
wap05a	905	43081	0.11	50	**50**	2.8	856	3.4	856	*51*	-
myciel6	95	755	0.17	7	7	175	$> 10^6$	73.3	$> 10^6$	7	-

5.3 Column Generation Approach

Preliminary experiments showed that most of the time of the column generation algorithm is spent in solving the pricing subproblem, while solving the master takes a fraction of the whole time. Therefore, we decided to focus on the number and the quality of the generated columns. While solving the optimization pricing subproblem at each iteration does yield good columns reducing the number of iterations, the resulting overall computation time is longer. To trade off the number of generated columns and their quality, we tested the two rules to compute τ proposed in Section 4.2.2, comparing the results with those obtained fixing the threshold as proposed in Mehrotra and Trick (1996).

The column generation approach given in Section 3 is implemented in C++ using CPLEX 12.1 as Linear Programming solver, and using three different methods for the pricing subproblems. The pricing decision subproblem is solved with an implementation of model (14)–(19) using Gecode 3.1, while the pricing optimization subproblem is solved with CLIQUER (Ostergard, 2002). On big and sparse instances, we have also used the QUALEX-MS algorithm (Busygin, 2006) for computing heuristic solutions. To compute the initial set of stable sets $\overline{\mathcal{S}}$, we used the coloring heuristic part of the Boost Graph Library. The coloring heuristic is executed a few times using different random vertex ordering, and all the classes of colors found are stored in $\overline{\mathcal{S}}$. Note that before adding a class of colors to $\overline{\mathcal{S}}$, the corresponding stable set is increased to a maximal one.

5.3.1 Adaptive thresholds

Table 4 compares the proposed strategies for setting the value of τ with the approaches found in the literature. For each method, the table gives the number of generated columns and the computation time in seconds. The timeout is set to 3600 seconds. Note that indeed a threshold different from 1.0 is necessary, otherwise a

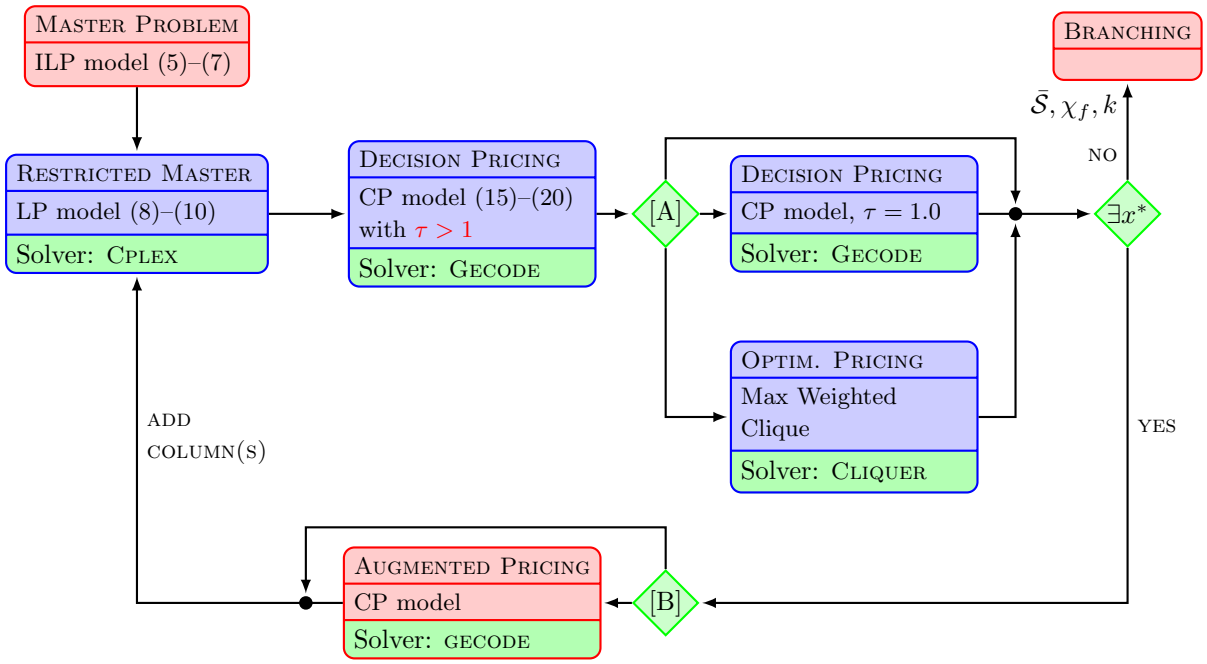


Figure 1: Sketch of the CP–CG algorithm with the augmented pricing.

huge number of columns is generated, resulting in long computation time. Fixing the threshold to 1.1, as in Mehrotra and Trick (1996), pays off for some graphs, but not in general. Moreover, we noticed that when the results obtained with the fixed threshold are comparable with those obtained with either formula (23) or (24), the corresponding values are very close to 1.1. Clearly, threshold (24) is better than both $\tau = 1.1$ and (23), since it produces less columns and has a shorter computation time, thus it has been adopted in all remaining tests. Note that the lower bounds reported in bold are improving the best values known in the literature. For the `DSJC` graphs, the lower bounds are better than those proposed in Gvozdenović and Laurent (2008).

5.3.2 Augmented pricing

The implementation of the CG–CP approach with the augmented pricing described in Section 4.3 follows the scheme given in Figure 1. The are two conditional points. In `[A]` the type of pricing problem is selected; we solve the optimization pricing (thus we update the value of τ) whenever the decision pricing with the tentative $\tau > 1$ failed. In `[B]` the application of the augmented pricing is decided: our strategy is to solve the augmented pricing in the first iterations as long as it finds a solution within a short time (e.g. two seconds); once it fails, it is applied only whenever the threshold τ is updated. Also in this case every stable set found, even by the Augmented Pricing, is (heuristically) maximally increased.

Table 5 compares the CP–CG with and without the Augmented Pricing. Since the main advantage of

Table 4: Comparing different strategies for threshold τ. Bold values in column $\lceil \chi_g(G) \rceil$ denote the new best known lower bounds. The times in bold point out the fastest approach. The timeout is set to 3600 secs denoted by '-' when reached.

						$\tau = 1.0$		$\tau = 1.1$		τ_1 (23)		τ_2 (24)	
Instance	$\|V(G)\|$	$\|E(G)\|$	$d\%$	$\tilde{\chi}(G)$	$\lceil \chi_f(G) \rceil$	Iter	Time	Iter	Time	Iter	Time	Iter	Time
1-Insertions_4	67	232	0.10	5	3	569	1.6	320	1.8	544	4.7	27	**0.03**
1-Insertions_5	202	1227	0.06	6	4	7249	-	1089	-	2754	-	644	**361**
1-Insertions_6	607	6337	0.03	7	?	1311	-	404	-	543	-	322	-
2-Insertions_4	149	541	0.05	5	3	3836	-	820	-	1795	-	45	**0.18**
2-Insertions_5	597	3936	0.02	6	?	1953	-	759	-	194	-	290	-
3-Insertions_3	56	110	0.07	4	3	455	0.94	298	1.8	366	3.4	38	0.04
3-Insertions_4	281	1046	0.03	5	3	4846	-	506	-	254	-	122	**9.7**
3-Insertions_5	1406	9695	0.01	6	?	1040	-	421	-	188	-	203	-
4-Insertions_3	79	156	0.05	4	3	1426	29	687	112	715	176	938	**33**
4-Insertions_4	475	1795	0.02	5	?	3682	-	413	-	228	-	238	-
1-FullIns_4	93	593	0.14	5	4	112	0.11	75	0.09	57	0.08	28	**0.05**
1-FullIns_5	282	3247	0.08	6	4	1703	22	356	3.7	393	4.9	52	**0.46**
2-FullIns_4	212	1621	0.07	6	5	673	2.9	168	0.72	59	0.22	47	**0.18**
2-FullIns_5	852	12201	0.03	7	5	11109	-	806	59	363	29	102	**7.0**
3-FullIns_3	80	346	0.11	6	6	51	0.04	10	0	15	0.01	6	0.01
3-FullIns_4	405	3524	0.04	7	6	5502	181	278	3.6	94	1.2	61	**1.1**
4-FullIns_3	114	541	0.08	7	7	88	0.11	23	0.03	16	0.02	12	0.03
4-FullIns_4	690	6650	0.03	8	7	3188	182	255	9.7	119	3.8	82	**3.0**
5-FullIns_3	154	792	0.07	8	8	47	0.08	19	0.04	20	0.04	17	0.07
5-FullIns_4	1085	11395	0.02	9	8	1717	-	343	-	325	116	121	**14**
DSJC1000.5	1000	249826	0.50	83	?	1376	-	614	-	49	-	186	-
DSJC1000.9	1000	449449	0.90	224	**215**	6970	984	3046	564	6970	1467	1705	**311**
DSJC125.5	125	3891	0.50	17	**16**	1239	6.8	561	4.0	1043	12	290	**1.7**
DSJC125.9	125	6961	0.90	44	**43**	77	0.12	63	0.10	73	0.16	51	0.12
DSJC250.5	250	15668	0.50	28	**26**	5096	360	1703	190	5096	1115	838	**72**
DSJC250.9	250	27897	0.90	72	**71**	479	3.0	298	2.2	479	4.5	214	**1.8**
DSJC500.5	500	62624	0.50	48	?	7794	-	2683	-	2992	-	1404	-
DSJC500.9	500	112437	0.90	126	**123**	1454	45	944	34	1454	67	674	**23**
DSJR500.5	500	58862	0.47	122	**122**	3238	429	341	54	530	81	123	**29**
flat1000_50_0	1000	245000	0.49	50	?	1412	-	1115	-	74	-	1786	-
flat1000_60_0	1000	245830	0.49	60	?	1188	-	581	-	51	-	287	-
flat1000_76_0	1000	246708	0.49	82	?	1090	-	575	-	52	-	154	-
flat300_20_0	300	21375	0.48	20	**20**	12052	822	4646	**256**	6769	979	8896	1194
flat300_26_0	300	21633	0.48	26	**26**	9779	1501	3372	942	8443	-	1945	**302**
flat300_28_0	300	21695	0.48	28	**28**	9359	1843	2338	966	7955	-	986	**251**
myciel5	47	236	0.22	6	**4**	160	0.11	79	0.05	160	0.17	22	0.02
myciel6	95	755	0.17	7	**4**	381	0.77	192	0.42	381	1.5	98	0.18
myciel7	191	2360	0.13	8	**5**	1274	11	398	6.8	1274	27	318	**4.1**
r1000.1c	1000	485090	0.97	98	**96**	779	27	612	23	525	22	463	**19**
r1000.5	1000	238267	0.48	234	**234**	871	-	646	-	664	-	611	**2434**
r250.1	250	867	0.03	8	**8**	1	0.02	1	0.02	1	0.01	1	0.02
r250.5	250	14849	0.48	65	**65**	904	9.7	139	2.1	232	3.3	54	**1.5**

having the Augmented Pricing consists of updating the upper bound within the column generation algorithm, we compare the Augmented Pricing with a so-called *Price-and-Branch* algorithm, where we solve the restricted integer master problem with the set of columns returned by the column generation algorithm, denoted by Restricted IP in the table. In this comparison, we have to pay attention to the following:

- Both methods start with an upper bound equal to UB_1 that is determined as described in Section 2, using the graph coloring heuristic present in the Boost Graph Library.
- Both methods solve the column generation to optimality to get a lower bound (with a time limit of 1 hour), given in the sixth column of Table 5. The main difference is that when the column generation with the augmented pricing ends, it has eventually computed a new upper bound UB_3, while the Restricted IP method has still UB_1 as upper bound.
- After the column generation algorithm stops, both methods solve the restricted integer master program, but indeed with a different set of columns. Let UB_2 and UB_4 denote the upper bounds obtained with those columns.

It is interesting to notice that in many instances, the value of UB_3 improves that of UB_1 (see for instance `3-FullIns-4` and `DSJC125.9`), which means that the augmented pricing has computed better integer solutions at roughly the same time needed for the execution of a regular column generation. Moreover, in other instances, the final upper bound UB_2 is not better than UB_3, even if it has required a lot of additional computation time (see for instance `DSJC125.5`). There is only one case where the Restricted IP heuristic does perform better, that is for the instance `DSJC250.9`, for which the Restricted IP heuristic finds the best known upper bound. Note that with the augmented pricing, we could avoid to solve the restricted integer master problem.

5.4 Branch-and-Price

The column generation algorithm provides tight lower bounds, but to compute the integer optimal solution we need to set up an enumeration based on branching. Even if several general branching rules in the context of column generation exist (Vanderbeck, 2009), Min–GCP admits a simple, but effective, branching scheme originally due to Zykov (1949) that mainly consists of:

1. Select two non-adjacent vertices v and w.
2. **Either** v and w belong to the same class of colors, i.e., they merge in a single node, corresponding to adding the edges between v and all neighbors of w, and removing w.
3. **Or** v and w cannot belong to the same class of colors, corresponding to adding edge $\{v, w\}$.

Table 5: Solving the Restricted Integer Master Problem (RIP) with and without Augmented–Pricing.

						Restricted IP					Augmented Pricing				
						CG			IP		CG			IP	
Instance	$\|V(G)\|$	$\|E(G)\|$	$d\%$	$\tilde{\chi}(G)$	$\lceil\chi_f(G)\rceil$	Time	Iter	UB_1	Time	UB_2	Time	Iter	UB_3	Time	UB_4
1-Insertions_4	67	232	0.10	5	3	0.0	30	5	0.3	5	0.1	30	5	0.3	5
1-Insertions_5	202	1227	0.06	6	3	1991	1063	6	5596	6	2083	1025	6	5688	6
2-Insertions_3	37	72	0.11	4	3	0.0	12	4	0.0	4	0.0	8	4	0.0	4
2-Insertions_4	149	541	0.05	5	3	0.4	53	5	1.3	5	6.2	54	5	6.9	5
3-Insertions_3	56	110	0.07	4	3	0.1	38	4	0.2	4	0.1	32	4	0.2	4
3-Insertions_4	281	1046	0.03	5	3	6.8	91	5	9.5	5	53	135	5	59	5
4-Insertions_3	79	156	0.05	4	3	0.2	78	4	0.6	4	0.4	79	4	0.8	4
1-FullIns_4	93	593	0.14	5	4	0.1	29	5	0.1	5	0.1	29	5	0.2	5
1-FullIns_5	282	3247	0.08	6	4	3.0	179	7	5.2	6	10.0	183	6	12.8	6
2-FullIns_4	212	1621	0.07	6	5	0.6	67	7	1.0	6	1.2	58	6	1.5	6
3-FullIns_3	80	346	0.11	6	6	0.0	11	6	0.0	6	0.0	16	6	0.0	6
3-FullIns_4	405	3524	0.04	7	6	4.1	117	8	6.9	7	15	80	7	16	7
4-FullIns_3	114	541	0.08	7	7	0.0	18	7	0.0	7	0.1	14	7	0.1	7
4-FullIns_4	690	6650	0.03	8	7	18	177	8	20	8	34	176	8	36	8
5-FullIns_3	154	792	0.07	8	8	0.1	17	8	0.1	8	4.1	17	8	4.1	8
5-FullIns_4	1085	11395	0.02	9	8	502	449	10	651	9	170	372	9	1035	9
DSJC1000.9	1000	449449	0.90	224	215	344	1701	314	3944	245	468	1701	314	4069	245
DSJC125.5	125	3891	0.50	17	16	1.7	254	25	815	19	21	260	19	2128	19
DSJC125.9	125	6961	0.90	44	43	0.1	51	53	0.6	44	7.1	44	49	7.6	44
DSJC250.5	250	15668	0.50	28	26	72	827	41	3674	33	101	850	35	3702	33
DSJC250.9	250	27897	0.90	72	71	2.0	218	94	389	72	11	171	84	3613	73
DSJC500.9	500	112437	0.90	126	123	25	657	172	3626	133	50	657	172	3651	133
DSJR500.1c	500	121275	0.97	85	85	0.8	86	106	1.0	85	5.8	57	87	6.3	85
DSJR500.5	500	58862	0.47	122	122	94	538	144	3695	130	15	6	127	3615	129
flat300_20_0	300	21375	0.48	20	20	900	7557	45	900	20	906	7687	39	906	20
flat300_26_0	300	21633	0.48	26	26	305	1723	46	305	26	330	1747	39	330	26
flat300_28_0	300	21695	0.48	28	28	277	1022	46	3878	38	322	970	39	3923	38
myciel5	47	236	0.22	6	4	0.0	19	6	0.2	6	0.0	19	6	0.2	6
myciel6	95	755	0.17	7	4	0.3	117	7	20	7	0.4	90	7	8.9	7
myciel7	191	2360	0.13	8	5	5.0	314	8	3614	8	24	316	8	3632	8
queen10_10	100	2940	0.59	11	10	3.9	394	14	669	12	11	370	12	216	12
queen11_11	121	3960	0.55	11	11	15	497	16	3619	14	21	464	13	3627	15
queen12_12	144	5192	0.50	12	12	65	570	17	3668	16	106	603	14	3710	16
queen13_13	169	6656	0.47	13	13	341	672	18	3943	18	379	674	16	3982	18
r1000.1c	1000	485090	0.97	98	96	25	548	131	1176	98	38	538	131	706	98
r125.1c	125	7501	0.97	46	46	0.0	1	47	0.0	46	0.0	1	47	0.0	46
r125.5	125	3838	0.50	36	36	0.2	60	42	0.6	37	0.1	2	38	0.3	36
r250.1	250	867	0.03	8	8	0.0	1	8	0.1	8	0.0	1	8	0.1	8
r250.5	250	14849	0.48	65	65	4.5	239	77	3607	67	4.4	52	67	37	66

This branching scheme is adopted in Mehrotra and Trick (1996), where an interpretation as a Ryan-Foster (Ryan and Foster, 1981) branching is proposed. The advantage of this branching scheme is that at each node of the Branch-and-Price tree we are facing an instance of MIN–GCP, though on a different graph.

The critical issue of this branching scheme is how vertices v and w are selected. The branching used in Mehrotra and Trick (1996) considers the stable set corresponding to the most fractional variable λ_i^* of the restricted master and another stable set corresponding to another column such that the two sets share a vertex v and there is another vertex w belonging to one of the two sets only. We consider a similar selection rule, but in choosing the second vertex w we break ties using the vertex degree.

Another critical issue in the Branch-and-Price algorithm is the order for visiting the nodes of the search tree. Since the most important decisions are made in the first branching steps, i.e., the top of the search tree, we have adopted an iterative depth-first search. That is, first, we fix a maximal depth; second, we entirely visit the search tree up to the fixed depth; third, in case the search tree is not closed (by closing the lower and upper bound gap) we increase the depth by one and restart the Branch-and-Price algorithm (almost) from scratch. Note that every time the execution restarts, a new upper bound is computed. We could save some computation time by storing the node that are still open at the given depth, but this would consume much more memory, since the number of open nodes is exponential with the depth of the search tree. In addition, in order to diversify the branching decisions each time the search restarts, we randomly generate the columns of the initial pool of columns that form the initial feasible solution. Moreover, we use the ILP solver also as a sort of primal heuristic: at each node of the search tree, we let CPLEX solve the restricted master problem with a branch-and-bound node limit set to 100 (see, parameter `MIP node limit` in CPLEX manuals), and with an upper bound set to the current upper bound -1.

Table 6 shows the results we obtain for some hard instances that in the recent survey of Malaguti and Toth (2010) are reported as challenging for exact methods. The first six columns describe the instance features. In addition, the column "BEUB." gives the best results among those obtained with the exact methods, as reported in Malaguti and Toth (2010). Note that `DSJC125.5`, `DSJC125.9`, `DSJC500.1c`, `flat300_20_0`, and `flat300_26_0` are solved to optimality for the first time with an exact method. For three other instances, `DSJC250.5`, `DSJC250.9`, and `DSJC500.9` we provide the best known lower bound obtained via column generation (see also Table 4) and the best upper bound obtained with an exact methods, though, for these last three instances, meta-heuristics provide better upper bounds.

Table 6: Hard instances solved by our Branch-and-Price. Comparison with the best results found in the literature with an exact method (column "BEUB") as found in Malaguti and Toth (2010) and by heuristic algorithms (column"bks"). Note that for the instances `queen10_10` and `queen11_11` the optimum values are reported in Vasquez (2004).

Instance	$\|V(G)\|$	$\|E(G)\|$	$d\%$	Bks	$\lceil\chi_f(G)\rceil$	BEUB	UB	Time	Nodes
queen9_9	81	1056	0.33	10	10	10	10	98	84
queen10_10	100	2940	0.59	11	10	12	11	34951	2271
queen11_11	121	3960	0.55	11	11	13	11	28946	1207
DSJC125.5	125	3891	0.50	17	16	19	**17**	19033	2649
DSJC125.9	125	6961	0.90	44	43	45	**44**	44	423
DSJC250.5	250	15668	0.50	*28*	**26**	35	31	timeout	
DSJC250.9	250	27897	0.90	*72*	**71**	87	72	timeout	
DSJC500.9	500	112437	0.90	*126*	**123**	160	136	timeout	
DSJR500.1c	500	121275	0.90	85	**85**	88	**85**	0.8	1
DSJR500.5	500	58862	0.48	122	**122**	130	125	timeout	
flat300_20_0	300	21375	0.48	20	**20**	n.a.	**20**	683	1
flat300_26_0	300	21633	0.48	26	**26**	n.a.	**26**	289	1
flat300_28_0	300	21633	0.48	28	**28**	n.a.	34	timeout	
r1000.1	1000	14378	0.13	20	20	n.a.	**20**	1.0	1
r1000.5	1000	238267	0.48	234	**234**	n.a.	261	timeout	
r1000.1c	1000	485090	0.97	98	**96**	n.a.	98	timeout	
r250.5	250	14849	0.48	65	**65**	n.a.	**65**	9	75

6 Graph Multicoloring Problem

6.1 Formulation

Min–GMP differs from Min–GCP because it requires that each vertex v is assigned to b_v different colors, i.e. to a given number of stable sets. Let $\mathcal{S}$ be the collection of all maximal stable sets in $G = (V, E)$, and $\mathcal{S}_v$ be the collection of maximal stable sets containing $v \in V$. The set covering formulation of Min–GMP is:

$$z_{MP} = \min \quad \sum_{i \in \mathcal{S}} \lambda_i \tag{29}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{S}_v} \lambda_i \geq b_v, \qquad \forall v \in V, \tag{30}$$

$$\lambda_i \geq 0, \quad \text{integer}, \qquad \forall i \in \mathcal{S}. \tag{31}$$

Differently from Min–GCP formulation, here variables λ_i are integer and represent the number of times the i-th class of colors is used in the solution. However, the pricing subproblem is the same as for Min–GCP, and Min–GMP can be solved with the CP–CG approach given in Section 4 using (29)–(31) as master problem. This approach was investigated in Mehrotra and Trick (2007) extending the work presented in Mehrotra and Trick (1996).

Note that the graph reduction presented in Section 2.1 can be extended to Min–GMP: given the weight $\omega_m(G)$ of a maximum weighted clique, we can remove every vertex having the sum of the weights of its neighbours plus its own weight smaller than $\omega_m(G)$.

6.2 Branching scheme

The branching rule for MIN–GMP proposed in Mehrotra and Trick (2007) is sophisticated and the addition of the branching constraints to the restricted master problem spoils the pricing subproblem structure.

Here, we take advantage of the new branching scheme introduced in Gualandi and Malucelli (2010) that generalizes the Zykov branching rule to MIN–GMP. This allows to maintain the same problem structure at each node of the Branch-and-Price tree. The branching scheme is as follows:

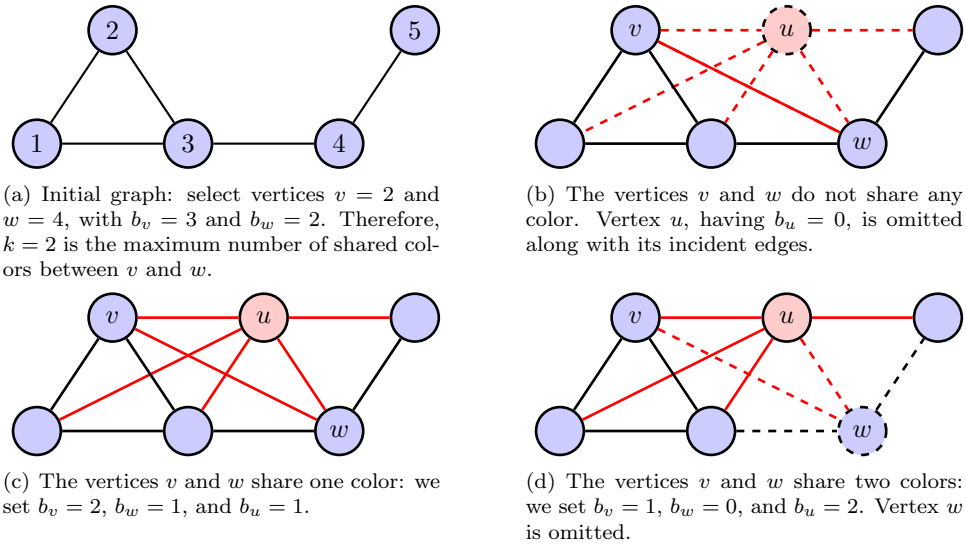
1. Two non-adjacent vertices v and w are selected, requiring b_v and b_w colors, respectively.
2. A dummy vertex u with $b_u = 0$, is introduced together with one edge $\{u, u'\}$ for each $u' \in N(v) \cup N(w)$.
3. Let $k = \min\{b_v, b_w\}$ be the maximum number of colors that v and w can share. There are $k + 1$ alternative feasible cases depending on the exact number of colors that v and w have in common, from 0 up to k. In order to explore these $k + 1$ possibilities, we create $k + 1$ branching nodes as follows.
4. **Case** $k = 0$ v and w cannot belong to the same class of colors. This corresponds to add the edge $\{v, w\}$. Note that in this case the introduction of vertex u can be omitted.
5. **Case** $t = 1, \ldots, k$, vertices v and w have t colors in common, thus $b_v \leftarrow b_v - t$, $b_w \leftarrow b_w - t$, and $b_u \leftarrow b_u + t$.

Note that when all the b_i are equal to one for all vertices i, this branching scheme reduces to the Zykov one.

Figure 2 shows an example of this branching scheme on a small graph with 5 vertices. In each branching node we have always an instance of the MIN–GMP problem, and therefore we can reuse the same column generation algorithm, without modifying the pricing subproblems.

6.3 Computational Results

Table 7 shows the results for MIN–GMP obtained with our Branch-and-Price algorithm for the so–called geometric instances. The table reports the result of the preprocessing, that is, the weight of the maximum weighted clique $\omega_m(g)$, and the number of vertices and edges after the graph reduction, that is, $|V_r(G)|$ and $|E_r(G)|$, respectively. Then, the table reports for the column generation, the lower bound LB, the computation time in seconds, and the number of iterations, i.e., the number of generated columns. For the Branch-and-Price, the table shows the optimum value Opt, the number of branching nodes, and the computation time. Note that these are easy instances, since only eight of them require branching. Our results are in line with those reported in Mehrotra and Trick (2007), but since in addition we have implemented the branching phase, we are able prove the optimality for all the instances, while in Mehrotra and Trick (2007) 11 were left open.



(a) Initial graph: select vertices $v = 2$ and $w = 4$, with $b_v = 3$ and $b_w = 2$. Therefore, $k = 2$ is the maximum number of shared colors between v and w.

(b) The vertices v and w do not share any color. Vertex u, having $b_u = 0$, is omitted along with its incident edges.

(c) The vertices v and w share one color: we set $b_v = 2$, $b_w = 1$, and $b_u = 1$.

(d) The vertices v and w share two colors: we set $b_v = 1$, $b_w = 0$, and $b_u = 2$. Vertex w is omitted.

Figure 2: Example of the branching rule for the MIN–GMP problem: (a) the initial graph, and (b)–(d) the three branching nodes, when the vertices 2 and 4 are selected for branching.

Table 8 reports the results obtained using the miscellaneous instances introduced in the DIMACS challenge. The best known results (column Bks) are obtained by taking the best results as reported in Mehrotra and Trick (2007) and in Prestwich (2008). Note that 22 out of 40 instances are solved at the root node, since the lower bound equals the upper bound. For 18 instances, branching is indeed necessary. 13 instances out of 18 are solved to optimality with our Branch-and-Price algorithm. Remarkably, our algorithm requires a small number of branching nodes for the instances that is able to solve. In addition, 13 instances are solved to optimality for the first time.

Apparently most of MIN–GMP literature instances are too small and relatively easy. For this reason we generated a set of new more challenging instances derived from conflict graphs generated by the GLPK Mixed Integer Linear Programming solver separating clique inequalities in solving MIPLIB2003 benchmarks. Vertex weights are those assigned at the first call of the separation algorithm scaled and rounded in the range [1..11]. We have omitted instances having more than 20000 vertices in the original graph and less than 60 vertices after preprocessing.

Table 9 shows the results obtained with our Branch-and-Price on the new **CO**nflict **G**raph (COG) instances. The first five columns describe each instance: note that the number of vertices can be very high, while the density is always very small. Nevertheless, after the graph reduction based on the maximum weighted clique (computed with QUALEX), the resulting graphs are much smaller. In the new benchmarks, there are four instances for which our Branch-and-Price algorithm reaches the timeout in the branching phase, and five cases when the column generation algorithm does not even terminate at the root node. The remaining 9 instances are solved to optimality. Our algorithm main difficulty emerges in the solution of

Table 7: Branch-and-Price results on the geometric instances. In bold the optimal results proved for the first time.

					Preprocessing			Column Generation			Branch-and-Price		
Instance	$\|V(G)\|$	$\|E(G)\|$	$d\%$	$\|K\|$	$\omega_m(G)$	$\|V_r(G)\|$	$\|E_r(G)\|$	LB	Time	Iter	Opt.	Nodes	Time
GEOM20	20	20	0.1	10	28	5	10	28	0.0	1	28	1	0.0
GEOM20a	20	37	0.2	10	30	8	21	30	0.0	1	30	1	0.0
GEOM20b	20	32	0.2	3	8	3	3	8	0.0	1	8	1	0.0
GEOM30	30	50	0.1	10	26	6	15	26	0.0	1	26	1	0.0
GEOM30a	30	81	0.2	10	40	5	10	40	0.0	1	40	1	0.0
GEOM30b	30	81	0.2	3	11	4	6	11	0.0	1	11	1	0.0
GEOM40	40	78	0.1	10	31	5	10	31	0.0	1	31	1	0.0
GEOM40a	40	146	0.2	10	46	6	15	46	0.0	1	46	1	0.0
GEOM40b	40	157	0.2	3	14	28	109	14	0.0	1	14	1	0.0
GEOM50	50	127	0.1	10	35	5	10	35	0.0	1	35	1	0.0
GEOM50a	50	238	0.2	10	61	14	72	61	0.0	1	61	1	0.0
GEOM50b	50	249	0.2	3	17	19	81	17	0.0	1	**17**	1	0.0
GEOM60	60	185	0.1	10	36	6	15	36	0.0	1	36	1	0.0
GEOM60a	60	339	0.2	10	65	10	45	65	0.0	1	65	1	0.0
GEOM60b	60	366	0.2	3	22	11	46	22	0.0	1	22	1	0.0
GEOM70	70	267	0.1	10	44	8	28	44	0.0	1	44	1	0.0
GEOM70a	70	459	0.2	10	71	11	55	71	0.0	1	71	1	0.0
GEOM70b	70	488	0.2	3	22	12	60	22	0.0	1	**22**	1	0.0
GEOM80	80	349	0.1	10	63	7	21	63	0.0	1	63	1	0.0
GEOM80a	80	612	0.2	10	68	14	85	68	0.0	1	68	1	0.0
GEOM80b	80	663	0.2	3	25	37	298	25	0.0	1	**25**	1	0.0
GEOM90	90	441	0.1	10	51	8	28	51	0.0	1	**51**	1	0.0
GEOM90a	90	789	0.2	10	65	56	454	65	0.0	2	**65**	1	0.0
GEOM90b	90	860	0.2	3	28	61	587	28	0.0	1	**28**	1	0.2
GEOM100	100	547	0.1	10	60	9	36	60	0.0	1	60	1	0.0
GEOM100a	100	992	0.2	10	81	15	96	81	0.0	1	81	1	0.0
GEOM100b	100	1050	0.2	3	30	63	661	30	0.0	2	**30**	1	0.0
GEOM110	110	638	0.1	10	62	20	109	62	0.0	1	**62**	1	0.0
GEOM110a	110	1207	0.2	10	91	13	78	91	0.0	1	**91**	1	0.0
GEOM110b	110	1256	0.2	3	37	15	105	37	0.0	1	37	1	0.0
GEOM120	120	773	0.1	10	63	33	208	64	0.0	1	64	1	0.0
GEOM120a	120	1434	0.2	10	93	95	1097	93	0.1	13	**93**	41	20.1
GEOM120b	120	1491	0.2	3	34	51	568	34	0.0	1	**34**	1	0.0

the pricing subproblem. For example, for the `COG-nsrand-ipx` five iterations of the column generation, corresponding to the solution of five pricing problems require one hour of computation.

7 Conclusions

We presented a computational study for Vertex Coloring Problems using two approaches Constraint Programming and Column Generation, both as stand-alone methods and combined into a non-trivial hybrid approach. A number of techniques improving the original implementations are also introduced. The resulting algorithms provide very interesting results on almost all the instances. Considering the Min–GCP in particular, with respect with the best results presented in the literature and summarized in Malaguti and Toth (2010), the main achievements can be summarized as follows:

1. *CP–UB* approach (or *CP–LB*, though with longer computation times) is able to prove the optimality

Table 8: Branch-and-Price results on several MIN–GMP instances. In italics the upper bound of the open instances, in bold the optimal results proved for the first time.

						Preprocessing			Column Generation			Branch-and-Price		
Instance	$\|V(G)\|$	$\|E(G)\|$	$d\%$	$\|K\|$	Bks	$\omega_m(G)$	$\|V_r(G)\|$	$\|E_r(G)\|$	LB	Time	Iter	UB	Nodes	Time
myciel5g	47	236	0.2	5	14	10	47	236	14	0.0	45	14	1	0.0
myciel5gb	47	236	0.2	20	45	37	47	236	45	0.0	31	45	1	0.1
myciel6g	95	755	0.2	5	16	10	95	755	16	0.2	89	16	1	0.2
myciel6gb	95	755	0.2	20	58	39	95	755	58	0.2	100	58	1	0.2
myciel7g	191	2360	0.1	5	17	10	191	2360	17	1.2	175	*18*	1052	-
myciel7gb	191	2360	0.1	20	63	40	191	2360	63	1.2	182	63	1	3.2
queen8_8g	64	728	0.4	5	28	28	64	728	28	0.0	20	**28**	22	5.0
queen8_8gb	64	728	0.4	20	113	113	64	728	113	0.0	4	113	1	0.0
queen9_9g	81	1056	0.3	5	35	35	81	1056	35	0.0	8	**35**	50	10
queen9_9gb	81	1056	0.3	20	135	135	81	1056	135	0.0	9	135	1	0.3
queen10_10g	100	1470	0.3	5	38	38	100	1470	38	0.1	12	**38**	222	142
queen10_10gb	100	1470	0.3	20	136	136	100	1470	136	0.0	12	136	18	4.0
queen11_11g	121	1980	0.3	5	41	41	121	1980	41	0.2	19	**41**	84	119
queen11_11gb	121	1980	0.3	20	140	140	121	1980	140	0.3	38	**140**	27	21
queen12_12g	144	2596	0.3	5	42	42	144	2596	42	1.3	71	*44*	4952	-
queen12_12gb	144	2596	0.3	20	163	163	144	2596	163	0.2	20	**163**	10	7.6
R50_1g	50	108	0.1	5	12	12	21	42	12	0.0	1	12	1	0.0
R50_1gb	50	108	0.1	20	45	45	7	10	45	0.0	1	45	1	0.0
R50_5g	50	612	0.5	5	29	27	50	612	29	0.0	49	29	1	0.1
R50_5gb	50	612	0.5	20	100	98	50	612	100	0.1	54	100	1	0.1
R50_9g	50	1092	0.9	5	64	64	50	1092	64	0.0	2	64	1	0.0
R50_9gb	50	1092	0.9	19	228	228	50	1092	228	0.0	2	228	1	0.0
R75_1g	70	251	0.1	5	14	14	64	234	14	0.0	37	**14**	40	3.0
R75_1gb	70	251	0.1	20	53	53	63	229	53	0.0	21	**53**	1	0.0
R75_5g	75	1407	0.5	5	38	31	75	1407	38	0.2	123	38	1	0.3
R75_5gb	75	1407	0.5	20	131	114	75	1407	131	0.2	124	131	1	0.5
R75_9g	75	2513	0.9	5	94	85	75	2513	94	0.0	3	94	1	0.0
R75_9gb	75	2513	0.9	20	328	298	75	2513	328	0.0	4	328	1	0.0
R100_1g	100	509	0.1	5	16	15	98	501	15	0.4	108	**15**	993	1841
R100_1gb	100	509	0.1	20	56	56	98	501	56	0.2	81	**56**	20	2.1
R100_5g	100	2456	0.5	5	43	35	100	2456	42	0.6	212	*43*	63596	-
R100_5gb	100	2456	0.5	20	153	132	100	2456	153	0.5	198	153	4320	1162
R100_9g	100	4438	0.9	5	118	108	100	4438	118	0.0	19	118	1	0.1
R100_9gb	100	4438	0.9	20	422	390	100	4438	422	0.0	13	422	1	0.0
DSJC125.1g	125	736	0.1	5	19	19	123	725	19	0.1	30	**19**	3105	2992
DSJC125.1gb	125	736	0.1	20	67	67	122	717	67	0.1	38	**67**	63	20
DSJC125.5g	125	3891	0.5	5	55	40	125	3891	53	1.3	292	*54*	25185	-
DSJC125.5gb	125	3891	0.5	20	164	125	125	3891	162	1.2	281	*163*	21260	-
DSJC125.9g	125	6961	0.9	5	140	122	125	6961	139	0.1	47	**139**	1	0.1
DSJC125.9gb	125	6961	0.9	20	497	425	125	6961	497	0.1	41	497	1	0.1

Table 9: Branch-and-Price results on the new COG instances. In bold the optimal results; time limit set to 3600 sec.

					Preprocessing			Column Generation		Branch-and-Price			
Instance	$\|V(G)\|$	$\|E(G)\|$	$d\%$	$\|K\|$	$\omega_m(G)$	$\|V_r(G)\|$	$\|E_r(G)\|$	LB	Time	Iter	UB	Nodes	Time
COG-10teams	3200	124480	0.02	11	73	1600	122880	-	-	1	-	-	-
COG-air04	17808	2121648	0.01	11	377	368	67528	377	1.7	1	**377**	1	1.8
COG-air05	14390	2527253	0.02	11	413	5295	1913983	-	-	1	-	-	-
COG-atlanta-ip	8124	14607	0.00	11	15	174	271	15	0.1	23	**15**	2573	1844
COG-cap6000	11992	12103	0.00	11	14	64	96	14	0.2	80	**14**	337	304
COG-ds	15252	2057486	0.02	11	500	915	246525	500	6.1	1	**500**	1	6.5
COG-gesa2-o	192	144	0.01	11	12	128	80	12	2.1	434	13	1444	-
COG-misc07	410	2928	0.03	11	36	162	2286	36	2.5	4	39	549	-
COG-mkc	10394	154870	0.00	11	169	157	12246	169	0.0	1	**169**	1	0.1
COG-mod011	192	336	0.02	11	12	160	240	12	2.7	353	13	1777	-
COG-mzzv11	19942	257012	0.00	11	101	92	4186	101	0.1	1	**101**	1	0.1
COG-mzzv42z	18806	225687	0.00	11	91	83	3403	91	0.0	1	**91**	1	0.1
COG-net12	3202	4835	0.00	11	17	88	204	17	0.0	23	**17**	2583	1301
COG-nsrand-ipx	13240	69510	0.00	11	30	240	2280	-	-	5	-	-	-
COG-opt1217	1536	6528	0.01	11	26	624	4680	-	-	3	-	-	-
COG-rd-rplusc-21	904	11785	0.03	11	109	100	4950	109	0.0	1	**109**	1	0.0
COG-rout	560	2940	0.02	11	30	120	1140	30	4.9	34	32	1435	-
COG-swath	12480	958000	0.01	11	317	6160	948600	-	-	1	-	-	-

of two open instances `ash958GPIA` and `wap05a`. In addition, it is able to compute the optimal solutions for `myciel6`, `qg.order40`, and `le450_15a`, for which the optimum is known by construction, but existing exact methods cannot prove optimality.

2. The Constraint Programming-based Column Generation algorithm is able to provide the best known lower bounds for 17 hard instances (see Table 4).

3. Our Branch-and-Price implementation is able to close 7 open instances (see Table 6).

Considering the multicoloring problem, we propose the implementation of a Branch-and-Price that exploits a branching scheme able to maintain the problem structure and exploiting all the successful techniques adopted for the vertex coloring. The algorithm is able to close all but five of the 73 benchmark instances of Min–GMP (see Tables 7 and 8). Given these results we introduced a new class of more challenging instances.

This represents the state-of-the-art of exact methods for Minimum Vertex Coloring and Multicoloring problems. These results and the possible updates are published in the website `http://sites.google.com/site/graphcoloring`.

Acknowledgments

The authors thank Marco Chiarandini for useful discussions and two anonymous referees for their suggestions.

References

Apt, K.R. 2003. *Principles of Constraint Programming*. Cambridge University Press.

Barnier, N., P. Brisset. 2004. Graph coloring for air traffic flow management. *Ann. Oper. Res.* **130** 163–178.

Brélaz, D. 1979. New methods to color the vertices of a graph. *Commun. ACM* **22** 251–256.

Busygin, S. 2006. A new trust region technique for the maximum weight clique problem. *Disc. Appl. Math.* **154** 2080–2096.

Capone, A., G. Carello, I. Filippini, S. Gualandi, F. Malucelli. 2010. Solving a resource allocation problem in wireless mesh networks: a comparison between a CP-based and a classical column generation. *Networks* doi:10.1002/net.20367.

Coleman, T.F., J.J. More. 1984. Estimation of sparse Hessian matrices and graph coloring problems. *Math. Program.* **28** 243–270.

Coll, P., J. Marenco, I. Mendez-Diaz, P. Zabala. 2002. Facets of the graph coloring polytope. *Ann. Oper. Res.* **116** 79–90.

Demassey, S., G. Pesant, L.M. Rousseau. 2006. A cost-regular based hybrid column generation approach. *Constraints* **11** 315–333.

Easton, K., G.L. Nemhauser, M.A. Trick. 2002. Solving the travelling tournament problem: A combined integer programming and constraint programming approach. *Proc. Practice Theory Automated Timetabling*, *LNCS*, vol. 2740. Springer, 100–112.

Fahle, T. 2002. Simple and fast: Improving a branch-and-bound algorithm for maximum clique. *Proc. Ann. Eur. Symp. Algorithms*. Springer, 47–86.

Fahle, T., U. Junker, S.E. Karisch, N. Kohl, M. Sellmann, B. Vaaben. 2002. Constraint programming based column generation for crew assignment. *J. Heuristics* **8** 59–81.

Fahle, T., M. Sellmann. 2002. Cost based filtering for the constrained knapsack problem. *Ann. Oper. Res.* **115** 73–93.

Gabteni, S., M. Grönkvist. 2006. A hybrid column generation and constraint programming optimizer for the tail assignment problem. *Proc. Integration of AI and OR Techniques in CP for Combin. Optim.*, *LNCS*, vol. 3990. Springer, 89–103.

Gendron, B., H. Lebbah, G. Pesant. 2005. Improving the cooperation between the master problem and the subproblem in constraint programming based column generation. *Proc. Integration of AI and OR Techniques in CP for Combin. Optim.*, *LNCS*, vol. 3524. Springer, 217–227.

Gent, I.P., K.E. Petrie, J.-F. Puget. 2006. *Handbook of Constraint Programming*, chap. 10. Symmetry in Constraint Programming. Elsevier Science, 329–376.

Gomes, C.P., B. Selman, H. Kautz. 1998. Boosting combinatorial search through randomization. *Proc. AAAI*. Madison/WI, USA, 431–437.

Grönkvist, M. 2004. A constraint programming model for tail assignment. *Proc. Integration of AI and OR Techniques in CP for Combin. Optim.*, *LNCS*, vol. 3011. Springer, 142–156.

Grönkvist, M. 2006. Accelerating column generation for aircraft scheduling using constraint propagation. *Comp. & OR* **33** 2918–2934.

Gualandi, S. 2008. Enhancing constraint programming-based column generation for integer programs. Ph.D. thesis, Politecnico di Milano.

Gualandi, S., F. Malucelli. 2009. Constraint programming-based column generation. *4OR: Quart. J. Oper. Res.* **7** 113–137.

Gualandi, S., F. Malucelli. 2010. A branching scheme for vertex coloring problems. Tech. Rep. Print ID: 2009-11-2450, Optimization on-line. Submitted to *Discr. Appl. Math.*

Gvozdenović, N., M. Laurent. 2008. Computing semidefinite programming lower bounds for the (fractional) chromatic number via block-diagonalization. *SIAM J. Optim.* **19** 592.

Hansen, J., T. Liden. 2005. Group construction for airline cabin crew: Comparing constraint programming with branch and price. *Proc. Integration of AI and OR Techniques in CP for Combin. Optim.*, *LNCS*, vol. 3524. Springer, 228–242.

Hansen, P., M. Labbé, D. Schindl. 2009. Set covering and packing formulations of graph coloring: algorithms and first polyhedral results. *Disc. Optim.* **6** 135–147.

Harvey, W.D., M.L. Ginsberg. 1995. Limited discrepancy search. *Proc. Internat. Joint Conf. Artificial Intelligence*. 607–615.

Hertz, A., D. de Werra. 1987. Using tabu search techniques for graph coloring. *Computing* **39** 345–351.

Junker, U., S.E. Karisch, N. Kohl, B. Vaaben, T. Fahle, M. Sellmann. 1999. A framework for constraint programming based column generation. *Proc. Principles and Practice of Constraint Programming*, *LNCS*, vol. 1713. Springer, 261–274.

Leighton, F. T. 1979. A graph coloring algorithm for large scheduling problems. *J. Res. National Bureau of Standards* **84** 489–506.

Lübbecke, M.E., J. Desrosiers. 2005. Selected topics in column generation. *Oper. Res.* **53** 1007–1023.

Malaguti, E., P. Toth. 2010. A survey on vertex coloring problems. *Internat. Trans. Oper. Res.* **17** 1–34.

Mehrotra, A., M.A. Trick. 1996. A column generation approach for graph coloring. *INFORMS J. Comput.* **8** 344–354.

Mehrotra, A., M.A. Trick. 2007. A branch-and-price approach for graph multi-coloring. *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies.* 15–29.

Méndez-Díaz, I., P. Zabala. 2006. A branch-and-cut algorithm for graph coloring. *Disc. Appl. Math.* **154** 826–847.

Méndez-Díaz, I., P. Zabala. 2008. A cutting plane algorithm for graph coloring. *Disc. Appl. Math.* **156** 159–179.

Milano, M., M. Wallace. 2006. Integrating operations research in constraint programming. *4OR: Quart. J. Oper. Res.* **4** 1–45.

Ostergard, P.R.J. 2002. A fast algorithm for the maximum clique problem. *Disc. Appl. Math.* 197–207.

Otten, L., M. Grönkvist, D.P. Dubhashi. 2006. Randomization in constraint programming for airline planning. *Proc. Principles and Practice of Constraint Programming*, *LNCS*, vol. 4204. Springer, 406–420.

Pisinger, D., M. Sigurd. 2007. Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. *INFORMS J. Comput.* **19** 36–51.

Prestwich, Steven. 2008. Generalised graph colouring by a hybrid of local search and constraint programming. *Disc. Appl. Math.* **156** 148–158.

Régin, J.C. 1994. A filtering algorithm for constraints of difference in CSPs. *Proc. National Conf. Artificial Intelligence*. 362–362.

Rossi, F., P. Van Beek, T. Walsh. 2006. *Handbook of Constraint Programming*. Elsevier Science.

Rousseau, L.M. 2004. Stabilization issues for constraint programming based column generation. *Proc. Integration of AI and OR Techniques in CP for Combin. Optim.*, *LNCS*, vol. 3011. Springer, 402–408.

Rousseau, L.M., M. Gendreau, G. Pesant, F. Focacci. 2004. Solving VRPTWs with constraint programming based column generation. *Ann. Oper. Res.* **130** 199–216.

Ryan, D.M., B.A. Foster. 1981. An integer programming approach to scheduling. *Computer scheduling of public transport urban passenger vehicle and crew scheduling* 269–280.

Sadykov, R., L.A. Wolsey. 2006. Integer programming and constraint programming in solving a multimachine assignment scheduling problem with deadlines and release dates. *INFORMS J. Comput.* **18** 209–217.

Schrjiver, A. 2008. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer.

Schulte, C., M. Lagerkvist, G. Tack. 2009. *Modeling with Gecode*.

Sellmann, M., K. Zervoudakis, P. Stamatopoulos, T. Fahle. 2002. Crew assignment via constraint programming: Integrating column generation and heuristic tree search. *Ann. Oper. Res.* **115** 207–225.

Vanderbeck, François. 2009. Branching in Branch-and-Price: a Generic Scheme. Tech. Rep. 00311274, INRIA.

Vasquez, M. 2004. New results on the queens_n2 graph coloring problem. *J. Heuristics* **10** 407–413.

Yunes, T.H., A.V. Moura, C.C. de Souza. 2000. Solving very large crew scheduling problems to optimality. *Proc. ACM Symp. Appl. Comput.*. ACM New York, NY, USA, 446–451.

Yunes, T.H., A.V. Moura, C.C. de Souza. 2005. Hybrid column generation approaches for urban transit crew management problems. *Transp. Sci.* **39** 273–288.

Zykov, A. A. 1949. On some properties of linear complexes. *Mat. Sb.* **24(66)** 163–188.