An exact algorithm for the Partition Coloring Problem

Fabio Furini, Enrico Malaguti, Alberto Santini

An Exact Algorithm for the Partition Coloring Problem

Fabio Furini

Université Paris Dauphine, PSL Research University, CNRS, 75016 Paris, France
fabio.furini@dauphine.fr

Enrico Malaguti[1]

DEI, University of Bologna, 40136 Bologna, Italy
enrico.malaguti@unibo.it

Alberto Santini

Department of Economics and Business, Pompeu Fabra University, 08005 Barcelona, Spain
alberto.santini@upf.edu

Abstract

We study the Partition Coloring Problem (PCP), a generalization of the Vertex Coloring Problem where the vertex set is partitioned. The PCP asks to select one vertex for each subset of the partition in such a way that the chromatic number of the induced graph is minimum. We propose a new Integer Linear Programming formulation with an exponential number of variables. To solve this formulation to optimality, we design an effective Branch-and-Price algorithm. Good quality initial solutions are computed via a new metaheuristic algorithm based on adaptive large neighbourhood search. Extensive computational experiments on a benchmark test of instances from the literature show that our Branch-and-Price algorithm, combined with the new metaheuristic algorithm, is able to solve for the first time to proven optimality several open instances, and compares favourably with the current state-of-the-art exact algorithm.

Keywords: Vertex Coloring, Partitioning Coloring, Selective Coloring, Column Generation, Branch-and-Price Algorithm.

1. Introduction

Graph coloring problems are among the most studied ones in both graph theory and combinatorial optimization. Given an undirected graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges, the classical *Vertex Coloring Problem* (VCP) consists of assigning a color to

[1]Corresponding author

each vertex of the graph in such a way that two adjacent vertices do not share the same color and the total number of colors is minimized. The *chromatic number* of G, denoted by $\chi(G)$, is the minimum number of colors in a coloring of G.

The VCP is an $\mathcal{NP}$-hard problem and it has a variety of applications, among which: scheduling, register allocation, seating plan design, timetabling, frequency assignment, sport league design, and many others (we refer the interested readers to Pardalos et al. [25], Marx [22], Lewis [17]). The VCP and its variants are very challenging from a computational viewpoint; the best performing exact algorithms are usually based on exponential-size Set Covering formulations, and require Branch-and-Price techniques to be solved (see, e.g., Malaguti et al. [21], Gualandi and Malucelli [14], Held et al. [15], Furini and Malaguti [13]). For dense graphs, good results are obtained by advanced Integer Linear Programming (ILP) compact formulations, like the so-called *representatives formulation* (see Campêlo et al. [6], Cornaz et al. [7]), which are able to remove the symmetry affecting classical descriptive compact ILP models.

In this manuscript we study the *Partition Coloring Problem* (PCP), a generalisation of the VCP where the vertex set is partitioned and exactly one vertex of each subset of the partition has to be colored. The PCP asks to select one vertex for each subset of the partition in such a way that the chromatic number of the induced graph is minimum. The PCP is $\mathcal{NP}$-hard since it generalizes the VCP and it is also known in the literature as the *Selective Graph Coloring Problem*.

Formally, let $\mathscr{P} = \{P_1, \ldots, P_k\}$ be a k-partition of the vertex set V of G. A *stable set* is a subset $S \subseteq V$ of non-adjacent vertices, i.e., $\forall u, v \in S, uv \notin E$. A *partial coloring* $\tilde{C}$ of G is a partition of a subset of vertices $\tilde{V} \subset V$ into h non-empty stable sets or colors ($\tilde{C} = \{\tilde{V}_1, \ldots, \tilde{V}_h\}$), while the remaining vertices $V \setminus \tilde{V}$ are uncolored. Let $f(v)$ be a function which returns the color of a colored vertex v ($v \in \tilde{V}$). The PCP consists of finding a partial coloring $\tilde{C}$ such that:

(i) $|\tilde{V} \cap P_i| = 1$ for $i = 1, 2, \ldots, k$;

(ii) $f(v) \neq f(w)$ for all $v, w \in \tilde{V}$, $vw \in E$;

(iii) h is minimum.

The minimum number of colors used in any optimal PCP solution is denoted in this manuscript as *Partition Chromatic Number* $\chi_P(G, \mathscr{P})$.

Let us introduce an example, called *Example 1*. In the left part of Figure 1, we depict a graph G of ten vertices and thirteen edges. The graph is partitioned in five subsets ($k = 5$), each subset is composed of two vertices; the dotted lines are used to identify the subsets of the partition. In the right part of Figure 1, we depict a feasible partial coloring $\tilde{C}$ using two colors (gray and black). For each subset of the partition exactly one vertex is colored. The colored vertices, i.e., the vertices $v \in \tilde{V}$, are colored with the corresponding color (gray or black) while the uncolored ones are white.

The PCP models many real-world applications (see Demange et al. [9]) including: routing and wavelength assignment, dichotomy-based constraint encoding, antenna positioning and frequency assignment, as well as a wide variety of scheduling problems (timetabling, quality

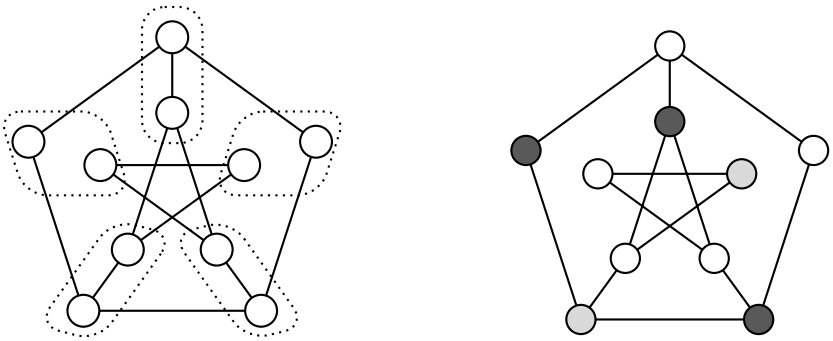

Figure 1: Example: (left) a graph G and a partition of its vertices in 5 subsets ($k = 5$); (right) a feasible partition coloring of G with two colors (gray and black).

test, berth allocation) and a variant of the classical Travelling Salesman Problem.

1.1. Literature review

The PCP was introduced by Li and Simha [18] to model wavelength routing and assignment problems. Three heuristic algorithms for the VCP, i.e., the *Largest-First*, the *Smallest-Last* and the *Color-Degree* have been adapted to tackle the PCP. In Li and Simha [18], a first set of benchmark instances for the PCP has been described, representing mesh optical networks and the National Science Foundation Net (called `nsf` in the following).

Noronha and Ribeiro [24] presented a heuristic algorithm for the PCP, again in the setting of wavelength assignment. The authors obtain initial solutions by using an adaptation of *Color-Degree* algorithm proposed by Li and Simha [18], and then try to decrease the number of colors via Tabu Search (TS). They show how TS outperforms the previous constructive heuristics, and investigate the relationship between algorithm performance and graph density. This algorithm was tested on large random instances, with up to 1800 vertices. A memetic heuristic algorithm was designed by Pop et al. [26], which combines genetic operators with a local search phase.

Theorethical results on the complexity of the PCP on particular classes of graphs have been obtained in Bonomo et al. [4], Demange et al. [8] and Demange et al. [10].

To the best of our knowledge, only two works described exact algorithms for the PCP: Frota et al. [12] and Hoshino et al. [16]. The first one proposes a branch-and-cut algorithm based on the asymmetric representatives formulation introduced by Campêlo et al. [6, 5] for the VCP. A number of valid inequalities are introduced and used within a branch-and-cut framework. A Tabu Search heuristic algorithm has also been proposed to initialize the formulation. Computational tests are reported on randomly generated instances (called `random`), VCP instances from the literature, and instances derived from the routing and wavelength assignment literature (including the `nsf` instances, and a new set of instances called `ring`).

The second exact algorithm, i.e., the one presented in Hoshino et al. [16], is a branch-and-price algorithm based on the Dantzig-Wolfe reformulation of the representatives formulation. In

the latter, each vertex is the representative of one color, and the color can be used if and only if the associated vertex is assigned the color. In order to deal with an exponential number of variables, a column generation scheme has been proposed which is based on a set of pricing problems, one for each vertex. The authors show how to adapt the valid inequalities used by Frota et al. [12] to the reformulated model. However, since the inequalities did not prove to be computationally effective, they were not added to the model. Several heuristic algorithms have also been presented in Hoshino et al. [16]. Computational results on the `random`, `nsf`, and `ring` instances showed that the branch-and-price algorithm of Hoshino et al. [16] outperforms the branch-and-cut algorithm of Frota et al. [12].

1.2. Paper Contribution

In Section 2 we introduce a new formulation for the PCP with an exponential number of variables and in Section 3 we design a Brach-and-Price algorithm to solve it to proven optimality. Based on a study of the mathematical structures of the formulation, we managed to design a pricing phase based on a unique pricing problem. This is a main improvement with respect to the state-of-the-art branch-and-price algorithm of Hoshino et al. [16], which requires instead to solve several pricing problems, one for each "representative" vertex. To effectively initialize our branch-and-price algorithm, a new metaheuristic algorithm is presented in Section 4. Several instances of the considered test bed have been solved to proven optimality at the root node, i.e., no branching is required, thanks to the quality of the heuristic solutions and the strength of the lower bound provided by the linear programming relaxation of the new formulation. In Section 5 we present extensive computational experiments comparing the new exact algorithm with the state-of-the-art approach. Finally, in Section 6, we draw some conclusions.

2. Integer Linear Programming Formulations

In this section we first introduce a natural ILP formulation for the PCP and then we derive a new extended formulation based on the Dantzig-Wolfe reformulation of the natural formulation. A trivial upper bound on the number of colors used in any optimal PCP solution is given by the number k of subsets of the partition. We can then introduce a set of binary variables y with the following meaning:

$$y_c = \begin{cases} 1 & \text{if color } c \text{ is used} \\ 0 & \text{otherwise} \end{cases} \qquad c = 1, 2, \ldots, k;$$

and a set of binary variables x with the following meaning:

$$x_{vc} = \begin{cases} 1 & \text{if vertex } v \text{ is colored with color } c \\ 0 & \text{otherwise} \end{cases} \qquad v \in V,\ c = 1, 2, \ldots, k.$$

The first natural ILP formulation (called ILP^{N}) reads:

$$(\text{ILP}^{\text{N}}) \qquad \min \quad \sum_{c=1}^{k} y_c \tag{1}$$

$$\sum_{c=1}^{k} \sum_{v \in P_i} x_{vc} = 1 \qquad i = 1, 2, \ldots, k \tag{2}$$

$$x_{vc} + x_{uc} \leq y_c \qquad uv \in E,\ c = 1, 2, \ldots, k \tag{3}$$

$$x_{vc} \in \{0, 1\} \qquad v \in V,\ c = 1, 2, \ldots, k \tag{4}$$

$$y_c \in \{0, 1\} \qquad c = 1, 2, \ldots, k, \tag{5}$$

where the objective function (1) minimizes the number of used colors, constraints (2) impose that one vertex per subset of the partition is colored, and constraints (3) impose that adjacent vertices do not receive the same color. Finally, constraints (4) and (5) define the variables of the formulation.

By replacing constraints (4) and (5) with

$$x_{vc} \geq 0 \qquad v \in V,\ c = 1, 2, \ldots, k \tag{6}$$

$$y_c \geq 0 \qquad c = 1, 2, \ldots, k, \tag{7}$$

we obtain the Linear Programming relaxation of ILP^{N}, that will be denoted as LP^{N} in what follows.

Descriptive natural models for coloring problems are known to produce weak linear programming relaxations and are affected by symmetry (see Malaguti and Toth [20], Cornaz et al. [7]), hence, in general they can be solved to optimality only for small graphs. In order to improve the strength of the linear programming relaxation, and to remove the symmetry of model (1)–(5), we convexify constraints (3) through Dantzig-Wolfe reformulation (see [11],[2] and [3]). Let us introduce the following exponential-size collection $\mathscr{S}$ of stable sets of G which intersect each subset of the partition at most once:

$$\mathscr{S} = \{S \subseteq V \ : \ uv \notin E,\ \forall u, v \in S\ ;\ |S \cap P_i| \leq 1,\ i = 1, \ldots, k\}\,. \tag{8}$$

A valid model for the PCP can be obtained by introducing, for each subset $S \in \mathscr{S}$, a binary variable ξ_S with the following meaning:

$$\xi_S = \begin{cases} 1 & \text{if vertices in } S \text{ take the same color} \\ 0 & \text{otherwise} \end{cases} \qquad S \in \mathscr{S}$$

then the extended ILP formulation reads:

$$(\text{ILP}^{\text{E}}) \qquad \min \quad \sum_{S \in \mathscr{S}} \xi_S \tag{9}$$

$$\sum_{S \in \mathscr{S} : |S \cap P_i| = 1} \xi_S = 1 \qquad i = 1, \ldots, k \tag{10}$$

$$\xi_S \in \{0, 1\} \qquad S \in \mathscr{S}, \tag{11}$$

where the objective function (9) minimizes the number of stable sets (colors), whereas constraints (10) ensure that exactly one vertex of each subset of the partition is colored. Finally constraints (11) impose all variables be binary. Constraints (10) can be rewritten as:

$$\sum_{S\in\mathscr{S}:|S\cap P_i|=1} \xi_S \geq 1 \qquad i=1,\ldots,k, \tag{12}$$

since it is always possible to transform a solution of model (9), (12) and (11) into a solution of model (9)–(11) of same value. Constraint (12) ensures that the associated dual variables take non negative values. The resulting formulation (9)-(12)-(11) is denoted as ILP$^{\mathrm{E}}$ in the following sections.

Finally, by relaxing the integrality of constraints (11) to

$$\xi_S \geq 0 \qquad S\in\mathscr{S}, \tag{13}$$

we obtain the Linear Programming relaxation of ILP$^{\mathrm{E}}$, that is denoted as LP$^{\mathrm{E}}$ in what follows.

By observing that ILP$^{\mathrm{E}}$ is obtained by applying Dantzig-Wolfe reformulation of constraints (3) of ILP$^{\mathrm{N}}$ and since constraints (3) do not form a totally unimodular matrix, it follows that the quality of the lower bound obtained solving the LP relaxation of ILP$^{\mathrm{N}}$ is dominated by its counterpart associated with ILP$^{\mathrm{E}}$:

Proposition 1. *The optimal value of LP^E is greater than or equal to the optimal value of LP^N.*

Proof. Proving the proposition for the specific PCP models give more insight on the structure of the LP relaxation optimal solutions of ILP$^{\mathrm{N}}$ and ILP$^{\mathrm{E}}$. We first show that any feasible solution of LP$^{\mathrm{E}}$ can be converted into a feasible solution of LP$^{\mathrm{N}}$ with the same objective function value. Given a function $p(v)$ which returns the corresponding index i ($i=1,2,\ldots,k$) of the subset of the partition of a vertex v ($v\in V$), we can uniquely define the color $c(S)$ of any $S\in\mathscr{S}$ as $\min_{v\in S} p(v)$. Let ξ^* denote a feasible solution of LP$^{\mathrm{E}}$ and assume, without loss of generality, that no subset of the partition is covered by more than one selected subset $S\in\mathscr{S}$. Let us define a solution (x^*,y^*) as follows: for each color c set

$$y^*_c = \sum_{S\in\mathscr{S}\,:\,c=c(S)} \xi^*_S \quad \text{and} \quad x^*_{vc} = \sum_{\substack{S\in\mathscr{S}\,:\,c=c(S),\\ v\in S}} \xi^*_S. \tag{14}$$

Thus, inequalities (10) ensure that constraints (2) are satisfied. Observe that, by construction, for each edge $uv\in E$ and for each color $c=1,2,\ldots,k$ we have $x^*_{vc}+x^*_{uc}\leq y^*_c$; thus, (x^*,y^*) is feasible to LP$^{\mathrm{N}}$ and the objective function value remains unchanged.

We then show a case where the optimal value of LP$^{\mathrm{E}}$ is strictly larger than the optimal value of LP$^{\mathrm{N}}$. Consider the instance of Figure 2, where we depict a graph G of ten vertices and twenty edges. The graph is partitioned into five subsets ($k=5$), and each subset is composed of two vertices. The dotted lines define the subsets of the vertex partition. The figure report also a numbering of the vertices of the graph. Let consider the following solution of LP$^{\mathrm{E}}$:

$\xi^*_{S_1} = \xi^*_{S_2} = \xi^*_{S_3} = \xi^*_{S_4} = \xi^*_{S_5} = 0.5$ where the five stable sets are $S_1 = \{1, 8\}$, $S_2 = \{1, 9\}$, $S_3 = \{2, 9\}$, $S_4 = \{2, 10\}$, and $S_5 = \{8, 10\}$. This solution has value 2.5 and it is optimal. Since in G all stable sets intersecting each subset at most once have at most 2 vertices, then at least 2.5 of them are necessary to cover one vertex per subset of the partition. The optimal solution value of LP^{E} is larger than the optimal solution value of LP^{N}, which is 1, and it is obtained by setting $y_c = 0.2$ $(c = 1, \ldots, 5)$ and $x_{vc} = 0.1$ $(v \in V, c = 1, \ldots, 5)$. □

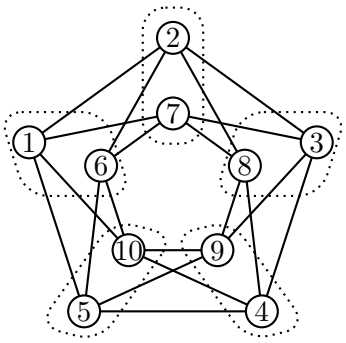

Figure 2: Example: a graph G of 10 vertices and a partition of its vertices in 5 subsets ($k = 5$).

Model $\mathrm{ILP}^{\mathrm{E}}$ has exponentially many ξ_S variables ($S \in \mathscr{S}$), which cannot be explicitly enumerated for large-size instances. *Column Generation* (CG) techniques are therefore necessary to efficiently solve $\mathrm{ILP}^{\mathrm{E}}$. In the following section we present a new Branch-and-Price algorithm for $\mathrm{ILP}^{\mathrm{E}}$, and refer the interested reader to [11] for further details on CG.

3. A New Branch-and-Price Algorithm

There are two main ingredients of a Branch-and-Price algorithm, i.e., a CG algorithm to solve the LP relaxation of the exponential-size integer model, and a branching scheme. We discuss separately these two aspects in the next sections.

3.1. Solving the Linear Programming relaxation of ILP^E

Model (9), (12) and (13), initialized with a subset of variables containing a feasible solution, is called the *Restricted Master Problem* (RMP). Additional new variables, needed to solve LP^{E} to optimality, can be obtained by separating the following dual constraints:

$$\sum_{\substack{i=1,2,\ldots,k\ :\\ |P_i \cap S|=1}} \pi_i \leq 1 \qquad S \in \mathscr{S}, \tag{15}$$

where π_i $(i = 1, 2, \ldots, k)$ is the dual variable associated with the i-th constraint (12). Accordingly, the CG performs a number of iterations, where violated dual constraints are added to the RMP in form of primal variables, and the RMP is re-optimized, until no violated dual constraint exists. At each iteration, the so-called *Pricing Problem* (PP) is solved. This

problem asks to determine (if any) a stable set $S^* \in \mathscr{S}$ for which the associated dual constraint (15) is violated, i.e., such that

$$\sum_{\substack{i=1,2,\ldots,k \ : \\ |P_i \cap S^*|=1}} \pi_i^* > 1, \tag{16}$$

where π^* is the optimal vector of dual variables of RMP at the current iteration of the CG procedure.

The pricing problem can be modeled as a *Maximum Weight Stable Set Problem* (MWSSP) on an *auxiliary graph* $\hat{G} = (V, \hat{E})$, constructed as follows: the vertex set of $\hat{G}$ coincides with the vertex set of G, while the edge set $\hat{E}$ is constructed from the edge set of G and its partition $\mathscr{P} = \{P_1, \ldots, P_k\}$:

$$\hat{E} = E \cup \{uv \ : \ u, v \in P_i, \ i = 1, \ldots, k\}. \tag{17}$$

In other words, each subset of the partition of G is transformed into a clique in $\hat{G}$. Given a weight vector $c \in \mathbb{R}_+^{|V|}$, where the weight c_v of the vertex $v \in P_i$ is set to the value π_i^* associated with the i-th subset of the partition, the pricing problem corresponds then to a MWSSP in $\hat{G}$, that is, to determine a stable set S of $\hat{G}$ maximizing $\sum_{v \in S} c_v$.

Notice that since each partition subset has been turned into a clique, such a stable set contains at most one vertex per subset P_i and therefore collects each profit π_i at most once. The MWSS can be solved by means of a specialized combinatorial Branch-and-Bound algorithm (see Section 5).

If a stable set S^* has total weight larger than one (that is, the reduced cost is negative), the associated column is added to the RMP and the problem is re-optimized. If, on the other hand, the total weight is not larger than 1, by linear programming optimality conditions no column can improve the objective function of the RMP and therefore $\mathrm{LP^E}$ is solved to proven optimality.

3.2. Branching scheme for ILP^E

The design of a branching scheme is crucial for the performance of a branch-and-price algorithm [30]. In the following we describe the branching scheme adopted in our new Branch-and-Price framework. Two are its main properties. Firstly, it is a complete scheme, i.e., it ensures that integrality can be imposed in all cases. Secondly, it does not require modifications neither on the master problem nor the pricing algorithm. The latter means that our branching does not alter the structure of the pricing problem so that the same algorithm can be applied during the entire search.

Consider a fractional solution ξ^* of $\mathrm{LP^E}$, at a given node of the branching tree, and let $\hat{\mathscr{S}} \subseteq \mathscr{S}$ be the set of columns in the RMP at the node. We propose a branching scheme composed of two rules applied in sequence, i.e., when the branching condition for the first rule fails, the second is applied.

The *first branching rule* is designed to impose that exactly one vertex is colored for each subset. Constraints (12) impose that the sum of the values of the variables associated with stables sets intersecting each subset is at least one, but in a fractional solution these stable

sets can include different vertices in the same subset of the partition. A given subset P_i has more than one (partially) colored vertex if:

$$|\{v \in P_i \ : \sum_{S \in \hat{\mathscr{S}}, v \in S} \xi_S^* > 0\}| > 1. \tag{18}$$

In case more than one of such subsets exists, we select the subset i with the largest number of (partially) colored vertices, breaking ties by size of the subsets (preferring smaller subsets and breaking further ties randomly). We then branch on the vertex $v \in P_i$ with the largest value of $\sum_{S \in \hat{\mathscr{S}}, v \in S} \xi_S^*$. Two children nodes are created:

- in the first node we impose that v is the colored vertex for subset P_i;
- in the second node, we forbid that v is the colored vertex for subset P_i.

This branching rule can be enforced without any additional constraint neither for the RMP nor for the pricing problem. To force the coloring of v in the children nodes of the branching scheme, we remove from the graph G all other vertices $u \in P_i$ ($u \neq v$); to forbid the coloring of v, we simply remove the vertex from the graph G. This first branching rule is not complete since it may happen that a vertex (partially) belongs to more that one stable set in the solution ξ^*.

If this happens for a vertex v, there must be another vertex u (belonging to a different subset of the partition) such that:

$$\sum_{S \in \hat{\mathscr{S}}: v, u \in S} \xi_S^* = \gamma, \ \gamma \text{ is fractional.} \tag{19}$$

We say that v and u are a *fractionally colored* pair of vertices.

The *second branching rule* is designed to impose that each pair of (colored) vertices either takes the same color, or the two vertices of the pair take different colors. This rule has been proposed for the VCP by Zykov [31] and used to derive several effective Branch-and-Price algorithms for the VCP, starting from the seminal work by Mehrotra and Trick [23], see, e.g., [21, 14, 15]. In case more pairs of fractionally colored vertices exist, we select the pair v and u with the largest γ value. Two children nodes are then created:

- in the first node we force vertices v and u to take the same color;
- in the second node we force vertices v and u to take different colors.

The second branching rule can also be enforced without any additional constraint neither for the RMP nor for the pricing problem. To force different colors for a pair of vertices v and u in the children nodes of the branching scheme, we add the edge vu to E. On the other hand, to force v and u to take the same color, we remove v and u from the graph G and replace them with a new vertex z; then we add edges zw for all $w \in V$ if $uw \in E$ or $vw \in E$ (or both). We then consider a stable set containing the vertex z coloring both $P_{p(v)}$ and $P_{p(u)}$, where the function $p(v)$ ($v \in V$) returns the index of the subset of the partition containing vertex v.

In our Branch-and-Price algorithm we first define the vertices for each subset of the partition to be colored, i.e., we apply the first branching rule. Then, in case the solutions are still fractional, we apply the second branching rule in order to obtain integer solutions.

After branching, the variables that are incompatible with the branching decision are removed from the children nodes. The following proposition states that the two proposed branching rules define a complete branching scheme for $\mathrm{ILP^E}$:

Proposition 2. *The two branching rules applied in sequence provide a complete branching scheme for model ILP^E.*

Proof. After the application of the first branching rule, the colored vertex in each subset of the partition is determined. In [1] it is proved that for any 0-1 constraint matrix A (as for the case of $\mathrm{LP^E}$), if a basic solution ξ^* to $A\xi = 1$ is fractional, then there exist two rows i and j such that:

$$0 < \sum_{S \in \mathscr{S} \,:\, i,j \in S} \xi^*_S < 1 \tag{20}$$

This result allows us to conclude that if a solution is fractional then we can determine two subsets of the partition such that (20) holds. The same holds for the case in which $A\xi^* > 1$: in any optimal fractional solution to $\mathrm{LP^E}$, the rows for which covering constraints are satisfied with equality must be covered by at least two columns with associated fractional variables, and the previous result applies. By picking the colored vertex from the first and the colored vertex from the second subset, the two vertices constitute a fractionally colored pair of vertices on which to apply the second branching rule. □

4. Initialization of the Branch-and-Price algorithm

In order to initialize the Branch-and-Price algorithm with a feasible solution of good quality in short computational time, we devised a metaheuristic algorithm based on the Adaptive Large Neighbourhood Search (ALNS, first introduced by Ropke and Pisinger [27]), improved by a Local Search phase. In the following paragraphs we describe the skeleton of the ALSN while, in Section 5, we report the details on how the algorithm has been run. In particular, we comment on how the ALSN parameters have been tuned in Section 5.2. Finally we describe some additional procedures which allow to improve the quality of the solutions computed by the metaheuristic algorithm in Section 5.4.

4.1. ALNS-based heuristic

The basic idea behind the Adaptive Large Neighbourhood Search (ALNS) is to explore the solution space using a large collection of neighbourhoods. At each iteration, the neighbourhood to explore is chosen randomly, with a probability proportional to a given score. The score, in turn, reflects the past performance of the neighbourhood during the solution process. Algorithm 1 shows the general framework of ALNS. In the following, it is assumed that when a tie is present, it is resolved randomly.

In line 1 and line 2 the current and best solutions are initialized. The initial solution is created by a simple greedy procedure that constructs stable sets one at the time. Starting from a stable set composed of a vertex from an uncolored subset, the procedure keeps adding the least connected vertices of uncolored subsets to the current stable set. When this is not possible anymore, it starts a new stable set. The iteration counter is initialized at line 3, and line 4 initializes the neighbourhood scores. The algorithm is run for a given number M of iterations. At each iteration, a neighbourhood is selected (line 6) using a roulette-wheel selection mechanism, with probabilities proportional to the scores. Since the neighbourhood size is often exponential, N is not explored completely, but just sampled, in order to produce a new solution x' (line 7). Next, in line 8, the new solution is evaluated and either accepted or rejected, according to an *acceptance criterion*. The acceptance criterion uses a set of parameters that can change during the solution process: for example, accepting worsening solutions might be more likely at the beginning of the process than at the end. The current (line 9) and best (line 12) solutions are possibly updated, and finally the scores (line 14), the acceptance criterion parameters (line 15) and the iteration counter (line 16) are updated, and the best known solution is returned on line 18.

In our algorithm, the set $\mathcal{N}$ of neighbourhoods is not explicitly enumerated. Rather, we give a set of destroy methods and a set of repair methods. The former transform a feasible solution into an unfeasible one, and the latter transform an unfeasible solution into a feasible one. Each combination of a destroy method followed by a repair method gives rise to a neighbourhood. Rather than keeping scores for the neighbourhoods, then, we keep the scores of the individual destroy and repair methods and perform two independent roulette-wheel selections. Notice that this approach can only work if (as in our case) all destroy and repair methods are compatible, meaning that it is possible to repair a destroyed solution produced by any destroy method, with any repair method.

In our implementation, we devised the destroy and repair methods described below. In order to compact the exposition, some similar methods have been grouped together and their distinctive elements are listed in curly braces.

- **Destroy methods**

  1. Select {a random, the smallest, the biggest} stable set of the solution, and remove a random vertex from that set. (Methods **D1**, **D2**, and **D3**.)

  2. Select the colored vertex with {smallest, largest} external degree, and remove it from the stable set it belongs to. (Methods **D4**, and **D5**.)

  3. Select the colored vertex with {smallest, largest} color degree, and remove it from the stable set it belongs to. The color degree of a vertex v is the number of vertices w such that $vw \in E$ and w is colored in the current solution. (Methods **D6**, and **D7**.)

  4. As in items 2 and 3 but the vertex to be removed is chosen with a roulette wheel method, in which the probability of being chosen is {directely, inversely} proportional to its degree. (Methods from **D8** to **D11**.)

  5. Select {a random, the smallest} stable set of the solution, remove the stable set.

Algorithm 1: ALNS Framework

Input : Number of iterations: M
Input : Initial solution: x_0
Input : List of neighbourhoods: $\mathcal{N}$
Input : Neighbourhood scores: λ_N for $N \in \mathcal{N}$
Input : Acceptance parameters: α
Input : Objective to minimize: $f(\cdot)$

$x = x_0$
$x^* = x_0$
$i = 1$
$\lambda_N = 1, \quad \forall N \in \mathcal{N}$

while $i \leq M$ **do**
  Choose neighbourhood $N \in \mathcal{N}$ with probability proportional to λ_N
  Select $x' \in N(x)$
  if *Accept new solution* x' *(using parameters* α*)* **then**
    $x = x'$
  end
  if $f(x) < f(x^*)$ **then**
    $x^* = x$
  end
  Update scores λ
  Update acceptance parameters α
  $i = i + 1$
end
return x^*

(Methods **D12**, and **D13**.)

6. As in item 5, but the criterion used to choose the set is that it has the smallest cumulative {external, color} degree, defined as the sum of the degrees of its vertices. (Methods **D14**, and **D15**.)

7. As in item 6, but where the set is chosen with a roulette wheel method, in which the probability of being chosen is inversely proportional to the cumulative degree. (Methods **D16**, and **D17**).

- **Repair methods**

1. For each uncolored subset, select a random vertex from the subset and add it to {a random, the smallest, the largest} feasible stable set of the current solution. If it is not possible to put the vertex in any existing stable set, define a new stable set. (Methods **R1**, **R2**, and **R3**.)

2. As in item 1, but for each uncolored subset, we select the vertex with smallest {external, color} degree. (Methods from **R4** to **R9**.)

The scores of the destroy and repair heuristics are updated at each iteration as follows: if a method produced a new best solution, its score is multiplied by a parameter `AlnsBestMult`; otherwise, if a method produced a solution accepted by the acceptance criterion, its score is multiplied by `AlnsAcceptMult`; otherwise, its score is multiplied by `AlnsRejectMult`.

The classical acceptance criterion used within ALNS is Simulated Annealing, in which a solution is accepted with probability $\exp((f(x) - f(x'))/T)$, where T is a parameter (called *temperature*) that decreases exponentially during the solution process. However, the objective function $f(\cdot)$ we consider simply counts the number of used colors, and therefore it only assumes a very limited range of discrete values, while moving from one value to the next (i.e., reducing the number of colors by one) is a relatively rare occurence. For these reason, an acceptance criterion that accepts a solution based on its objective value does not seem particularly suited for the PCP. We, therefore, decided to use the "*Worse Accept*" criterion, proposed by Santini et al. [29], which accepts a new solution x' if either it uses strictly fewer colors than the current one, or otherwise with a certain probability p, which starts at a high value, and decreases linearly to reach 0 at the end of the solution process. Notice that p does not depend on the value $f(x')$ of the new solution. In our implementation, the start value is dictated by parameter `AlnsInitProb`.

4.2. Local Search refinement

The local search is a heuristic procedure that can be applied each time a new solution is generated by the destroy and repair heuristics, before the solution is evaluated. Although applying the refinement to all generated solutions certainly increases the running time of the algorithm, it also produces solutions of higher quality, and gives an important improvement on the overall quality of the algorithm, as outlined by the computational experiments reported in Section 5.3.

The local search operator tries to reduce the number of colors used in a solution by one unit, by emptying the smallest cardinality stable set in the solution. Assume the current solution uses h colors $S_1, \ldots, S_h$ and, without loss of generality, that S_h is the smallest cardinality stable set. The local search heuristic first uncolors all vertices of S_h. It then considers each uncolored partition, and tries to color any vertex (say v) of the partition by inserting it in one of $S_1, \ldots, S_{h-1}$.

If there is a stable set S_i such that $S_i \cup \{v\}$ is still a stable set, v is placed in S_i. Otherwise, the procedure tries to insert v in one stable set S_i by removing all vertices $w_1, \ldots, w_r$ in S_i that are not compatible with v, i.e., $vw_j \in E$ for $j = 1, \ldots, r$. If it is possible to greedily relocate all vertices w_j in other stable sets, the vertices are relocated, and v is inserted in S_i. If there is no stable set where vertex v can be inserted, the procedure tries to color another vertex from the same uncolored partition. The uncolored partitions, the vertices v from the uncolored partition and stable sets S_i are considered in random order.

If, for some uncolored partition, no vertex can be inserted in a stable set $S_1, \ldots, S_{h-1}$, local search is stopped. In this case the input solution is returned. On the other hand, if the local search manages to recolor one vertex for each uncolored partition, it has reduced the number of colors in the solution by one. In this case, the improved solution is used as the current solution at the next iteration of ALNS.

5. Computational Results

The experiments have been performed on a single core of a computer equipped with a 3.25 GHz 4-core i5 processor and 8Gb RAM, running a 64-bit Linux operating system. For all the tests we used a single thread.

The goal of the experiments was to assess the performance of the new B&P algorithm setting a computing time limit of one hour.

The algorithms were coded in `C++` and all the codes were compiled with `gcc 6.2` and `-O3` optimizations. At each iteration of the Column Generation procedure (see Section 3), we used `Cplex 12.6.1` as a Linear Programming solver. The pricing MWSS subproblems were solved using the open-source implementation of the algorithm described in Held et al. [15] and available at `https://github.com/heldstephan/exactcolors`. The algorithm implementation and the instances used are available from Santini [28].

5.1. Instances

In order to compare our results with the ones present in the literature, we tested our approach on the instance classes `random`, `nsfnet` and `ring` presented in Section 1.1. The entire set of instances can be downloaded at `http://www2.ic.uff.br/~celso/grupo/pcp.htm`. In their work, Hoshino et al. [16] considered a subset of 187 out of a total of 199 instances, removing those instances solved to optimality in less than a second by either their algorithm or that of Frota et al. [12].

We, in turn, removed 12 instances of the `ring` class, as we realised that they were identical copies of the same three basic instances. In particular, instances `n10_p1.0_1` to `n10_p1.0_5` all correspond to the same instance (therefore only 1 out of the 5 instances has been kept), as do the analogous instances of base type `n15` and `n20`. This reduced the total number of instances to 175. We used therefore 56 `random`, 32 `nsfnet`, and 87 `ring` instances. We also note that in `ring` instances, all elements of the partition have cardinality two.

5.2. Metaheuristic algorithm parameter tuning

The metaheuristic algorithm is aimed at providing a good quality solution to initialize the B&P algorithm in a small fraction of the available computing time of one hour. The relevant parameters have been tuned on a subset of 28 randomly selected instances (8 `random`, 5 `nsfnet`, 15 `ring`) by means of the `irace` package (see López-Ibáñez et al. [19]).

Since the number of vertices $|V|$ of an instance determines the computing time for one iteration of the heuristic algorithm, the number of iterations for an instance was set to $\beta / \log_2 |V|$, where β is a parameter to be experimentally tuned. This way, we limited the number of iterations for large instances, as explained below.

The ALNS with Local Search (ALNS+LS) algorithm has four input parameter to be tuned in addition to β, i.e., the `AlnsBestMult`, `AlnsAcceptMult`, `AlnsRejectMult`, and `AlnsInitProb`.

The parameters were tuned in two steps. First we kept the value of β fixed to a sufficiently large number, and tuned the other four parameters. In this way, we allowed the algorithm to reach its "plateau", i.e., to run for many consecutive iterations without further improvement of the objective function. In the second step, we tuned the parameter β by first choosing, for each test instance, the minimum value which allowed the ALNS with Local Search (ALNS+LS) to reach the plateau, and then taking the maximum (among all test instances) of these values, rounded to the closest multiple of 1 000. This way, we guaranteed for each instance a sufficient number of iterations to reach its plateau but at the same time the scale factor $\log_2 |V|$ limited the increase of iterations for large instances. The resulting parameter configurations were: `AlnsBestMult` = 1.9307, `AlnsAcceptMult` = 1.3667, `AlnsRejectMult` = 0.8836, `AlnsInitProb` = 0.8975, β = 15 000. Notice that ALNS+LS parameters appeared to diverge from common values in the literature (see, e.g., Santini et al. [29]). In particular, we would have expected the values of `AlnsBestMult`, `AlnsAcceptMult`, and `AlnsRejectMult` to be closer to 1.0, and that of `AlnsInitProb` to be much smaller. The flat-landscape nature of the problem could have complicated the tuning task; however, we believe that considerable noise was introduced by the high number of destroy and repair methods.

When tuning the parameters of ALNS+LS, in fact, all the 17 destroy and 9 repair methods were enabled. To assess their impact and eventually reduce their number, we ran further experiments. Let I be the set of tuning instances, and Best(i) be the best solution value obtained by ALNS for instance $i \in I$ during all the runs.

First we disabled each method one by one, and ran the algorithm on the tuning instances, with five reruns for each instance. If there was an instance i for which the ALNS without the method did not reach Best(i) in at least one of the reruns, then i was flagged for the method. We dropped all the methods with fewer than three (10%) of flagged instances. Using the approach described above, we discarded the following destroy and repair methods: **D5**, **D7**, **D9**, **D10**, **D11**, **D15**, **D16**, **R9**.

Finally, we re-tuned the ALNS+LS algorithm using the remaining destroy and repair methods. The final parameter configuration was as follows: `AlnsBestMult` = 1.1488, `AlnsAcceptMult` = 1.0877, `AlnsRejectMult` = 0.9972, `AlnsInitProb` = 0.1010, β = 21 000.

Figure 3 shows the number of times, in percentage, in which each destroy method produced an improving solution, coupled with any repair method. Figure 4 shows the analogous measure for each repair method, coupled with any destroy method. Using the notation of Algorithm 1, a new solution x' is considered improving over the current solution x if $f(x') < f(x)$ before applying the local search operator. Since we often accept worsening solutions, the percentages shown in the figure are relatively high and the lowest recorded percentage, for method D8,

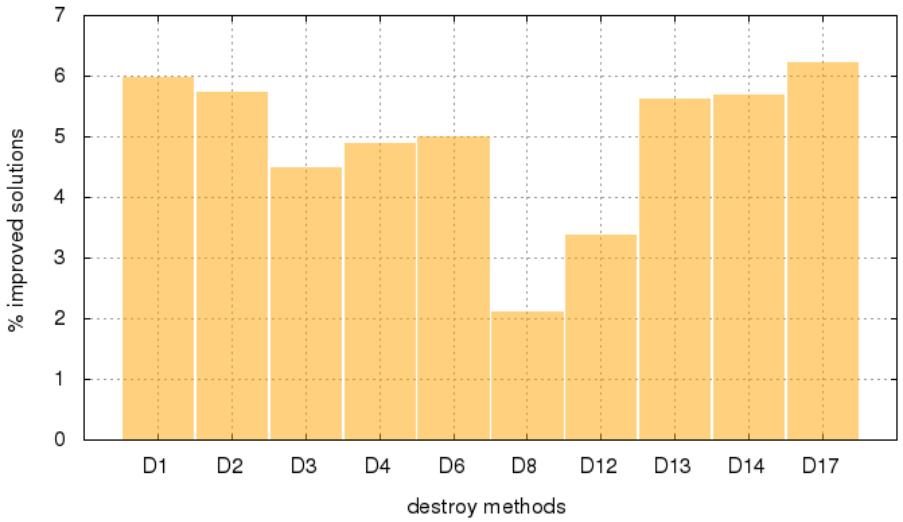

Figure 3: Percentage of times in which each destroy method has produced an improving solution (together with some repair method).

Difference with χ_P	**ALNS**		**ALNS + LS**	
	Avg	Var	Avg	Var
0	99.6	2.3	158.4	0.8
1	45.4	2.3	11.6	2.3
2	15.6	0.3	2.0	0.5
3	9.4	0.3	0.0	0.0
4	2.0	0.0	0.0	0.0

Table 1: Quality of the solution produced by ALNS and ALNS enhanced with local search.

is just above 2%. All methods show to be effective in improving the current solution. The figures refer to the 28 instances used for tuning.

5.3. Metaheuristic algorithm performance

This section evaluates the initial metaheuristic presented in Section 4. Out of the 175 instances we considered, we know the optimal result (either from our branch-and-price algorithm, or from that of Hoshino et al. [16]) of 172 of them. For these instances, we can compute how well the heuristic performs with respect to the optimal solution value.

Table 1 shows the number of instances for which ALNS and ALNS + LS has found: the optimal solution; a solution with one, two, three or four colors more than the optimum. The algorithm never produced solutions with five colors more than the optimum.

Column *Avg* reports the average across five reruns; the variance is reported in column *Var*. The table shows that the introduction of a local search phase greatly enhances the effectiveness of ALNS.

Figure 5 shows computational times. The chart shows a positive correlation between the

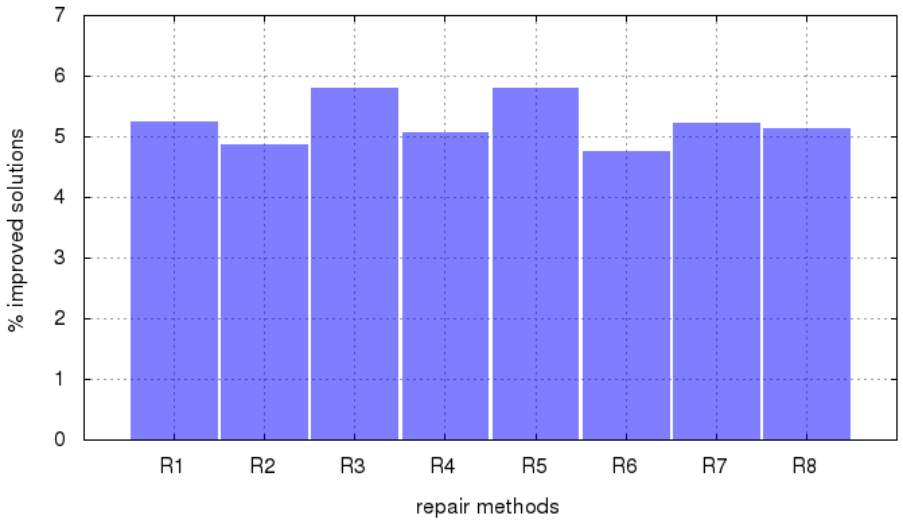

Figure 4: Percentage of times in which each repair method has produced an improving solution (together with some destroy method).

running time and the number of vertices, even if the number of iterations is scaled as explained in Section 5.2. Furthermore, the introduction of a local search phase to ALNS increases the computation time, but the improved solution quality definitely justifies such a choice. Several instances of size $|V| = 90$ are present in the test bed and the different computing time is explained by their different edge density. This figure affects the time necessary for an iteration of the metaheuristic algorithm.

5.4. Branch-and-price Algorithm

In this section, we report the results obtained by our new branch-and-price algorithm with a time limit of one hour. Since linear relaxation bounds are generally tight, the performance of our algorithm depends on the ability to find good feasible solutions (upper bounds) early in the branching tree. Therefore, at the root node we adopt the following approach based on the ALSN+LS (described in Section 4), denoted as *root node heuristic* (RH):

1. We generate an initial solution using the ALNS+LS heuristic (the corresponding columns are used to initialize the Restricted Master problem).
2. We expand the column pool using the POPULATE method of Hoshino et al. [16].
3. We solve the root node relaxation via column generation.
4. If the difference between the upper and lower bounds (UB and LB, respectively) is still greater than one, we try to strengthen the UB.
   (a) We solve ILP^{E} as an integer program with a general purpose solver (`Cplex 12.6.1`), where the set $\mathscr{S}$ only contains the generated stable sets. The solver is run with a time limit of 30 seconds.

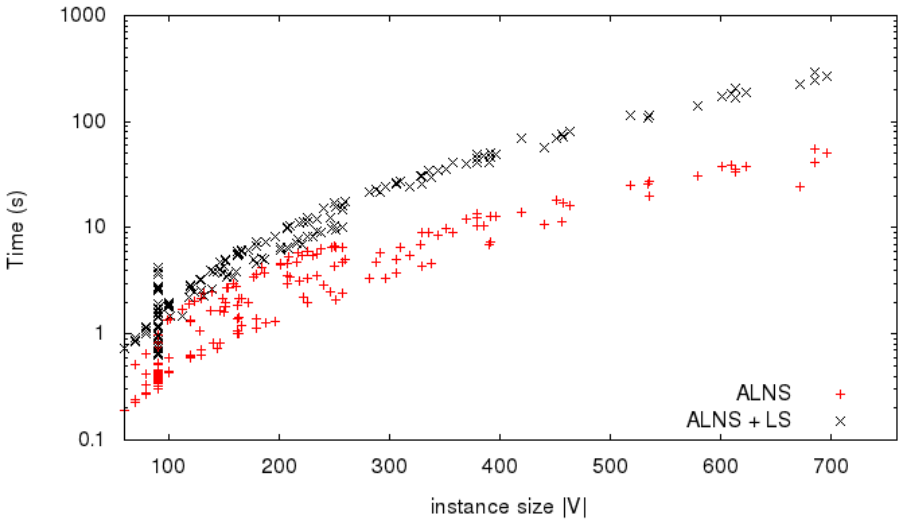

Figure 5: Running time of heuristics, versus instance size (measured in number of vertices of the graph).

(b) If the difference between UB and LB is again greater than one, we restart the ALNS+LS metaheuristic using the solution provided by the integer program as a starting solution.

Notice that these operations are only performed at the root node, because solving the MIP and executing the ALNS heuristic are time consuming. This is true especially for larger instances, which tend to be the ones for which item 4 above is executed. The time spent by the procedure described above is subtracted from the overall time budget, therefore less time can be used exploring the rest of the Branch-and-Price tree.

The RH combines the ALNS + LS with additional procedures, and is able to improve nine times on the feasible solution provided by the latter.

We divided the presentation of the computational results in two batches. In the first batch, we considered those instances that we were able to solve using the RH, without having to perform branching. In the second batch, we include those instances that required branching, and one instance for which we were not able to fully explore the root node within the time limit. The Branch-and-Price tree is explored according to a depth first strategy; the binary branching scheme we used is described in Section 3.2, and the nodes are explored in the order in which they are defined.

Table 2 presents the results relative to the instances of the first batch. Columns $|V|$ and $|E|$ report, respectively, the number of vertices and edges in the graph. Column χ_P is the partition chromatic number of the graph. Column *Cols* reports the final size of the column pool, and column *Time* is the solution time in seconds.

The first batch of instances is composed by 124 instances. The new Branch-and-Price algorithm can solve at the root node 20 out of 56 `random` instances with a computing time smaller than six seconds, all the 32 `nsfnet` instances with a computing time smaller than 22 seconds, and 72 out of 87 `ring` instances with a computing time smaller than 342 seconds. These tests

Instance	$\|V\|$	$\|E\|$	χ_P	Cols	Time	Instance	$\|V\|$	$\|E\|$	χ_P	Cols	Time
n100p5t2s1	100	2515	7	208	2.89	ring_n10p0.9s5	166	5497	13	255	6.95
n100p5t2s2	100	2460	7	233	2.71	ring_n10p1.0s1	180	6450	13	321	9.96
n100p5t2s3	100	2532	7	214	3.70	ring_n15p0.3s3	130	3430	13	201	3.63
n100p5t2s4	100	2468	7	196	2.68	ring_n15p0.3s4	140	3970	12	235	4.73
n100p5t2s5	100	2524	7	221	3.29	ring_n15p0.3s5	150	4525	12	249	5.09
n60p5t2s1	60	924	5	100	1.58	ring_n15p0.4s1	162	5343	15	178	6.46
n80p5t2s1	80	1616	6	195	1.58	ring_n15p0.4s2	188	7099	14	324	8.72
n80p5t2s2	80	1570	6	204	1.59	ring_n15p0.4s3	164	5484	15	267	7.37
n90p1t2s4	90	435	2	316	5.71	ring_n15p0.4s4	180	6549	14	273	8.31
n90p2t2s2	90	821	3	254	4.91	ring_n15p0.4s5	196	7761	16	330	11.19
n90p5t2s4	90	2040	7	220	2.12	ring_n15p0.5s1	208	8760	18	263	11.42
n90p5t2s5	90	2082	7	177	2.19	ring_n15p0.5s2	226	10280	18	376	13.97
n90p6t2s1	90	2462	8	166	2.29	ring_n15p0.5s3	208	8828	19	317	11.81
n90p6t2s2	90	2403	8	178	2.27	ring_n15p0.5s4	210	8974	17	334	13.02
n90p8t2s1	90	3268	12	142	3.44	ring_n15p0.5s5	226	10281	18	334	15.28
n90p8t2s5	90	3239	12	136	3.16	ring_n15p0.6s1	250	12567	19	390	20.01
n90p9t2s1	90	3637	16	160	4.79	ring_n15p0.6s3	258	13529	22	420	19.91
n90p9t2s2	90	3619	16	149	4.45	ring_n15p0.6s4	260	13724	20	435	24.42
n90p9t2s3	90	3621	16	153	4.67	ring_n15p0.6s5	252	12808	21	320	18.68
n90p9t2s5	90	3640	16	151	4.51	ring_n15p0.7s1	288	16745	21	468	28.48
nsf_p0.4_s3	101	625	6	206	1.89	ring_n15p0.7s2	330	21921	25	508	36.37
nsf_p0.4_s4	99	579	7	139	2.04	ring_n15p0.7s3	306	18974	23	479	34.35
nsf_p0.4_s5	112	769	6	262	2.52	ring_n15p0.7s4	310	19455	23	505	37.54
nsf_p0.5_s1	124	1030	8	358	2.93	ring_n15p0.7s5	282	16126	23	454	30.35
nsf_p0.5_s2	130	1041	8	220	3.36	ring_n15p0.8s1	336	22826	24	551	46.29
nsf_p0.5_s4	118	828	7	355	2.71	ring_n15p0.8s2	352	24949	25	549	43.64
nsf_p0.5_s5	132	1099	7	279	3.11	ring_n15p0.8s3	338	23111	26	535	39.63
nsf_p0.6_s1	149	1506	9	373	4.66	ring_n15p0.8s4	344	23984	25	543	47.18
nsf_p0.6_s2	154	1443	9	451	4.44	ring_n15p0.8s5	328	21736	25	536	41.66
nsf_p0.6_s3	153	1404	9	353	4.06	ring_n15p0.9s2	386	30115	27	651	76.92
nsf_p0.6_s4	161	1650	9	451	4.77	ring_n15p0.9s3	380	29257	28	614	63.03
nsf_p0.6_s5	139	1211	9	269	3.25	ring_n15p0.9s4	380	29233	27	621	64.37
nsf_p0.7_s1	180	2157	10	477	6.24	ring_n20p0.2s1	128	3345	10	184	4.04
nsf_p0.7_s2	202	2718	11	471	7.79	ring_n20p0.2s2	172	6040	15	288	7.74
nsf_p0.7_s3	177	1966	10	513	6.20	ring_n20p0.2s3	162	5293	13	257	6.14
nsf_p0.7_s4	187	2191	11	340	6.33	ring_n20p0.2s4	152	4727	13	258	5.45
nsf_p0.7_s5	159	1642	9	469	5.00	ring_n20p0.2s5	162	5449	16	273	6.79
nsf_p0.8_s1	201	2594	11	600	8.45	ring_n20p0.3s1	218	9618	18	347	13.22
nsf_p0.8_s2	221	3167	11	559	10.90	ring_n20p0.3s2	246	12431	24	394	15.00
nsf_p0.8_s3	208	2743	11	604	8.45	ring_n20p0.3s3	234	11068	18	369	14.98
nsf_p0.8_s4	209	2876	11	557	8.77	ring_n20p0.3s4	222	10169	20	366	14.10
nsf_p0.8_s5	184	2049	11	403	6.48	ring_n20p0.3s5	240	11922	21	390	18.75
nsf_p0.9_s1	216	2966	12	437	9.93	ring_n20p0.4s1	306	18925	24	483	30.57
nsf_p0.9_s2	231	3370	12	610	10.83	ring_n20p0.4s2	318	20852	28	513	31.86
nsf_p0.9_s3	217	2889	12	542	9.87	ring_n20p0.4s3	296	17718	21	471	30.33
nsf_p0.9_s4	226	3169	12	704	15.00	ring_n20p0.4s4	292	17451	25	465	25.46
nsf_p0.9_s5	234	3424	12	624	12.82	ring_n20p0.4s5	330	22422	29	547	34.90
nsf_p1.0_s1	250	3996	13	730	15.11	ring_n20p0.5s1	392	31120	32	645	55.26
nsf_p1.0_s2	251	4026	13	762	18.95	ring_n20p0.5s2	396	32349	32	652	63.21
nsf_p1.0_s3	238	3465	13	735	12.78	ring_n20p0.5s3	380	29251	27	608	55.95
nsf_p1.0_s4	257	4320	13	736	21.42	ring_n20p0.5s4	358	26197	28	589	51.05
nsf_p1.0_s5	248	3879	13	734	14.34	ring_n20p0.5s5	390	31285	32	640	56.36
ring_n10p0.7s3	130	3353	10	190	4.08	ring_n20p0.6s1	458	42545	36	703	93.05
ring_n10p0.8s1	146	4221	11	177	5.84	ring_n20p0.6s2	464	44339	36	769	109.62
ring_n10p0.8s2	152	4605	12	227	6.16	ring_n20p0.6s3	456	42134	32	762	103.11
ring_n10p0.8s3	144	4115	11	212	4.76	ring_n20p0.6s5	440	39665	34	710	78.43
ring_n10p0.8s4	146	4233	11	218	4.85	ring_n20p0.7s1	536	58410	39	869	158.49
ring_n10p0.8s5	138	3785	11	204	4.57	ring_n20p0.7s2	580	68948	43	985	199.56
ring_n10p0.9s1	162	5225	12	239	6.66	ring_n20p0.7s4	536	58641	38	944	218.01
ring_n10p0.9s2	164	5360	12	238	6.77	ring_n20p0.8s2	624	79813	46	1019	251.36
ring_n10p0.9s3	166	5479	13	255	7.62	ring_n20p0.8s3	614	76597	43	1005	216.42
ring_n10p0.9s4	164	5337	12	269	7.76	ring_n20p0.9s1	672	92075	48	1101	341.50

Table 2: Computational results for instances solved at the root node (RH).

			B&P [16]		B&P (new)								
Instance	$\|V\|$	$\|E\|$	LB	UB	rUB	rLB	UB	Nodes	rCols	nCols	Cols	rTime	Time
n120p5t2s1	120	3616	8	**8**	9	8	**8**	4992	135	1.00	5251	4.70	438.52
n120p5t2s2	120	3563	8	**8**	9	7	**8**	538	150	3.59	2223	4.79	31.40
n120p5t2s3	120	3638	8	**8**	9	8	**8**	11993	127	0.71	8747	4.65	1148.29
n120p5t2s4	120	3565	8	**8**	9	7	**8**	490	125	3.24	1833	4.35	22.41
n120p5t2s5	120	3653	8	**8**	9	8	**8**	9020	145	1.56	3158	4.75	1626.14
n70p5t2s2	70	1204	6	**6**	6	5	**6**	21	62	8.85	302	1.23	1.52
n70p5t2s3	70	1218	6	**6**	6	5	**6**	229	66	4.53	1178	1.13	2.92
n70p5t2s4	70	1217	6	**6**	6	5	**6**	35	63	8.53	431	1.18	1.59
n80p5t2s3	80	1611	6	**6**	7	6	**6**	2998	75	1.21	3776	1.51	38.38
n80p5t2s4	80	1595	6	**6**	7	6	**6**	497	76	2.76	1536	1.47	6.02
n80p5t2s5	80	1634	6	**6**	7	6	**6**	497	68	3.17	1733	1.70	6.18
n90p1t2s1	90	445	2	3	3	2	**3**	483	234	32.54	15981	7.75	399.67
n90p1t2s2	90	442	2	3	3	2	**3**	1451	211	27.28	39905	6.93	* 2082.66
n90p1t2s3	90	465	3	**3**	3	2	**3**	63	204	36.74	2635	3.93	33.64
n90p1t2s5	90	485	3	**3**	3	2	**3**	17	219	49.56	1074	5.84	20.98
n90p2t2s1	90	823	3	4	4	3	**4**	4881	181	6.91	34938	3.98	* 2496.60
n90p2t2s3	90	869	3	4	4	3	**4**	281	163	15.16	4512	3.49	73.09
n90p2t2s4	90	821	3	4	4	3	4	5133	176	9.27	47854	3.97	tl
n90p2t2s5	90	862	3	4	4	3	**4**	1019	178	13.28	13815	3.84	341.48
n90p3t2s1	90	1215	4	5	5	4	**5**	9085	126	3.21	29454	2.43	1652.40
n90p3t2s2	90	1234	4	5	5	4	**5**	9293	146	3.15	29573	2.32	* 2054.82
n90p3t2s3	90	1275	5	**5**	5	4	**5**	197	139	8.62	1921	2.36	18.33
n90p3t2s4	90	1211	4	5	5	4	5	11771	145	3.19	37847	2.77	tl
n90p3t2s5	90	1268	5	**5**	5	4	**5**	865	141	6.75	6 072	2.42	77.84
n90p4t2s1	90	1624	5	6	6	5	**6**	12455	114	1.46	18361	2.01	1052.99
n90p4t2s2	90	1600	5	6	6	5	**5**	3498	113	2.42	8681	2.03	159.76
n90p4t2s3	90	1650	6	**6**	6	5	**6**	865	117	3.98	3661	1.99	29.44
n90p4t2s4	90	1638	6	**6**	6	5	**6**	1777	127	3.36	6182	2.05	72.11
n90p4t2s5	90	1671	6	**6**	6	5	**6**	53	113	9.60	712	2.15	4.65
n90p5t2s1	90	2039	7	**7**	7	6	**7**	81	99	6.70	716	2.12	3.83
n90p5t2s2	90	1988	7	**7**	7	6	**7**	6357	93	1.12	7321	2.01	175.09
n90p5t2s3	90	2064	7	**7**	7	6	**7**	23	86	10.00	388	2.22	2.97
n90p6t2s3	90	2463	8	**8**	9	8	**8**	494	73	1.88	1103	2.32	5.75
n90p6t2s5	90	2478	9	**9**	9	8	**9**	1741	82	0.97	1897	2.51	13.63
n90p8t2s2	90	3200	12	**12**	12	11	**12**	49	38	4.02	327	3.11	3.35
n90p8t2s3	90	3282	12	**12**	13	12	**12**	40	37	2.79	264	3.49	3.65
ring_n15p0.6s2	258	13395	19	**19**	20	19	**19**	64	83	18.22	1528	22.25	56.79
ring_n15p0.9s1	370	27654	26	**26**	27	26	**26**	75	39	21.65	2217	57.94	359.13
ring_n15p0.9s5	392	31121	27	**27**	28	27	**27**	84	109	21.82	2536	132.95	698.32
ring_n15p1.0s1	420	35700	28	29	29	28	**28**	67	174	30.59	2838	287.97	1332.80
ring_n20p0.6s4	452	41756	32	**32**	34	32	**32**	97	92	22.45	2976	173.14	1171.84
ring_n20p0.7s3	534	57846	37	**37**	38	37	**37**	130	45	22.26	3796	214.32	* 2179.74
ring_n20p0.7s5	518	55043	38	**38**	39	38	**38**	136	88	21.18	3785	250.88	* 2054.76
ring_n20p0.8s1	614	76875	44	**44**	45	44	**44**	153	27	23.29	4560	262.78	* 2814.38
ring_n20p0.8s4	610	76046	43	**43**	44	43	**43**	35	45	37.61	1405	268.90	936.30
ring_n20p0.8s5	602	74191	43	**43**	44	43	**43**	81	18	28.11	2571	265.96	* 2088.95
ring_n20p0.9s2	696	99162	49	**49**	50	49	50	40	88	12.43	1714	807.23	tl
ring_n20p0.9s3	686	95915	—	48	48	47	**47**	16	147	24.56	544	851.66	1652.91
ring_n20p0.9s4	686	96132	—	48	48	47	**47**	21	171	17.27	1091	1383.25	* 3091.55
ring_n20p0.9s5	706	101953	49	**49**	51	49	51	19	125	5.16	1551	1250.69	tl
ring_n20p1.0s1	760	117800	—	—	51	42	50	0	—	—	1482	tl	tl

Table 3: Computational results for instances not solved at the root node.

show that the `nsfnet` instances are not computationally challenging. For all these instances there is no integrality gap between the bound provided by LP^E and the optimal solution value. A large majority of the `ring` instances are solved to proven optimality and the largest one has 672 vertices and 92075 edges. Finally less than half of the `random` instances can be solved at the root node by our Branch-and-Price method, i.e., this set of instances is the most challenging one. As far as the number of generated columns is concerned, all instances are solved with at most 1101 columns. The large majority of the columns are generated by the heuristic and the POPULATE method. The number of columns generated by solving the MWSSP is approximately one tenth of the total number, on average.

Table 3 presents the results relative to the instances of the second batch. We report under *B&P [16]* the best results obtained by any of the four implementations of Hoshino et al. [16], i.e., the final lower and upper bounds, while under *New B&P* the results obtained by our algorithm. Columns *rLB* and *rUB* are the lower and upper bounds obtained at the end of the exploration of the root node, while column *UB* is the final upper bound. Column *Nodes* displays the number of explored Branch-and-Price nodes. Column *rCols* is the number of columns generated at the root node, solving the MWSSP; column *nCols* is the average number of columns generated solving the MWSSP, at nodes other than the root node. Column *Cols* reports the total number of columns generated; this includes columns generated solving the MWSSP, as well as those generated by the heuristic, and by the POPULATE method. Finally, columns *rTime* and *Time* list, respectively, the root node and the overall solution time (both including the initial heuristic), in seconds. The values in bold under columns *UB* denote instances solved to optimality. An asterisk next to the total computation time denotes instances solved in more than 1800s, so as to facilitate comparison with Hoshino et al. [16] (see comments of Table 5 on the comparison of computing times on different computers).

Notice that we could not solve the root node of one instance (`ring_n20p1.0s1`), for which we provide a lagrangean lower bound $LB = \lceil z_{LP^E}/z_{Viol} \rceil$, where z_{LP^E} is the solution of the last linear relaxation of the restricted master problem solved, and z_{Viol} is the last solution value found by the pricing problem.

The second batch of instances is composed by 51 instances. In summary, we managed to find the optimal solution to 34 out of 36 `random` instances and 12 out of 15 `ring` instances, and in 38 cases the solution was found in less than half an hour (1800s). In 25 out of 36 `random` instances, we can notice a difference of one unit between the partition chromatic number (UB) and the optimal solution value of LP^E (rLB). To solve these instances to proven optimality, our branch-and-price algorithm explores often several thousands of nodes (max. 12455 nodes). For other two instances, namely, `n90p2t2s4` and `n90p3t2s4`, we are unable to determine if such a gap exists since the optimal solution values are not known. For all `ring` instances for which the optimal solution value is known, the partition chromatic number always coincides with the optimal solution value of LP^E. This gap for the unsolved instance `ring_n20p1.0s1` is not known. The table shows that our new branch-and-price algorithm is able to solve nine unsolved `random` instances. As far as the remaining two unsolved `random` instances are concerned (`n90p2t2s4` and `n90p3t2s4`), also none of the Branch-and-Price algorithms proposed in [16] were successful in solving them. Two out of the three `ring` instances for which our Branch-and-Price stops at the time limit with a difference between the LB and

the UB of one unit, were instead solved in [16]. The largest `ring` instance with 760 vertices cannot be solved to proven optimality but, while for the Branch-and-Price algorithms of [16] neither a LB or an UB are reported, our Branch-and-Price algorithm computes a LB of 42 and an UB of 50.

At the root node, the number of columns generated by solving MWSSP never exceeds 234 (see column *rCols* of Table 3), with the exception of `ring_n20p1.0s1` for which the root node was not solved. The average number of priced out columns in the subsequent nodes (see column *nCols* of Table 3) is smaller than 50 on average. We can then conclude that stabilizing the Column Generation phase for a such a small number of columns would not substantially improve convergence (see e.g., [11] for a discussion of classical *stabilization methods*).

Table 4 reports on the impact of the root node heuristic (RH) and of the method POPULATE on the overall performance of the Branch-and-Price algorithm. By root node heuristic we refer to all methods used at the root node: ALNS+LS, POPULATE, solving a MIP, and (possibly) re-launching ALNS+LS. The results refer to the 28 instances already used in Section 5.2 for tuning. Columns under *RH* report results obtained when the B&P algorithm is initialized via RH. Columns under *RH without* POP report results obtained when disabling method POPULATE. Columns under *Base* report results obtained disabling the RH; in this case the column pool is initialised with one dummy column which covers all partitions, and has a very high cost.

We can see that the use of RH has a dramatic impact in terms of nodes explored, columns generated, and solution time. On the other hand, since we include the time used by RH (column *rTime*), it is often the case that the root node times of configuration *Base* are lower than those for the other two configurations. The use of method POPULATE has a positive effect in reducing the number of column generation iterations at the root node, as can be observed by columns *rCols*. The ratio between columns *rCols* without and with POPULATE is on average 3.38, with median 1.77, minimum 0.78 and maximum 14.50. Concerning the associated computing times, the ratio between the columns *rTime* of *RH without Pop* and *RH* in Table 4 shows an average speedup of 2.18 with a median of 2.07, a maximum of 2.48 and a minimum of 0.64. Only in one out of the 27 instances the method made the computation slower. As far as the overall running time of the B&P algorithm without and with POPULATE, the ratio between the respective *Time* columns is 1.92 on average with a median of 2.02, a maximum of 4.47 and a minimum of 0.14. A decrease in performance was only observed in 5 instances and, noticeably for instance `ring_n15p1.0s1`. Finally, notice that we solve to optimality 27 instances with RH, 26 with RH without POPULATE, and just 22 with Base.

Table 5 concisely lists the number of instances solved to optimality, in each of the three classes, by the best algorithm in the literature (that of Hoshino et al. [16]) and by our branch-and-price algorithm. Column *Instances* lists the number of instances considered. In column *[16] (any)* we report the number of instances solved by at least one of the four implementations of Hoshino et al. [16]. Column *[16] (best)* reports the number of instances solved by the best of the four implementations, as listed in Table 5 of Hoshino et al. [16] (after removing the duplicate instances). These results are obtained with a time limit of 1800 seconds on computer equipped with a Pentium Core2 Quad 2.83 GHz with 8 GB of RAM. Our computer is equipped with a 3.25 GHz 4-core i5 processor and 8 GB RAM, and is therefore faster than the one used

	RH				RH without Pop				Base			
Instance	Nodes	rCols	Time	rTime	Nodes	rCols	Time	rTime	Nodes	rCols	Time	rTime
n120p5t2s1	4992	135	438.52	4.70	2993	172	260.65	8.91	68605	216	tl	1.92
n120p5t2s4	490	125	22.41	4.35	493	171	27.31	9.53	106273	203	tl	1.60
n40p5t2s1	1	30	0.43	0.42	1	42	1.88	1.88	37	48	0.41	0.36
n60p5t2s3	1	52	0.91	0.90	1	68	2.40	2.40	209	87	1.23	0.44
n80p5t2s4	497	76	6.02	1.47	1	107	3.28	3.23	10035	123	217.09	0.50
n90p1t2s3	63	204	33.64	3.93	63	228	32.03	4.93	1697	225	462.32	3.25
n90p2t2s1	4881	181	2496.60	3.98	4999	191	tl	5.95	6751	193	tl	2.87
n90p5t2s2	6357	93	175.09	2.01	6189	116	207.76	4.50	13448	136	511.37	0.53
nsf_p0.4_s3	1	3	1.89	1.89	1	10	3.82	3.82	48	61	1.72	0.23
nsf_p0.5_s2	1	30	3.36	3.35	1	71	6.68	6.67	63	92	3.34	0.49
nsf_p0.7_s2	1	10	7.79	7.75	1	56	14.86	14.84	134	134	69.85	0.85
nsf_p0.8_s5	1	22	6.48	6.46	1	56	13.22	13.20	200	115	14.94	0.39
nsf_p1.0_s3	1	16	12.78	12.72	1	98	25.08	25.05	513	201	580.50	3.22
ring_n10p0.3s3	1	8	0.41	0.41	1	9	1.35	1.35	7	21	0.31	0.30
ring_n10p0.3s5	1	23	0.85	0.85	1	18	2.90	2.90	11	46	0.46	0.43
ring_n10p0.7s4	1	10	3.76	3.75	1	26	8.15	8.14	27	64	0.97	0.27
ring_n10p0.8s2	1	11	6.16	6.16	1	13	12.96	12.96	31	87	2.36	0.36
ring_n15p0.2s2	1	2	2.26	2.25	1	9	4.90	4.90	19	65	0.60	0.30
ring_n15p0.2s5	1	9	2.45	2.44	1	16	6.01	6.00	23	47	0.54	0.34
ring_n15p0.5s5	1	5	15.28	15.27	1	16	31.38	31.38	63	132	10.71	0.68
ring_n15p0.6s1	1	24	20.01	19.99	1	82	43.12	43.10	65	174	15.11	1.90
ring_n15p0.9s4	1	6	64.37	63.11	1	83	123.62	123.62	75	292	512.94	18.16
ring_n15p1.0s1	67	174	1332.80	287.97	1	159	185.46	185.46	73	557	1580.96	390.05
ring_n20p0.3s4	1	3	14.10	14.09	1	5	29.56	29.56	60	123	5.82	0.82
ring_n20p0.3s5	1	10	18.75	18.73	1	45	38.88	38.86	57	153	16.27	1.24
ring_n20p0.8s1	153	27	2814.38	262.78	137	232	2738.91	542.01	131	428	tl	53.25
ring_n20p0.9s1	1	16	341.50	341.40	1	232	671.52	671.44	135	473	tl	95.15
ring_n20p1.0s1	0	310	tl	tl	0	548	tl	tl	0	869	tl	tl

Table 4: Impact of the root node heuristic (RH) and the RH without the Populate method on the overall performance of the algorithm.

Class	Instances	[16] (any)	[16] (best)	B&P 3600s	B&P 1800s
`random`	56	45	42	**54**	51
`nsfnet`	32	**32**	**32**	**32**	**32**
`ring`	87	83	76	**84**	79
Total	175	160	150	**170**	162

Table 5: Summary of the computational results from Hoshino et al. [16] and our branch-and-price algorithm.

in [16]. However, since it is not possible to precisely evaluate the relative speed of different machines, we report the results with the same time limit used in [16], i.e. 1800 seconds, in column *B&P 1800s*. Finally, column *B&P 3600s* lists the number of instances solved by our branch-and-price algorithm in 3600 seconds.

The new Branch-and-Price algorithm can solve twelve more instances than the best performing implementation in Hoshino et al. [16] with the same time limit, on a faster machine. When considering the results obtained with 3600 seconds of computing time, the new Branch-and-Price algorithm can solve eight additional instances, showing the usefulness of the extra computing time.

6. Conclusions

In this manuscript we have studied the Partition Coloring Problem (PCP), a generalization of the classical Vertex Coloring Problem with several real world applications in telecommunications and scheduling. For the PCP, we propose a new ILP formulation with an exponential number of variables and a new Branch-and-Price algorithm to effectively tackle it. Thanks to the new exact algorithm, which exploits the solutions provided by a metaheuristic algorithm in its initialization phase, as well as an effective method proposed in the literature to enrich the initial column pool, we were able to solve to proven optimality 170 out of 175 PCP instances from the literature. Extensive computational results have proven that the new Branch-and-Price framework improves on the previous state-of-the-art exact approaches from the literature.

7. Acknowledgments

The authors thank Stefan Held for making the source code for the MWSS problem available online, and Edna Hoshino for providing detailed computational results for the branch-and-price algorithm of [16]. Fabio Furini is partially supported by PGMO Gaspard Monge program. Enrico Malaguti is partially supported by MIUR (Italy), grant PRIN 2015. Thanks are due to two anonymous referees for careful reading and useful comments.

Bibliography

[1] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[2] M. Bergner, A. Caprara, F. Furini, M. E. Lübbecke, E. Malaguti, and E. Traversi. Partial convexification of general mips by Dantzig-Wolfe reformulation. In O. Günlük and G. J. Woeginger, editors, *Integer Programming and Combinatoral Optimization: 15th International Conference, IPCO 2011*, pages 39–51. Springer, 2011.

[3] M. Bergner, A. Caprara, A. Ceselli, F. Furini, M. E. Lübbecke, E. Malaguti, and E. Traversi. Automatic Dantzig–Wolfe reformulation of mixed integer programs. *Mathematical Programming*, 149(1):391–424, 2015.

[4] F. Bonomo, D. Cornaz, T. Ekim, and B. Ries. Perfectness of clustered graphs. *Discrete Optimization*, 10(4):296 – 303, 2013.

[5] M. Campêlo, R. Corrêa, and Y. Frota. Cliques, holes and the vertex coloring polytope. *Information Processing Letters*, 89(4):159–164, 2004.

[6] M. Campêlo, V. A. Campos, and R. C. Corrêa. On the asymmetric representatives formulation for the vertex coloring problem. *Discrete Applied Mathematics*, 156(7): 1097–1111, 2008.

[7] D. Cornaz, F. Furini, and E. Malaguti. Solving coloring problems as maximum weight stable set problems. *Discrete Applied Mathematics*, 217:151–162, 2017.

[8] M. Demange, J. Monnot, P. Pop, and B. Ries. On the complexity of the selective graph coloring problem in some special classes of graphs. *Theoretical Computer Science*, 540: 89–102, 2014.

[9] M. Demange, T. Ekim, B. Ries, and C. Tanasescu. On some applications of the selective graph coloring problem. *European Journal of Operational Research*, 240(2):307–314, 2015.

[10] M. Demange, T. Ekim, and B. Ries. On the minimum and maximum selective graph coloring problems in some graph classes. *Discrete Applied Mathematics*, 204:77–89, 2016.

[11] G. Desaulniers, J. Desrosiers, and M. Solomon, editors. *Column generation*. Springer US, 2005.

[12] Y. Frota, N. Maculan, T. F. Noronha, and C. C. Ribeiro. A branch-and-cut algorithm for partition coloring. *Networks*, 55(3):194–204, 2010.

[13] F. Furini and E. Malaguti. Exact weighted vertex coloring via branch-and-price. *Discrete Optimization*, 9(2):130 – 136, 2012.

[14] S. Gualandi and F. Malucelli. Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing*, 24(1):81–100, 2012.

[15] S. Held, W. Cook, and E. Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, 4(4):363–381, 2012.

[16] E. A. Hoshino, Y. A. Frota, and C. C. De Souza. A branch-and-price approach for the partition coloring problem. *Operations Research Letters*, 39(2):132–137, 2011.

[17] R. M. Lewis. *A Guide to Graph Colouring*. Springer, 2015.

[18] G. Li and R. Simha. The partition coloring problem and its application to wavelength routing and assignment. In *Proceedings of the First Workshop on Optical Networks*, page 1, 2000.

[19] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[20] E. Malaguti and P. Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, 17:1–34, 2010.

[21] E. Malaguti, M. Monaci, and P. Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, 8(2):174–190, 2011.

[22] D. Marx. Graph colouring problems and their applications in scheduling. *Periodica Polytech., Electr. Eng*, 48(1-2):11–16, 2004.

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

[24] T. F. Noronha and C. C. Ribeiro. Routing and wavelength assignment by partition colouring. *European Journal of Operational Research*, 171(3):797–810, 2006.

[25] P. M. Pardalos, T. Mavridou, and J. Xue. The graph coloring problem: A bibliographic survey. In P. M. Pardalos and D.-Z. Du, editors, *Handbook of combinatorial optimization*, pages 1077–1141. Springer, 1998.

[26] P. C. Pop, B. Hu, and G. R. Raidl. A memetic algorithm for the partition graph coloring problem. In *Extended Abstracts of the 14th International Conference on Computer Aided Systems Theory, Gran Canaria, Spain*, pages 167–169, 2013.

[27] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[28] A. Santini. alberto-santini/selective-graph-colouring, jun 2017. URL `https://doi.org/10.5281/zenodo.806149`.

[29] A. Santini, S. Ropke, and L. M. Hvattum. A comparison of acceptance criteria for the adaptive large neighbourhood search metaheuristic. Submitted, 2016.

[30] F. Vanderbeck. Branching in branch-and-price: a generic scheme. *Mathematical Programming*, 130(2):249–294, 2011.

[31] A. A. Zykov. On some properties of linear complexes. *Matematicheskii sbornik*, 66(2): 163–188, 1949.