Graph Coloring via Constraint Programming-based Column Generation

Stefano Gualandi Federico Malucelli

Dipartimento di Elettronica e Informatica, Politecnico di Milano
Viale Ponzio 24/A, 20133, Milan, Italy

Abstract

This paper presents a Constraint Programming-based Column Generation approach to the Minimum Graph Coloring problem. The proposed approach has two versions of the pricing subproblems: the first version formulates the pricing subproblem as the optimization problem of finding the most negative reduced cost column, the second version as the decision problem of finding a maximal stable set with negative reduced cost smaller than a given threshold. The value of the thresholds is not fixed a priori, but is computed during the execution of the column generation. Additionally, we introduce a so-called Constraint Programming augmented pricing with the aim of generating integer feasible solutions during the execution of the column generation algorithm. The proposed CP-based Column Generation results in a very efficient exact method for the minimum graph coloring coloring, being able to solve many hard instances of the DIMACS coloring benchmarks.

Keywords: *Graph Coloring, Constraint Programming, Column Generation*

1 Introduction

We consider the classical Minimum Graph Coloring Problem (Min-GCP). Given a graph $G = (V, E)$ and an integer k, a k-coloring of the graph G is a mapping of each vertex to a color, such that adjacent vertices take different colors. Min-GCP consists in finding the minimum k such that a k-coloring exists. The minimum k is known as the chromatic number of the graph and is usually denoted by $\chi(G)$. Min-GCP is NP-hard. Constraint Programming (CP) is a successful method for checking if a k-coloring exists, since constraint propagation can be exploited quite effectively [1]. However, standard CP approaches lack of efficient mechanisms that compute tight lower bounds, and guide the search towards the optimal region. For this reason, hybrid methods that integrate CP with mathematical programming techniques have been recently investigated [12]. A promising hybrid approach is the so–called Constraint Programming-based Column Generation (CP–CG) that formulates and solves the pricing subproblem as a CP problem. The CP–CG framework has been introduced in [9] for crew management problems. For a recent survey on CP–CG, see [7].

A formulation based on Column Generation of Min-GCP is introduced in [10], where the master subproblem is a set partitioning problem and the pricing subproblem is a maximum weighted stable set problem. We propose a CP–CG approach where the pricing subproblem has two formulations: the first one as the optimization problem of finding the most negative reduced cost column, the second one as the decision problem of finding a maximal stable set with negative reduced cost smaller than a threshold τ. During the column generation algorithm, we alternatively use the two versions. Additionally, 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 twofolds: first, integer feasible solutions give columns that can be part of the optimal integer solution, and second, since

in many cases the solution of the column generation algorithm is by itself time-consuming, it is desirable to improve both the upper bound and the lower bound at the same time.

In addition to the branch-and-price algorithm presented in [10], other remarkable exact approaches to Min-GCP are the DSATUR algorithm [2] and the branch-and-cut algorithm [11]. DSATUR is a sequential vertex coloring algorithm, that is, it 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 to which the vertex is adjacent. The branch-and-cut algorithm is based on the polyhedral study of the graph coloring polytope presented in [3], and it implements several families of violated inequalities, such as, for instance, the clique and the neighborhood inequalities, which are facet defining. An attempt to implement a branch-and-price-and-cut approach for Min-GCP is reported in [8], where a family of valid inequalities that do not break the structure of the pricing subproblem is introduced.

The outline of this paper is as follows. Section 2 presents the master and pricing subproblems used in a branch-and-price algorithm. Section 3 introduces our CP–CG approach to Min-GCP. Section 4 discusses the computation of integer solutions, and Section 5 reports on the computational results.

2 Formulation

The fastest exact approach to Min-GCP is presented in [10] and is based on a branch-and-price algorithm that computes at each node of the search tree a lower bound by solving via column generation the linear relaxation of the master problem. The Min-GCP problem can be formulated as the problem of partitioning the vertices of a graph into the minimum number of stable sets, since every vertex in an stable set can take the same color, no matter which color. Since Min-GCP is a minimization problem, it is formulated in [10] as a set covering of the vertices by inclusion-wise maximal stable sets. A set packing formulation of Min-GCP is presented in [8], but since it yields the same lower bound than the set covering formulation, we consider here only the later.

Let $\mathcal{S}^m$ be the collection of all the maximal stable sets in a given graph $G = (V, E)$. The master and the pricing subproblems for the column generation formulation of Min-GCP are defined as follows:

$$z_{MP} = \min \sum_{S \in \mathcal{S}^m} \lambda_S \qquad (1)$$

$$\text{s.t.} \sum_{S \in \mathcal{S}^m : i \in S} \lambda_S \geq 1, \qquad \forall i \in V, \qquad (2)$$

$$\lambda_s \geq 0, \qquad \forall S \in \mathcal{S}^m. \qquad (3)$$

$$c^* = \max \sum_{i \in V} \bar{\pi}_i x_i \qquad (4)$$

$$\text{s.t.} \quad x_i + x_j \leq 1, \qquad \forall \{i, j\} \in E, \qquad (5)$$

$$x_i \in \{0, 1\}, \qquad \forall i \in V. \qquad (6)$$

The master subproblem (1)–(3) corresponds to the linear relaxation of the set covering formulation, and the pricing subproblem (4)–(6) to a maximum weighted stable set problem. Note that the optimal value of (1)–(3) is equal to the fractional chromatic number of G and is denoted by $\chi_f(G)$. The coefficients $\bar{\pi}_i$ in the objective function (4) are the optimal dual variables of the covering constraint (2). Note that the objective function (4) should be $\min 1 - \sum_{i \in V} \bar{\pi}_i x_i$; therefore, negative reduced cost columns correspond to solutions of problem (4)–(6) with cost greater than 1.0. Let z_{RMP} denote the value of the restricted master problem. When the pricing subproblem is solved by an exact method, it is possible to compute a lower bound of z_{MP} with the formula $\frac{z_{RMP}}{1-c^*}$. The pricing subproblem (4)–(6) is solved in [10] by a recursive algorithm that could be used both as an exact and a heuristic method. When used as a heuristic method, it is stopped once a stable set of cost greater than 1.1 is found.

3 CP–based Column Generation for Min-GCP

In this section, we formulate the pricing subproblem with Constraint Programming, adapting the CP–CG framework to Min-GCP. The pricing subproblem is formulated as both a decision problem and an optimization problem. The pricing decision problem consists in finding a negative reduce cost column, while the pricing optimization problem consists in finding the lowest reduced cost column. During the

$\tau_1(i) = \frac{z^i_{RMP}}{\underline{\kappa}}$	$\tau_2(i,j) = \frac{z^j_{RMP}}{z^i_{RMP}}(1 - c^{*\,i})$	$\tau_3(i,j,l) = \frac{z^l_{RMP}}{\phi_1\left(\frac{z^j_{RMP}}{1-c^{*\,j}} - \phi_2 \frac{z^i_{RMP}}{1-c^{*\,i}}\right)}$

Table 1: Formulae for computing the values of the threshold τ.

execution of the column generation algorithm we alternate between the two versions in order to take advantage of their strengths: the decision pricing problem is much faster to solve than the optimization problem, because it stops the execution once a negative reduced cost column is found. The optimization pricing problem is solved because it permits to compute lower bounds on the optimal linear relaxations, which allows for an early termination of the column generation algorithm.

Let Y be a finite domain set variable that has the domain ranging into the set of vertices V, and let τ be a parameter equal to or smaller than zero. Let `negativeReducedCost` be a symbolic constraint equivalent to the linear constraint:

$$\texttt{negativeReducedCost}(\overline{\pi}, \texttt{Y}, \tau, \texttt{c}) \;\leftrightarrow\; \texttt{c} = \sum_{i \in Y} \bar{\pi}_i > \tau.$$

Fixing the value of τ to 1.0 is equivalent to look for any negative reduced cost column. In [10], the value of τ is implicitly set to 1.1.

The CP model of the pricing subproblem is as follows:

$$\begin{aligned} \textit{variables:} \quad & Y \subseteq V, && (7)\\ \textit{constraints}: \quad & \texttt{stableset}(G, Y), && (8)\\ & \texttt{negativeReducecCost}(\bar{\pi}, Y, \tau, c). && (9) \end{aligned}$$

The `stableset` symbolic constraint forces the set of vertices contained in the set variable Y to be a stable set of the graph G. The stable set constraint has been introduced in [4] as a symbolic `clique` constraint (recall that a clique is a stable set in the complementary graph). The decision pricing problem is the problem of finding an assignment for the set variable Y such that the constraints (8) and (9) are satisfied. The optimization pricing problem performs a CP branch-and-bound search, by branching the constraint $c > \bar{c}$, each time a feasible assignment $\overline{Y}$ having cost $\bar{c}$ is found by the CP solver.

In our approach, the value of the parameter τ is not fixed, but is recomputed each time a new lower bound of the master is obtained, as a function of the lower bound itself. The intuition is to look for a column with "sufficiently" negative reduced cost by exploiting the effect of constraint propagation inherent in Constraint Programming. Since the lower bound on z_{RMP} is not monotone, we would like to have a new column that yields a lower bound at least equal to the current lower bound. That is, we fix τ to a value that shall yield the current lower bound. Exploiting this idea, three formulae for computing τ are derived. Let i,j, and l be three subsequent iterations of the column generation algorithm, let $\underline{\kappa}$ be a given lower bound on $\overline{z}_{MP}$, and let z^i_{RMP} and c^{*i} be the master and pricing solutions at the i-th iteration. Table 3 gives the formulae used to compute the values of τ. The formula τ_1 depends only on the current i-th iteration, and it uses the value of the restricted master problem z^i_{RMP} and the given lower bound $\underline{\kappa}$. The formula τ_2 is used at the j-th iteration and exploits the lower bound computed at a previous iteration i. The formula τ_3, used at the l-th iteration, predicts and exploits the value of the current lower bound using two lower bounds computed at the previous iterations i and j. The details on how these formulae are derived are given in [6].

4 Computing Integer Solutions

Once the CP–CG algorithm has solved the linear relaxation, we need to find an integer solution to Min-GCP. The exact integer solution is computed in [10] by a branch-and-price algorithm that works very well when the lower bound is tight. Otherwise, it has to explore a large search tree before it can terminate.

As an alternative, it is possible to get heuristic solutions by solving the integer program with the columns generated by the CP–CG algorithm. Note that in both cases, the integer upper bound is not improved until the column generation algorithm is terminated, and both methods can be very time consuming.

We introduce here a so–called *augmented pricing* subproblem, to be solved just after the regular pricing subproblem, with the aim of generating integer feasible solutions during the execution of the column generation algorithm. The motivations for having an *augmented pricing* subproblem are twofold: first, integer feasible solutions give columns that can be part of the optimal solution, and second, it is desirable to improve both the upper bound and the lower bound at the same time. For Min-GCP the *augmented pricing* is formulated as the CP problem of finding a k-coloring, where $(k+1)$ is the current upper bound, that has the solution of the regular pricing subproblem as a given class of colors.

Given the graph $G = (V, E)$, let $\mathcal{C}$ be a collection of cliques that cover the edges E. Let $\overline{Y}$ be the solution of the pricing subproblem (7)–(9). The CP model of the *augmented pricing* subproblem is:

$$\begin{aligned}
&\textit{variables:} && Z_i \subseteq V, && \forall i \in \{1,\ldots,k\}, \\
&\textit{constraints:} && \texttt{stableset}(G, Z_i), && \forall i \in \{1,\ldots,k\}, \\
& && \texttt{partition}([Z_1,\ldots,Z_k], V), && (10) \\
& && Z_1 \equiv \bar{Y}. && (11)
\end{aligned}$$

Each finite domain set variables Z_i has the domain ranging in V. The `stableset` constraints force each set variable Z_i to be a stable set. The partitioning constraint (10) constrains the set variables Z_i to be a partition of the vertices V, that is $V \equiv \bigcup_{i\in\{1,\ldots,k\}} Z_i$ and $Z_i \cap Z_j = \emptyset, \forall i, j \in \{1,\ldots,k\}, i \neq j$. The constraint (11) assigns to the first set variable Z_1 the solution of the pricing subproblem $\overline{Y}$.

The constraint (11) plays an important role, since by fixing the set variables Z_1, the CP solver propagates the effects of such assignment and reduces the domain of the other set variables. If the augmented pricing subproblem finds a solution, then k is a new upper bound, and the Z_i variables, representing an integer solution, can be added as new columns to the restricted master problem. If the augmented pricing subproblem does not admit a solution, is not longer solved at the subsequent iterations of the column generation algorithms.

5 Computational results

The CP–CG algorithm with the adaptive thresholds and the augmented pricing subproblem has been implemented in C++ using the CPLEX solver, the GECODE constraint system [5], and the Boost Graph Library. The problem instances are from the DIMACS challenge on graph coloring, and are available at `http://mat.gsia.cmu.edu/COLOR/instances.html`. The tests are executed on an Intel dual core P4-1.8Ghz (but using a single core), that gives on the `dfmax` benchmarks the following results: `r200.5`=0.06, `r300.5`=0.49, `r400.5`=2.98, `r500.5`=11.3.

The first objective of the computational experiments is to asses the efficiency of the adaptive thresholds τ_1, τ_2, and τ_3, with respect to a fixed value of $\tau = 1.0$, as done in standard CP–CG, or $\tau = 1.1$, as done in [10]. Table 2 shows the results for the solution of the linear relaxation by our CP–CG algorithm. The first four columns of the table describe each problem instance, giving the name, the number of vertices and edges, and the ceiling of the linear master problem. The remaining columns report the computational time in seconds obtained by using different values of τ. These results show that using a threshold strictly greater than 1.0 decreases the computational times. The thresholds τ_2 and τ_3 have similar computation time, but since τ_2 is simpler, it should be preferred.

Table 3 reports the second set of experiments that compare two heuristics for finding integer solutions: an IP branch-and-bound with the columns generated by the CP–CG algorithm, and the CP–CG algorithm enhanced with the *augmented pricing* subproblem. The first six columns of Table 3 describe the problem instances, giving the graph size and density, the fractional chromatic number, and the best-known upper bound. The three columns of the IP branch-and-bound heuristic report the lower and upper bounds (LB-UB), the computation time t_{RMP} to solve the restricted master problem, and the computation time

					[MT96]	Adaptive Thresholds		
Problem	$\|N\|$	$\|E\|$	$\lceil\chi_f\rceil$	$\tau = 1.0$	$\tau = 1.1$	τ_1	τ_2	τ_3
queen8_8	64	1456	9	5.6	1.8	1.4	0.5	0.5
queen9_9	91	2112	9	49.8	7.2	9.1	3.5	3.5
queen10_10	100	1470	10	135	23.1	32	11.7	12.1
myciel6	95	755	4	2.3	0.6	0.6	0.2	0.2
myciel7	191	2360	5	42.2	9.4	13.5	9.2	7.7
DSJC125.1	125	736	5	729	112	112	35	132
DSJC125.5	125	3891	**16**	49	15.3	9	7.3	8.5
DSJC125.9	125	6961	**43**	1.5	1.1	0.9	0.8	0.8
DSJC250.5	250	15668	**26**	1192	468	301	260	253
DSJC250.9	250	27897	**71**	56	26	17.4	17	19
DSJC500.9	500	22874	**123**	1210	609	266	257	258

Table 2: Time in seconds for solving the linear master problem. The numbers in bold on the fourth column denote new best lower bounds.

						IP-Heuristic			Augmented Pricing		
Problem	$\|N\|$	$\|E\|$	d	$\lceil\chi_f\rceil$	χ	LB-UB	t_{RMP}	t_{IP-H}	LB-UB	t_{UB}	t_{RMP}
queen8_8	64	1456	0.7	9	9	9-10	0.6	1.2	**9**	24.9	25.4
queen9_9	91	2112	0.5	9	10	9-11	3.9	16	9-10	29	32.6
queen10_10	100	1470	0.3	10	10	10-12	14	463	10-12	0.53	3600
will199GPIA	701	6772	0.1	7	7	7-8	1314	1336	**7**	3.8	448
le450_5d	450	9757	0.1	5	5	5-10	3600	-	5-6	20	3600
le450_15c	450	16680	0.2	15	15	15-24	3600	-	15-22	1191	3600
le450_15d	450	16750	0.2	15	15	15-25	3600	-	15-23	4.3	3600
DSJC125.1	125	736	0.1	5	5	5-6	132	136	5-6	0.5	1202
DSJC250.1	250	3218	0.1	6	?	6-10	3600	-	6-9	1.42	3600
DSJC125.5	125	3891	0.5	16	?	16-19	8.5	1274	16-20	0.69	3600
DSJC250.5	250	15668	0.5	26	?	26-43	253	3600	26-36	3.66	3600
DSJC125.9	125	6961	0.9	43	44	43-44	0.8	1.6	43-49	0.22	3600
DSJC250.9	250	27897	0.9	71	73	71-73	19	178	71-87	2.67	3600

Table 3: IP-heuristic vs. augmented pricing. Time in seconds. Question marks denote open problems.

t_{UB} to compute the integer solution, *i.e.*, the upper bound UB. The three columns of the CP–CG with the *augmented pricing* report the lower and upper bounds (LB-UB), the computation time t_{UB} for finding the integer solution with the augmented pricing, and the computation time t_{RMP} to solve the restricted master problem. Note that for the augmented pricing the computation time of the restricted master problem t_{RMP} is higher than the IP heuristic, since the integer solutions are computed by the augmented pricing. In several instances, the augmented pricing finds better integer solutions taking less time. For two instances, queen8_8 and will199GPIA, the augmented pricing is clearly better, since it closes the LB-UB gap. For the Leighton graphs le450_5d, le450_15c, and le450_15d, the augmented pricing improves the initial integer solutions, even if, as the IP heuristic, is not able to solve the restricted master problem. For the random graphs due to David Johnson, the DSJCn instances, the IP heuristic is faster and computes better solutions for dense graphs, while the augmented pricing is faster in computing integer solutions for sparse graphs.

Table 4 compares the CP–CG algorithm with the augmented pricing subproblem with the branch-and-cut [11] and the DSATUR [2] algorithms. For each method, the table reports the lower and upper bounds. DSATUR computes quickly solutions of good quality, but it lacks of a way of improving the lower bound. The branch-and-cut algorithm, which uses DSATUR as preprocessing, is able to improve the lower bounds, but without getting as good lower bound as our approach does.

					Branch&Cut [11]		DSATUR [2]		Aug.Pricing	
Problem	$\|N\|$	$\|E\|$	$\lceil\chi_f\rceil$	χ	LB	UB	LB	UB	LB	UB
queen9_9	91	2112	9	10	9	11	9	10	9	10
will199GPIA	701	6772	7	7	6	7	7	7	7	7
le450_5d	450	9757	5	5	5	10	5	8	5	6
le450_15c	450	16680	15	15	15	24	15	23	15	22
le450_15d	450	16750	15	15	15	23	15	23	15	23
DSJC125.5	125	3891	16	?	13	20	9	19	16	20
DSJC125.9	125	6961	43	44	42	47	29	45	43	49
DSJC250.5	250	15668	26	?	13	36	9	35	26	36
DSJC250.9	250	27897	71	73	48	88	34	87	71	87

Table 4: Comparing lower and upper bounds of different algorithms.

As side effect of our experiments, we have improved over four lower bounds found in literature, namely for the DSJC125.5, DSJC250.5, DSJC500.9, and DSJC1000.9. Implementing a branch-and-price algorithm as in [10], but using our implementation of the column generation algorithm with adaptive thresholds, we are able to prove optimality for two open instances, namely DSJC125.9 and DSJC250.9.

References

[1] N. Barnier and P. Brisset. Graph coloring for air traffic flow management. *Annals of Operation Research*, 130(1):163–178, 2004.

[2] D. Brélaz. New methods to color the vertices of a graph. *Commun. ACM*, 22(4):251–256, 1979.

[3] P. Coll, J. Marenco, I. Mendez-Diaz, and P. Zabala. Facets of the graph coloring polytope. *Annals of Operations Research*, 116:79–90(12), 2002.

[4] T. Fahle. Simple and fast: Improving a branch-and-bound algorithm for maximum clique. In *Algorithms - ESA 2002: 10th Annual European Symposium*, pages 47–86. Springer, 2002.

[5] Gecode. Generic Constraint Development Environment, 2006. `http://www.gecode.org`.

[6] S. Gualandi. *Enhancing Constraint Programming-based Column Generation for Integer Programs.* PhD thesis, Politecnico di Milano, 2008.

[7] S. Gualandi and F. Malucelli. Constraint programming-based column generation: a survey. Submitted, 2008.

[8] P. Hansen, M. Labbé, and D. Schindl. Set covering and packing formulations of graph coloring: algorithms and first polyhedral results. Technical Report 552, ULB, 2005.

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

[10] A. Mehrotra and M.A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–354, 1996.

[11] I. Méndez-Díaz and P. Zabala. A branch-and-cut algorithm for graph coloring. *Discrete Appl. Math.*, 154(5):826–847, 2006.

[12] M. Milano and M. Wallace. Integrating operations research in constraint programming. *4OR*, 4(3):1–45, 2006.