Discrete Optimization 8 (2011) 174–190

Contents lists available at ScienceDirect

Discrete Optimization

journal homepage: www.elsevier.com/locate/disopt

An exact approach for the Vertex Coloring Problem

Enrico Malaguti [a,*], Michele Monaci [b], Paolo Toth [a]

[a] Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna, Viale Risorgimento, 2-40136, Bologna, Italy

[b] Dipartimento Ingegneria dell'Informazione, University of Padova, Viale Gradenigo, 6/A-35131, Padova, Italy

ARTICLE INFO

Article history:
Received 18 December 2009
Received in revised form 27 July 2010
Accepted 31 July 2010
Available online 21 August 2010

Keywords:
Vertex Coloring
Column generation
Branch-and-Price
Computational experiments

ABSTRACT

Given an undirected graph $G = (V, E)$, the *Vertex Coloring Problem* (VCP) requires to assign a color to each vertex in such a way that colors on adjacent vertices are different and the number of colors used is minimized. In this paper, we present an exact algorithm for the solution of VCP based on the well-known Set Covering formulation of the problem. We propose a Branch-and-Price algorithm embedding an effective heuristic from the literature and some methods for the solution of the slave problem, as well as two alternative branching schemes. Computational experiments on instances from the literature show the effectiveness of the algorithm, which is able to solve, for the first time to proven optimality, five of the benchmark instances in the literature, and reduce the optimality gap of many others.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

Given an undirected graph $G = (V, E)$, the *Vertex Coloring Problem* (VCP) requires to assign a color to each vertex in such a way that colors on adjacent vertices are different and the number of colors used is minimized.

The Vertex Coloring Problem is one of the classical NP-hard problems (see [1]), and it is well known not only for its theoretical aspects and for its difficulty from the computational point of view, but also because it appears in many real world applications, including, among many others, scheduling [2,3], timetabling [4], register allocation [5], frequency assignment [6] and communication networks [7].

Despite the relevance of the problem, few exact approaches have been proposed in the literature, and they are able to consistently solve only small size instances. The only recent contribution presenting extensive computational results on benchmark instances is based on the so-called descriptive formulation, which is strengthened by means of inequalities derived from the structure of the problem (see [8,9]). In this paper we present an effective algorithm for the exact solution of VCP, based on the alternative Set Covering formulation, for which the last paper presenting extensive computational results on commonly considered benchmark instances dates back to 1996 (see [10]). For speeding up the solution of the continuous relaxation of the Set Covering formulation, we propose effective methods, including the use of a metaheuristic procedure and of the advanced capabilities of modern *Integer Linear Programming* (ILP) solvers. In addition, we compare two alternative branching schemes. Finally, the use of an effective metaheuristic algorithm for VCP, used to initially produce a very good upper bound, determines a further improvement of the effectiveness of the algorithm.

The perspective of this paper is mainly computational, and we think that its main contributions are:

- presenting an effective exact algorithm for VCP, able to solve, for the first time to proven optimality, some instances which have been "open" for several years, and to improve the best lower bound of many others;

* Corresponding author. Tel.: +39 0512093942; fax: +39 0512093073.
E-mail addresses: enrico.malaguti@unibo.it (E. Malaguti), monaci@dei.unipd.it (M. Monaci), paolo.toth@unibo.it (P. Toth).

1572-5286/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.disopt.2010.07.005

- illustrating an effective integration of metaheuristic approaches and exact methods;
- showing how modern computational tools can be used to tackle a branching scheme which otherwise would be hard to implement.

We recall here some definitions used in the following. Let n and m be the cardinalities of vertex set V and edge set E, respectively. A subset of V is called a *stable set* if no two adjacent vertices belong to it (note that, in VCP, all the vertices having the same color form a stable set, and viceversa). A *clique* of a graph G is a complete subgraph of G (note that the size of a clique represents a valid lower bound for VCP). A stable set (resp. clique) is *maximal* if no vertex can be added still having a stable set (resp. clique). A k *coloring* of G is a partition of V into k stable sets. Each stable set of a coloring is called a *color class*. An optimal coloring of G is a k *coloring* with the smallest possible value of k (the *chromatic number* $\chi(G)$ of G). For each vertex $v \in V$, let $N(v)$ be the neighborhood of v, i.e., the set of vertices adjacent to v, and $\delta(v)$ the degree of v (i.e. $\delta(v) = |N(v)|$). We say that v *dominates* w if $N(w) \subset N(v)$.

The paper is organized as follows. In Section 1.1 we review the main contributions and models proposed in the literature for VCP, including the Set Covering formulation. In Section 2 we present our algorithm: Section 2.1 describes an efficient heuristic from the literature that is embedded in the exact method, Section 2.2 presents methods for effectively handling the exponential number of variables in the model. Two branching schemes are evaluated and discussed in Section 2.3. Finally, Section 3 gives the results of our experiments on a large set of instances from the literature, and Section 4 draws some conclusions.

1.1. Literature review

As mentioned in the introduction, VCP and its variants have been widely studied in the literature so far. We recall here the main contributions, and refer the reader to the recent survey by Malaguti and Toth [11] for an extensive discussion. We note that VCP appears either directly or as a subproblem in many real world applications in different contexts. However, the state-of-the-art exact algorithms for VCP are able to solve consistently only small randomly generated instances, with up to 100 vertices (see, e.g., [12]), whereas real world applications commonly deal with graphs having hundreds or thousands of vertices. This motivates the large amount of literature concerning the heuristic and metaheuristic approaches for VCP.

Among the greedy algorithms we mention the sequential algorithm (generally called SEQ), that considers the vertices in a given order and assigns each vertex to the lowest-indexed color class in which it fits, and the *Recursive Largest First* (RLF) algorithm by Leighton [2], which colors the vertices, one class at a time, in a greedy way. As to the metaheuristics, the first proposed algorithm was the Tabu Search procedure TABUCOL by Hertz and de Werra [13]. This algorithm solves the problem in its *decision form*, i.e., it receives in input a threshold value k representing the desired solution value, and considers k available colors, moving among complete colorings. Infeasible solutions, corresponding to infeasible color classes, are considered during the evolution of the algorithm and penalized in the objective function. Johnson et al. [14] proposed a simulated annealing algorithm and computationally studied different neighborhoods. Davis [15] presented a genetic algorithm in which each solution is encoded as a permutation of the vertices, which are then colored through the SEQ algorithm. The *Impasse Class Neighborhood* proposed by Morgenstern [16] turned out to be for a long time the most effective approach for VCP. Given a threshold value k, each solution S is a partition of the vertex set in $k + 1$ color classes $\{V_1, \ldots, V_k, V_{k+1}\}$ in which all classes, but possibly the last one, are stable sets. Given the current solution, its neighborhood is defined as the set of solutions that can be obtained by moving a vertex from color class $k + 1$ to a different color class, and is explored to minimize the global degree of the uncolored vertices:

$$f(S) = \sum_{v \in V_{k+1}} \delta(v). \tag{1}$$

An effective evolutionary algorithm HCA (Hybrid Coloring Algorithm) was proposed by Galinier and Hao [17]; this algorithm works with a fixed k, and combines an improved version of TABUCOL with a crossover operator which is specialized for the VCP, thus obtaining one of the most performing algorithms for the problem. Algorithm MIPS-CLR by Funabiki and Higashino [18] is a combination of a Tabu Search technique and of the *Impasse Class Neighborhood*; a major difference with most of the other algorithms is that MIPS-CLR works with a variable k, i.e. in the *optimization form* of VCP. Galinier et al. [19] proposed an *adaptive memory* algorithm AMACOL, that works with fixed k, and embeds an improved version of TABUCOL. Recently, Blöchliger and Zufferey [20] proposed two Tabu Search algorithms working with fixed k and based on the *Impasse Class Neighborhood*, while Plumettaz et al. [21] presented an Ant Colony scheme that incorporates a local search procedure adopting the same neighborhood.

Finally, we mention the evolutionary algorithm recently proposed by Malaguti et al. [22], based on the *Impasse Class Neighborhood* and a crossover operator, which is an adaptation of the *Greedy Partitioning Crossover* proposed by Galinier and Hao [17]. The evolutionary algorithm, which solves the problem for a fixed value of k, is then embedded in an overall algorithm, called MMT, which solves the optimization form of the problem and is described in Section 2.1.

The large amount of literature on the heuristics for VCP has not a similar counterpart for what concerns the exact methods. Among them, we mention the Branch-and-Bound algorithm proposed by Brown [23], which is based on the idea of coloring

the graph one vertex at the time, by using colors already assigned to vertices of the graph or a new color. The same idea was further developed by Brélaz in the DSATUR algorithm [24] and, later, further improved by Sewell [25]. Herrmann and Hertz [12] presented a different approach based on the detection of the smallest subgraph that has the same chromatic number as the original graph.

The first ILP model proposed in the literature for VCP is the so-called descriptive formulation. In this model at most n colors are considered and two sets of binary variables are used: variables x_{ih} ($i \in V$, $h = 1, \ldots, n$), with $x_{ih} = 1$ if and only if vertex i is assigned to color h, and variables y_h ($h = 1, \ldots, n$), with $y_h = 1$ if and only if color h is used in the solution. A possible model for VCP, called VCP-ASS since vertices are assigned to colors, reads:

$$\text{(VCP-ASS)} \quad \min \sum_{h=1}^{n} y_h \tag{2}$$

$$\sum_{h=1}^{n} x_{ih} = 1 \quad i \in V \tag{3}$$

$$x_{ih} + x_{jh} \le y_h \quad (i, j) \in E, h = 1, \ldots, n \tag{4}$$

$$x_{ih} \in \{0, 1\} \quad i \in V, h = 1, \ldots, n \tag{5}$$

$$y_h \in \{0, 1\} \quad h = 1, \ldots, n. \tag{6}$$

Objective function (2) minimizes the number of colors used. Constraints (3) require that each vertex is colored, while (4) impose that at most one of a pair of adjacent vertices receives a given color, when the color is used. Finally, (5) and (6) impose the variables to be binary.

This model has been extensively used in the literature, mainly for its simplicity. Recently, Méndez-Díaz and Zabala [8,9] considered this model when additional inequalities are added to eliminate symmetries – which are one of the most relevant drawbacks of such models for any ILP solver – and to obtain a stronger continuous relaxation. In this way, different ILP models for VCP are obtained, which are computationally tested, in terms of the corresponding continuous relaxation lower bound and exact solution, both on random instances and on instances from the literature.

An alternative model for VCP is the so-called *Set Covering* formulation (VCP-SC), originally proposed by Mehrotra and Trick [10]. Let $\mathcal{S}$ be the family of all the stable sets of G. Each stable set (column) $s \in \mathcal{S}$ has an associated binary variable x_s taking value 1 if and only if the vertices of s receive the same color. VCP can be formulated through the following ILP model:

$$\text{(VCP-SC)} \quad \min \sum_{s \in \mathcal{S}} x_s \tag{7}$$

$$\sum_{s \in \mathcal{S}: i \in s} x_s \ge 1 \quad i \in V \tag{8}$$

$$x_s \in \{0, 1\} \quad s \in \mathcal{S}. \tag{9}$$

Objective function (7) minimizes the total number of stable sets (and hence of colors) used. Constraints (8) state that every vertex i in the graph must belong to at least one stable set (noting that a feasible solution can be easily obtained if some vertex i belongs to more than one stable sets).

This latter formulation (7)–(9) was used by Mehrotra and Trick [10] for developing a Branch-and-Price algorithm. Their algorithm is *robust* in the sense that branching does not change the structure of the subproblem to be solved at each node of the branch decision tree to obtain the optimal solution of the associated continuous relaxation. Computational results show that both the improved version of algorithm DSATUR by Sewell [25] and the algorithm by Mehrotra and Trick [10] can consistently solve instances with up to 70 vertices for random graphs and 250 vertices for random geometric graphs.

In [26], Hansen et al. propose a *Set Packing* formulation (VCP-SP), which is obtained from a *Set Partitioning* formulation (i.e., the VCP-SC formulation where the inequality constraints (8) are replaced with the corresponding equality constraints) after a simple transformation. The corresponding ILP model is equivalent, in terms of the continuous relaxation lower bound, to the VCP-SC formulation (see [26]), and computational experiments on a set of random, geometric random and queen graphs did not show a clear superiority of one model with respect to the other when integer solutions are considered.

2. The Branch-and-Price algorithm

In this section we present our exact algorithm, providing details on its most relevant components. In an initialization phase we execute the MMT metaheuristic algorithm [22], which, in addition to an (usually very tight) upper bound, produces a set of alternative feasible solutions to the problem, which are stored in a *pool* of solutions. The exact algorithm requires the optimal solution of model (7)–(9), defined by a very large number of variables. When medium and large sized instances are considered, the explicit enumeration of all the variables can be impossible, and one has to resort to column generation

techniques to solve the continuous relaxation of the model. The column generation procedure, however, does not start from scratch, since a large number of good columns can be retrieved from the pool of solutions generated by MMT. The optimal value of the continuous relaxation of model (7)–(9) provides a very tight lower bound on the optimal solution value, which often is able to prove the optimality of the best incumbent solution. When this is not the case, we apply the Branch-and-Price algorithm described in Section 2.3.

2.1. Initialization: algorithm MMT

In this section we review the heuristic algorithm MMT, proposed by Malaguti et al. [22], that is used within the exact algorithm to improve the performance of the method. Algorithm MMT is used both for obtaining an initial feasible solution (i.e., an upper bound for VCP) and for initializing with a suitable set of columns model (7)–(9) that is used within the exact algorithm.

Algorithm MMT is a heuristic algorithm, based on model (7)–(9), that operates in two phases. In the first phase a large set $\mathcal{S}' \subset \mathcal{S}$ of stable sets of G is generated by means of greedy algorithms for VCP (SEQ, DSATUR, RLF) and by an Evolutionary algorithm. This latter metaheuristic works in decision form (i.e., with a fixed number k of colors), and is executed with different values of k or until a given time limit is reached. The algorithm combines a Tabu Search algorithm derived from the *Impasse Class Neighborhood* proposed by Morgenstern [16], and a diversification operator based on the *Greedy Partitioning Crossover* proposed by Galinier and Hao [17]. In this phase, MMT computes as well a lower bound based on a maximal clique computation, which sometimes is able to prove the optimality of the best incumbent solution. If this is not the case, all the stable sets of G generated during this phase are stored in a pool and used to define a set $\mathcal{S}'$ of variables for a reduced version of model (7)–(9), that is solved in the second phase of the algorithm by means of the heuristic procedure for the Set Covering Problem proposed by Caprara et al. [27]. To improve the performance of the approach, only maximal stable sets are stored in the pool; non-maximal stable sets are enlarged using a greedy procedure.

In the proposed Branch-and-Price algorithm, we execute MMT at the root node of the branch decision tree to produce an (often tight) upper bound and a set of "good" columns to initialize the set of columns to be used for the solution of the continuous relaxation of model (7)–(9). The remaining columns that have to be defined to obtain a valid lower bound for VCP are generated by means of the column generation procedure described in the next section.

2.2. Column generation procedure

Model (7)–(9) has an exponential number of binary variables (corresponding to the exponentially many stable sets of G), hence we have to apply *column generation* techniques to generate only the variables we need.

By relaxing the integrality constraint (9) to:

$$x_s \geq 0 \quad s \in \mathcal{S} \tag{10}$$

we obtain the so-called *master problem*, whose rounded-up optimal solution value represents a lower bound on the optimal solution value of (7)–(9). To solve the master problem, we first initialize it with a subfamily $\mathcal{S}'$ of the family of all the stable sets $\mathcal{S}$ of G, thus obtaining a *restricted* master problem. This subfamily is composed by the stable sets of the best incumbent solution (computed by MMT), previously expanded to maximal stable sets by means of a greedy procedure. For each stable set, the greedy procedure considers the vertices not included in the set according to a non-increasing degree order and tries to insert them in the set. We then solve the restricted master problem to optimality, and obtain the optimal values π_i^*, $(i \in V)$, of the dual variables associated with constraints (8). To detect violated dual constraints, corresponding to variables (stable sets) to be added to the restricted master problem, we solve the following *slave problem*, where each binary variable y_i, $(i \in V)$, takes value 1 if and only if vertex i is inserted in the stable set:

$$\max \sum_{i=1}^{n} \pi_i^* y_i \tag{11}$$

$$y_i + y_j \leq 1 \quad (i, j) \in E \tag{12}$$

$$y_i \in \{0, 1\} \quad i \in V. \tag{13}$$

Model (11)–(13) defines a *Maximum Weighted Stable Set Problem* (MWSSP), with weights π^*. This problem is clearly NP-hard on general graphs, since it corresponds to a *Maximum Weighted Clique Problem* on the complement of graph G. If the optimal solution of MWSSP has value greater than one, then we have found a stable set with negative reduced cost, and we add the corresponding column to the restricted master problem and iterate. Otherwise, the current restricted master problem contains all the columns corresponding to an optimal solution, and hence we have reached the optimal solution of model (7), (8) and (10). Although the variables of the master problem do not have necessarily to represent *maximal* stable sets, it is helpful for the convergence of the algorithm to expand the generated stable sets to maximal ones, before inserting the corresponding columns in the restricted master problem (otherwise a maximal column might

have to be generated in a subsequent iteration of the algorithm). This is generally done, with the exceptions discussed in Section 2.3.

Despite the ease of the column generation scheme depicted above, one should consider that solving MWSSP on medium to large sized graphs is a challenging task from the computational viewpoint. For this reason, effective procedures to solve the slave problem are a key ingredient of the proposed algorithm.

As previously mentioned, algorithm MMT produces a large number of feasible solutions to model (7)–(9), and each stable set corresponding to these solutions is expanded to a maximal stable set and stored in a pool of columns. Thus, at each iteration of the column generation procedure, we first scan this pool, looking for columns having negative reduced cost. If one or more columns are found, they are added to the restricted master problem which is solved again. If no column is found in the pool, we try to generate a negative reduced profit column by means of a Tabu Search algorithm which produces maximal stable sets. If the Tabu Search algorithm finds a column having negative reduced cost, the column is added to the restricted master problem which is re-optimized. Otherwise, we tackle directly model (11)–(13) by using a general purpose ILP solver, and finally generate a column to be added to the restricted master problem or conclude that the master problem is optimally solved if such a column does not exist. Note that although exact algorithms for MWSSP (or for the equivalent Maximum Weighted Clique Problem) exist in the literature, e.g. the *cliquer* algorithm by Östergård [28], the use of a modern general purpose ILP solver has the advantage of easily producing an upper bound on the optimal value of the slave problem during the computation, which is used to compute a valid lower bound for VCP before the master problem is optimally solved (see Section 2.2.2). Moreover, one of the branching schemes proposed for finding the optimal solution of VCP (see Section 2.3) changes the structure of the slave problems, which can be tackled thanks to the flexibility of the ILP solver.

2.2.1. Tabu Search algorithm for the slave problem

The Tabu Search algorithm is aimed at determining a maximal stable sets of graph G, having large weight with respect to a vector π of non-negative weights associated with the vertices. The literature on heuristic and metaheuristic algorithms for the maximum clique problem (which is equivalent to the maximum stable set problem, i.e., the stable set having maximum cardinality) is very rich (see, e.g., [29]), and many ideas can be adapted to the weighted clique (resp. stable set) problem. An effective neighborhood for local search algorithms is based on *swap* moves, and was used by Grosso et al. [30] to design one of the state-of-the-art algorithms for the maximum clique problem (we refer the reader to this paper for a review of other heuristic and metaheuristic algorithms). The swap move consists in removing from the current clique a vertex if this allows one or more new vertices to enter the clique. We adapt the swap move to the weighted case, and embed it in a Tabu Search algorithm.

We use as weights the values of the dual variables associated with constraints (8). Any negative reduced cost column (i.e., any stable set whose weight is larger than one) could be added to the restricted master problem. At the same time, it is well known that adding columns with large negative reduced cost speeds-up the convergence of the column generation methods. Hence, we set a threshold $\beta > 1$ and stop the algorithm as soon as a stable set with weight larger than β is found. The detailed steps of the Tabu Search Algorithm are described in Fig. 1.

Initially the algorithm computes a maximal stable set s: the vertices are considered according to a random order and each vertex i is inserted into the stable set s if it is not adjacent to any vertex $j \in s$. The algorithm is then executed for at most L iterations. At each iteration, we consider each vertex j which is adjacent to exactly one vertex (say h) of s, and thus could enter the set by removing vertex h from the set (1–1 *exchange*). The best exchange (in terms of the weight of the corresponding solution) is executed, even if this reduces the weight of the current solution s. Since the set s, after the exchange, may be not maximal, it is randomly expanded to a maximal stable set (line 11). In order to avoid cycling, a vertex h which leaves the set s is not considered for reinsertion for the next T iterations (this value defines the so-called *tabu tenure*, line 9), unless inserting h would improve the best incumbent solution (line 7). We adopt a dynamic tabu tenure, by setting T equal to the current cardinality of the stable set s. If a 1–1 *exchange* is performed, the algorithm moves to the next iteration. Otherwise (line 14), we try a 2–1 *exchange*, which consists in inserting a vertex j which is adjacent to exactly two vertices (say h and l) of s, and removing these two vertices from s. If neither a 1–1 *exchange* nor a 2–1 *exchange* can be performed, the algorithm is stuck. Thus we stop the execution (line 23) and possibly call again the algorithm from scratch (thus computing a new random initial maximal stable set).

2.2.2. Tackling the slave problem with an ILP solver

When the Tabu Search algorithm described in the previous section fails in finding a column with negative reduced cost, we tackle directly the slave model (11)–(13) with a general purpose ILP solver, in our case CPLEX10.1.

Since the edge constraints (12) are weak, we replace them with a set of stronger clique constraints. In order to obtain the clique constraints, we associate with each edge $(i, j) \in E$ a "profit" $\delta(i) + \delta(j)$. We consider the edges according to non-increasing profits and initialize the clique K with the two vertices corresponding to the edge of largest profit. Initially all the edges in E have to be covered. Iteratively, we consider the edges (i, j) such that at least one between i and j is not in K. If $K \cup \{i, j\}$ is a clique then we insert the vertices i and j in K (if they are not already included). When K is a maximal clique we add the corresponding constraint (14), and initialize a new clique K (but we keep the information on the vertices

```
begin
    randomly construct a maximal stable set s;
    s* = s; π(s*) = Σ_{i∈s*} π_i; tabu list = ∅;
    if π(s*) > β return s*;
    for (iterations = 1 to L)
        max = −1;
// 1-1 exchange
        for each j ∈ V : |N(j) ∩ s| = 1
        let h ∈ s be such that (h, j) ∈ E;
            if (π(s ∪ {j} \ {h}) ≥ max and j not in tabu list)
            or (π(s ∪ {j} \ {h}) > π(s*)) then
                s := s ∪ {j} \ {h}, max = π(s),
                insert h in tabu list, (h cannot enter s for the next T iterations)
        end for each
        if max = −1 then
// 2-1 exchange
            for each j : |N(j) ∩ s| = 2
            let h, l ∈ s be such that (h, j), (l, j) ∈ E;
                if (π(s ∪ {j} \ {h, l}) ≥ max and j not in tabu list)
                or (π(s ∪ {j} \ {h, l}) > π(s*)) then
                    s := s ∪ {j} \ {h, l}, max = π(s);
                    insert h, l in tabu list (h and l cannot enter s for the next T iterations)
            end for each;
        randomly expand s to a maximal stable set;
        if π(s) > π(s*), s* = s;
        if π(s*) > β return s*;
        if max = −1 return s*;
    end for
end.
```

Fig. 1. Steps of the Tabu Search algorithm.

included in a previous clique, so as not to consider an edge which is already covered), until all the edges of E are covered.

$$\sum_{i \in K} y_i \leq 1. \tag{14}$$

Since a large amount of computing time could be spent in proving the optimality of the slave model solution, we stop the solver as soon as a column having negative reduced cost is found.

During the solution of the slave problem with CPLEX10.1, when an improved upper bound UB_{SP} on the optimal solution value of the slave problem is detected, we can compute a lower bound LB_{MP} on the optimal solution value of the master problem as follows (see, e.g., [31]):

$$LB_{MP} = \frac{\sum_{i \in V} \pi_i^*}{UB_{SP}}. \tag{15}$$

Of course, knowing the optimal solution value of the slave problem instead of an upper bound UB_{SP} improves the lower bound value on the master problem. This lower bound can be used to prove optimality of a feasible VCP solution or to fathom a node in a Branch-and-Bound scheme. In addition, the column generation at a given node can be stopped as soon as the rounded-up values of the current master problem solution and of LB_{MP} are equal, because this implies that the rounded-up value of LB_{MP} will not further improve. In our approach we always stop the column generation procedure whenever a lower bound on the master problem leads to one of the above conclusions.

Finally we want to briefly describe some tricks which may be used to speed-up the convergence of column generation techniques, but that did not produce the desired results in our preliminary computational experiments, and so were not included in the final version of our approach.

As mentioned above, the convergence of the column generation method requires that any column with negative reduced cost is added to the master problem before the latter is re-optimized. However, if on one hand adding columns with larger negative reduced cost reduces the overall number of columns to be generated before the method converges, on the other hand computing columns with larger negative reduced cost is more time consuming, especially in the last iterations of the method. Thus, when solving the slave problem with CPLEX10.1, we tested the following strategy: we do not stop the computation at the first negative reduced cost column, instead, we stop as soon as the current value of the LB_{MP} bound is improved (or the slave problem is optimally solved). We also tested a slightly different strategy, consisting in stopping the computation as soon as the lower bound returned by the slave problem is within a specified percentage of the best LB_{MP} previously computed. The strategy is actually effective in reducing the number of columns needed by the column generation

procedure, but the method requires a fine tuning of the threshold, which strongly depends on the particular instance to be solved.

A second procedure tried to speed-up the column generation consists in adding valid inequalities to the dual of the master problem, following the idea proposed by Valério de Carvalho for the Bin Packing Problem (BPP) [32]. The general idea in [32] is to impose constraints on the domain of the dual variables, so as to reduce oscillation of these variables, which corresponds to primal degeneracy of the master problem. In particular, given two vertices i and j such that $N(i) \setminus \{j\} \subseteq N(j)$ the following constraint is valid for the dual of the master problem:

$$\pi_i \leq \pi_j. \tag{16}$$

Adding such a constraint to the dual problem corresponds to introducing an artificial column to the master problem with a -1 entry at row i and a 1 entry at row j, with objective function coefficient equal to zero. A possible primal interpretation of this artificial column is the following: given a column of the master problem covering row j and not covering row i, one can obtain a new column (i.e., a stable set) by covering row i instead of row j (i.e., by adding i and removing j from the set). Note that if $(i, j) \notin E$, one could simply add i without removing j. However, in this case one should simply remove i from the set of vertices V, since any coloring of the induced subgraph obtained from G by removing vertex i can be extended to a coloring of G (and this is actually what we do in the preprocessing procedure described in Section 3). This method slightly reduces the number of columns needed by the column generation procedure, but the overall computing time is increased and thus we do not apply it in our method.

Finally, we also considered the possibility of not adding all the clique constraints (14) at the beginning of the slave problem solution, but to add them as soon as they are violated during the computation, according to a Cutting Plane approach. However, this procedure did not improve the performance of the algorithm, and thus we do not apply it in our method.

2.3. Branching scheme

The rounded-up value of the optimal solution of the master problem (7), (8) and (10) is a lower bound on the solution value of VCP. We call this bound LB_{SC}. If LB_{SC} equals the value of the best incumbent integer solution, or the solution of the master problem is integer, the problem is optimally solved. Otherwise, we need to embed the column generation scheme within a Branch-and-Bound algorithm, obtaining what is usually defined as a *Branch-and-Price* algorithm.

We consider and compare two alternative branching schemes: the first one implements a standard branching on the variables of model (7)–(9), while the second one exploits the specific structure of VCP.

Before discussing in detail the two schemes, we want to mention that in both cases we choose as branching variable the one having the largest fractional part in the current solution of the master problem. We also considered the selection of the variable which produces the largest improvement in the lower bound if chosen (*strong branching*). However this strategy, which is more time consuming at each node of the search tree but is expected to reduce the overall number of explored nodes, did not produce satisfactory results in our implementation.

Branching on variables

We choose the variable of the VCP-SC model (7)–(9) having the largest fractional part, say $x_{\bar{s}}$, and generate the two descendent nodes by fixing $x_{\bar{s}}$ first to 1, and then to 0.

Each node of the branching tree is solved through the column generation approach previously described. When a variable $x_{\bar{s}}$ is fixed to 1, the resulting subproblem is still a VCP, where the constraints (8) associated with vertices $i \in \bar{s}$ are automatically satisfied. Conversely, when a variable $x_{\bar{s}}$ is fixed to 0, we must forbid stable set $\bar{s}$ to be generated in any descendent subproblem. To be sure that the Tabu Search algorithm does not produce a column corresponding to a forbidden stable set, a pool of forbidden stable sets is maintained, and each improving solution is compared with the sets in the pool before updating the best incumbent solution (lines 12 and 21 in Fig. 1). When tackling the slave model with the ILP solver, two alternatives are available, depending on the possibility to include in the VCP-SC model columns which are not maximal (i.e., corresponding to not maximal stable sets).

If we decide to include only maximal columns in the model, to be sure that a forbidden column $\bar{s}$ is not generated in the slave problem we add to the latter the constraint:

$$\sum_{i \in \bar{s}} y_i \leq |\bar{s}| - 1, \tag{17}$$

imposing that at most $|\bar{s}| - 1$ vertices among the ones in set $\bar{s}$ are chosen. If the slave problem outputs a non-maximal column, we have to make it maximal before adding it to the master problem. This can be done by solving to optimality an auxiliary slave problem, which consists in maximizing the number of vertices which are added to the current column. This objective will force the optimal solution to be a maximal column, unless this is forbidden by some constraint of type (17). In this case, we do not add the current column to the restricted master problem, but explicitly forbid it in the slave problem by means of a corresponding constraint (17) and re-optimize the slave problem.

On the other hand, we can choose to include non-maximal columns in the master problem. When a non-maximal column is chosen for branching and has to be forbidden, we insert it into a pool of forbidden non-maximal columns, and impose CPLEX10.1 not to generate it by means of a suitable callback. In practice, before updating the best incumbent solution of the slave problem (11)–(13), CPLEX10.1 returns the candidate column, that we can compare with the columns in the pool, thus discarding it if present in the pool.

Branching on edges

This well-known branching rule for VCP was proposed by Zykov [33] and used by Mehrotra and Trick [10]. The basic idea is as follows: at each node of the search tree we select two vertices i and j such that $(i, j) \notin E$. We then consider the two subproblems corresponding to:

1. assigning the same color to i and j; this is imposed by collapsing i and j into a single vertex, which is obtained by removing i and j from the set of vertices and creating a new vertex k, such that $(k, h) \in E$ for every vertex h for which $(i, h) \in E$ or $(j, h) \in E$;
2. assigning different colors to i and j; this is imposed by adding an edge between i and j, i.e., by setting $E = E \cup (i, j)$.

In both cases the subproblem is still a VCP, i.e., the branching scheme is robust in the sense that it does not change the structure of the subproblems.

In selecting the two vertices i and j for the branching we follow [10]: first we determine the most fractional variable, say x_{s_1}. Let i be the first row covered by column s_1. Then we determine another column s_2 (in the base of the current solution of the master problem) that covers row i. Finally we find row j such that only one of the columns s_1 or s_2 covers it.

We can always complete the columns generated by the slave problem to maximal columns in a greedy way. However, when backtracking from a node of the search tree to one of its ancestors, columns which correspond to maximal stable sets in the graph associated with the node may be no more maximal. One can simply keep these non-maximal columns in the pool or, as an alternative, can remove these columns from the master model and store them in a special pool, to be checked for negative reduced cost columns in the next iterations of the method.

3. Computational experiments

In this section we report the computational results obtained by the exact algorithm presented in Section 2 on a subset of the DIMACS benchmark instances.

The DIMACS benchmark collects a large set of instances, which represent the standard set for experimenting algorithms for the Vertex Coloring Problem (see [34], all instances are available at ftp://dimacs.rutgers.edu/pub/challenge/graph/). The benchmark set includes: random graphs (DSJC), where for each pair of vertices $i, j \in V$, edge $(i, j) \in E$ is created with uniform probability; geometric random graphs (DSJR and r), where vertices are randomly distributed in a unit square, and an edge $(i, j) \in E$ is created if the distance between i and j is less than a given threshold; artificial graphs (Leighton, Queen, myciel, Insertion, FullIns, latin_square_10); register allocation graphs (musol, zeroin, inithx, fpsol); graphs from real world applications (see the web page http://mat.gsia.cmu.edu/COLOR03/ for a complete description of the instances).

Our test-bed includes 115 instances from the DIMACS benchmark, namely those considered in [8,9], for which extensive computational results are available, allowing a comparison of different algorithms. This set of instances includes all the instances considered in [22], where we described and computationally tested the metaheuristic algorithm MMT, with the exception of the r and flat ones, for which there is no computational comparison in the literature on exact approaches for VCP.

Some instances from the DIMACS benchmark can be preprocessed in order to reduce their size. This is done by removing vertices with the property that, if they are eliminated from the graph, any coloring of the resulting graph may be extended to a coloring of the original graph with no extra color. First, any dominated vertex i can be removed from the graph. Actually, if $N(i) \subset N(j)$, at least one of the colors assigned to a vertex in $N(j)$ can be assigned to vertex i. In addition, given a lower bound on the chromatic number of G, that we compute in a greedy way as the cardinality $\omega(G)$ of a maximal clique, one can remove every vertex i such that $\delta(i) < \omega(G)$. We refer the reader to [9,26] for further details on the graph reduction procedure.

To allow a meaningful – although approximate – comparison on results obtained with different machines, a benchmark program (dfmax), together with a benchmark instance (r500.5), are available at http://mat.gsia.cmu.edu/COLOR03/. Computing times obtained on different machines can be scaled with respect to the performance obtained on this program. All results of our algorithms were obtained on a Pentium IV 2.4 GHz with 1 GB RAM under Windows XP, which spent 7 s *user time* to solve the benchmark instance. We used CPLEX10.1 to solve the LP/ILP problems.

The computational experiments are organized as follows: in Section 3.1 we report the results corresponding to the root node of the Branch-and-Price algorithm. These include the results of the MMT algorithm. In Section 3.2 we investigate the performance of the Branch-and-Price algorithm for those instances not solved to proven optimality at the root node.

3.1. Root node

In this section we report the results obtained by the MMT algorithm [22] when integrated with the column generation method. The former produces very good solutions and a large number of columns, which are used to initialize the latter (see Section 2). The latter provides a strong lower bound which can prove the optimality of many solutions produced by MMT.

For the instances studied in [22], namely DSJC, DSJR, latin_square_10, we keep the same set up of the MMT algorithm, which includes a time limit for the Evolutionary algorithm (first phase) depending on the size of the instance and a time limit for the second phase of 100 s. Concerning the instances not considered in [22], we use a time limit of 80 s for the queen instances from queen13_13 to queen16_16 and of 150 s for the wap instances for the Evolutionary phase, and of 20 s for the second phase. For the remaining instances, which are easier, we use a time limit of 2 s for the Evolutionary algorithm and of 1 s for the second phase.

The first four columns of Table 1 report, for every instance, the name, the number of vertices n, the number of edges m and the chromatic number χ (when known, by construction or by computational proof). In the following columns we report the lower bound computed by MMT, which corresponds to the cardinality $\omega(G)$ of a maximal clique (column five), the solution value sol_{MMT} obtained by MMT (column six), and the corresponding computing time in seconds T_{MMT} (column seven). When the lower bound $\omega(G)$ can prove the optimality of the solution computed by MMT, the computation is stopped and the value of the optimal solution is reported in bold (e.g., for instance queen5_5). In the remaining columns of the table we report the results concerning the column generation procedure, which is executed with a time limit of 36 000 s: the best lower bound at the root node LB_{root} (column eight) and the corresponding computing time in seconds T_{root} (column nine). If the column generation bound allows us to prove the optimality of the incumbent solution, we report the bound value in bold in column height (e.g., for instance DSJC125.1). If the time limit is reached before convergence, column T_{root} reports tl while column LB_{root} reports the value of the bound LB_{MP} on the optimal value of the master problem if this improves the incumbent bound $\omega(G)$; when the time limit is not reached, LB_{root} equals LB_{SC}, i.e., the rounded-up value of the optimal solution of the master problem. The following columns of the table report the number of columns MMT_{cols} generated by the MMT algorithm which are included in the master problem (because they have negative reduced cost) (column ten), the number of columns $tabu_{cols}$ generated by the Tabu Search algorithm described in Section 2.2.1 (column eleven), the number of columns ILP_{cols} generated by the ILP solver as described in Section 2.2.2 (column twelve). At every iteration of the column generation procedure, the Tabu Search algorithm is executed with a maximum number of iterations equal to 1000. The execution is stopped as soon as a column with a reduced cost not greater than -0.1 is found (i.e., we set parameter $\beta = 1.1$). After 1000 iterations, if the best column returned by the Tabu Search algorithm has negative reduced cost, we add the column to the master problem and re-optimize it, otherwise we call again the Tabu Search algorithm, up to 1000 times, before tackling directly the slave problem with the ILP solver. Note that we can stop the column generation procedure before convergence when the lower bound LB_{MP} proves the optimality of the incumbent solution of MMT (see Section 2.2.2).

The MMT algorithm stand alone can solve to proven optimality 29 of the 115 considered instances. With the lower bound provided by the column generation procedure, the optimality of additional 33 instances is proved.

For the remaining instances, the column generation procedure provides a valid lower bound which improves the trivial bound obtained by a maximal clique computation in all but 20 cases, corresponding to very large instances. These difficult instances include the 3 DSJC instances with 1000 vertices, 6 wap instances with more than 1800 vertices, the ash958GPIA instance with 1916 vertices and the 3-Insertions_5 instance with 1406 vertices.

Finally, it should be noticed that instances DSJR500.1c, wap5 and wap6 are solved to proven optimality for the first time. The corresponding value of χ is thus reported in bold in column 4.

Concerning the difficulty of the instances, we did not observe any particular property of the graph affecting the performance of the MMT algorithm. For those graphs where the cardinality of the maximum clique ω equals the chromatic degree χ, often MMT can solve the problem to optimality without resorting to column generation. Concerning column generation applied to random graphs (DSJC), which are known to be extremely challenging, it clearly appears from the table that denser graphs are easier, and actually the root node is solved to optimality for instances with 125, 250 and 500 vertices and density equal to 0.9.

To investigate the behavior of column generation, in Figs. 2–4 we represent, for three representative instances, namely DSJC500.9 in Fig. 2, 4-FullIns_5 in Fig. 3 and abb313GPIA in Fig. 4, the values of the continuous relaxation of the master problem and of the lower bound LB_{MP} as functions of the computing time. In the figures, the upper line represents the value of the continuous relaxation, whose final optimal value is a lower bound for the problem, while an intermediate value (i.e., any value before convergence) does not provide a valid lower bound. The lower line represents the behavior of the lower bound LB_{MP}, which is always a valid lower bound. The computation is stopped as soon as the rounded-up values of the two functions are equal, because the value of the continuous relaxation cannot be smaller than LB_{MP}.

For instance DSJC500.9 (Fig. 2), all the columns for the optimal solution of the master problem are already present in the pool of columns provided by MMT. In order to prove the optimality of the solution, the slave problem is thus solved only once with the CPLEX10.1 ILP solver. The value of the continuous relaxation of the master problem initially drops from 127 (which is the value of the initial solution obtained when only the columns corresponding to the integer solution provided by MMT are available) to 123.29 and rapidly gets to a value of 122.31. The value of the lower bound LB_{MP}, which is updated as long as the CPLEX10.1 Branch-and-Bound code solves the slave problem, slowly grows from 35.89 to 122.04, when optimality is proven.

Table 1
MMT and column generation.

Instance	n	m	χ	$\omega(G)$	sol_{MMT}	T_{MMT}	LB_{root}	T_{root}	MMT_{cols}	$tabu_{cols}$	ILP_{cols}
DSJC125.1	125	736	5	3	5	21.0	**5**	121	198	1 130	6
DSJC125.5	125	3 891	?	9	17	121.8	16	247	819	160	5
DSJC125.9	125	6 961	?	28	44	120.9	43	20	317	2	0
DSJC250.1	250	3 218	?	4	8	20.4	6	*tl*	378	2 802	0
DSJC250.5	250	15 668	?	10	28	91.5	20	*tl*	2 308	439	8
DSJC250.9	250	27 897	?	34	72	96.0	71	446	1 262	2	0
DSJC500.1	500	12 458	?	4	12	213.1	5	*tl*	1 316	7 181	0
DSJC500.5	500	62 624	?	10	48	453.8	16	*tl*	8 715	1 239	0
DSJC500.9	500	112 437	?	44	127	3214.8	123	13 989	5 015	0	0
DSJC1000.1	1000	49 629	?	5	20	262.1	–	*tl*	2 852	5 542	0
DSJC1000.5	1000	249 826	?	13	84	9524.3	–	*tl*	63 035	3 065	0
DSJC1000.9	1000	449 449	?	51	224	8894.2	–	*tl*	32 872	0	0
DSJR500.1	500	3 555	12	11	12	24.3	**12**	11	1 556	0	0
DSJR500.1c	500	121 275	**85**	67	85	94.5	**85**	194	3 71	1	1
DSJR500.5	500	58 862	122	111	122	132.2	**122**	210	4 946	0	0
latin_sq._10	900	307 350	?	90	102	5568.2	90	683	23 610	0	0
le450_5a	450	5 714	5	5	**5**	0.3					
le450_5b	450	5 734	5	5	**5**	0.2					
le450_5c	450	9 803	5	5	**5**	0.1					
le450_5d	450	9 757	5	5	**5**	0.2					
le450_15a	450	8 168	15	15	**15**	0.1					
le450_15b	450	8 169	15	15	**15**	0.2					
le450_15c	450	16 680	15	15	**15**	3.1					
le450_15d	450	16 750	15	15	**15**	3.8					
le450_25a	450	8 260	25	25	**25**	0.1					
le450_25b	450	8 263	25	25	**25**	0.1					
le450_25c	450	17 343	25	25	**25**	1356.6					
le450_25d	450	17 425	25	25	**25**	66.6					
queen5_5	25	160	5	5	**5**	0.2					
queen6_6	36	290	7	6	7	3.0	**7**	3	167	0	0
queen7_7	49	476	7	7	**7**	0.2					
queen8_8	64	728	9	8	9	2.6	**9**	1	441	0	0
queen8_12	96	1 368	12	12	**12**	0.2					
queen9_9	81	1 056	10	9	10	2.6	9	3	974	63	0
queen10_10	100	2 940	11	10	11	2.9	10	3	1 543	83	0
queen11_11	121	3 960	11	11	12	2.7	11	10	1 861	66	0
queen12_12	144	5 192	12	12	13	3.9	12	12	2 193	48	0
queen13_13	169	6 656	13	13	14	83.8	13	22	2 167	47	0
queen14_14	196	8 372	14	14	15	136.	14	36	3 950	85	0
queen15_15	225	10 360	15	15	16	106.6	15	45	2 586	130	0
queen16_16	256	12 640	16	16	17	124.7	16	47	2 743	181	0
myciel3	11	23	4	2	4	3.7	3	1	15	0	0
myciel4	20	71	5	2	5	3.9	4	1	44	1	0
myciel5	47	236	6	2	6	3.9	4	1	95	20	0
myciel6	95	755	7	2	7	3.9	4	7	224	71	7
myciel7	191	2 360	8	2	8	3.9	5	254	327	60	106
1-Insertions_4	67	232	5	2	5	3.9	3	2	147	98	1
1-Insertions_5	202	1 227	?	2	6	3.9	3	53	798	1 275	52
1-Insertions_6	607	6 337	?	2	7	3.8	3	*tl*	2 234	11 915	1061
2-Insertions_4	149	541	?	2	5	3.9	3	22	360	987	4
2-Insertions_5	597	3 936	?	2	6	3.8	3	16 116	5 711	18 905	120
3-Insertions_3	56	110	4	2	4	3.6	3	1	98	76	0
3-Insertions_4	281	1 046	?	2	5	3.9	3	646	2 064	3 346	39
3-Insertions_5	1406	9 695	?	2	6	6.7	–	*tl*	5 251	5 828	0
4-Insertions_3	79	156	4	2	4	4.1	3	1	123	141	0
4-Insertions_4	475	1 795	?	2	5	3.9	3	14 492	5 722	12 612	100
1-FullIns_4	93	593	5	3	5	3.9	4	1	98	0	0
1-FullIns_5	282	3 247	6	3	6	3.9	4	3	158	30	1
2-FullIns_3	52	201	5	4	5	2.9	**5**	0	67	0	0
2-FullIns_4	212	1 621	6	3	6	3.7	5	1	167	2	0
2-FullIns_5	852	12 201	7	3	7	3.9	5	492	326	205	59
3-FullIns_3	80	346	6	5	6	2.9	**6**	0	68	0	0
3-FullIns_4	405	3 524	7	3	7	3.6	6	1	212	1	0
3-FullIns_5	2030	33 751	8	3	8	6.9	6	417	393	100	147
4-FullIns_3	114	541	7	2	7	3.4	**7**	0	69	0	0
4-FullIns_4	690	6 650	8	3	8	4.2	7	14	390	31	0
4-FullIns_5	4146	77 305	?	3	9	38.4	7	25 671	217	163	295
5-FullIns_3	154	792	8	7	8	3.6	**8**	1	73	0	0
5-FullIns_4	1085	11 395	?	3	9	6.9	8	47	255	9	5

(*continued on next page*)

Table 1 (*continued*)

Instance	n	m	χ	$\omega(G)$	sol_{MMT}	T_{MMT}	LB_{root}	T_{root}	MMT_{cols}	$tabu_{cols}$	ILP_{cols}
wap01	2368	110 871	?	40	43	286.4	–	*tl*	11 937	1 104	0
wap02	2464	111 742	?	40	42	260.1	–	*tl*	4 400	216	0
wap03	4730	286 722	?	40	47	354.0	–	*tl*	7 682	83	0
wap04	5231	294 902	?	40	44	329.0	–	*tl*	5 944	10	0
wap05	905	43 081	**50**	40	50	171.2	**50**	122	42 752	0	0
wap06	947	43 571	**40**	40	**40**	175.0					
wap07	1809	103 368	?	40	42	247.2	–	*tl*	7 768	1 271	0
wap08	1870	104 176	?	40	42	264.5	–	*tl*	4 347	457	0
qg_order30	900	26 100	30	30	**30**	0.2					
qg_order40	1600	62 400	40	40	**40**	2.9					
qg_order60	3600	212 400	60	60	**60**	3.8					
fpsol2_i_1	496	11 654	65	57	65	4.6	**65**	6	1 679	0	0
fpsol2_i_2	451	8 691	30	29	30	4.2	**30**	7	7 829	0	0
fpsol2_i_3	425	8 688	30	29	30	4.0	**30**	6	11 521	0	0
mug88_1	88	146	4	3	4	2.6	**4**	7	2 260	29	0
mug88_25	88	146	4	3	4	2.6	**4**	8	2 758	51	0
mug100_1	100	166	4	3	4	2.4	**4**	12	5 408	60	0
mug100_25	100	166	4	3	4	3.0	**4**	9	2 088	51	0
ash331GPIA	662	4 185	4	3	4	2.9	**4**	43	125	388	0
ash608GPIA	1216	7 844	4	3	4	5.8	**4**	2809	119	2 016	0
ash958GPIA	1916	12 506	4	3	4	5.6	–	*tl*	128	5 204	0
abb313GPIA	1557	46 546	?	6	9	7.9	8	11 517	547	3 807	687
will199GPIA	701	6 772	7	5	7	3.7	**7**	77	8 454	7	0
inithx.i.1	864	18 707	54	53	54	6.0	**54**	15	593	0	0
inithx.i.2	645	13 979	31	29	31	4.2	**31**	5	1 358	0	0
inithx.i.3	621	13 969	31	29	31	3.9	**31**	6	2 339	0	0
mulsol.i.1	197	3 925	49	49	**49**	0.2					
mulsol.i.2	188	3 885	31	30	31	3.7	**31**	1	8 880	0	0
mulsol.i.3	184	3 916	31	31	**31**	0.2					
mulsol.i.4	185	3 946	31	31	**31**	0.2					
mulsol.i.5	185	3 973	31	30	31	4.0	**31**	2	1 494	0	0
school1	385	19 095	14	14	**14**	0.4					
school1_nsh	352	14 612	14	13	14	3.0	**14**	14	1 036	680	0
zeroin.i.1	211	4 100	49	48	49	3.8	**49**	0	691	0	0
zeroin.i.2	211	3 541	30	28	30	3.4	**30**	1	6 847	0	0
zeroin.i.3	206	3 540	30	29	30	3.5	**30**	1	8 920	0	0
anna	138	493	11	8	11	3.6	**11**	0	42	0	0
david	87	406	11	11	**11**	0.2					
huck	74	301	11	11	**11**	0.2					
jean	80	254	10	10	**10**	0.2					
games120	120	638	9	9	**9**	0.2					
miles250	128	387	8	6	8	4.0	**8**	1	10 597	0	0
miles500	128	1 170	20	19	20	3.7	**20**	0	81	0	0
miles750	128	2 113	31	31	**31**	0.2					
miles1000	128	3 216	42	42	**42**	0.2					
miles1500	128	5 198	73	73	**73**	0.1					

For instance 4-FullIns_5 (Fig. 3), new columns are added to the master problem all along the computation. The value of the continuous relaxation of the master problem initially drops from 9.00 (initial solution corresponding to the columns of the integer MMT solution) to 7.72. In this phase many good columns are retrieved from the MMT pool and thus the decrease of the continuous relaxation value is very fast. Then, the value slowly decreases to a value of 6.49. The value of the lower bound LB_{MP}, which is updated during the column generation, slowly grows from 5.31 to 5.62, and after a long plateau suddenly jumps to 6.04 at the end of the computation, when optimality is proven.

Finally, also for instance abb313GPIA (Fig. 4), new columns are added to the master problem all along the computation. The value of the continuous relaxation of the master problem slowly decreases from 9 (initial solution corresponding to the columns of the integer MMT solution) to the final value of 8.00. The value of the lower bound LB_{MP}, which is updated during the column generation, slowly grows from 5.76 to 6.66, and after a plateau suddenly jumps to 8.00 at the end of the computation, when optimality is proven.

Although the behavior is different in the three cases, we observe that the value of the lower bound LB_{MP} is improved all along the computation: by stopping the computation before convergence (e.g., by early branching), one would obtain a poor lower bound, which would negatively affect the subsequent Branch-and-Bound algorithm. Preliminary computational experiments confirm that branching before convergence does not improve the performance of the algorithm and, at the same time, does not allow to detect at least a strong lower bound for the overall problem.

In the next section we discuss the results obtained by applying the Branch-and-Price algorithm to the 33 open instances for which the root node was solved within the time limit.

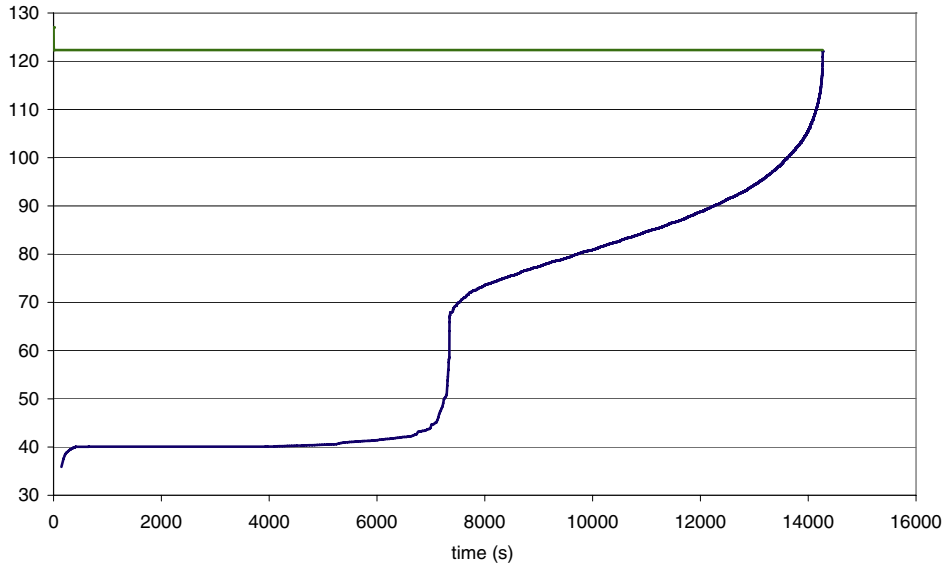

Fig. 2. Behavior of the column generation for instance DSJC500.9.

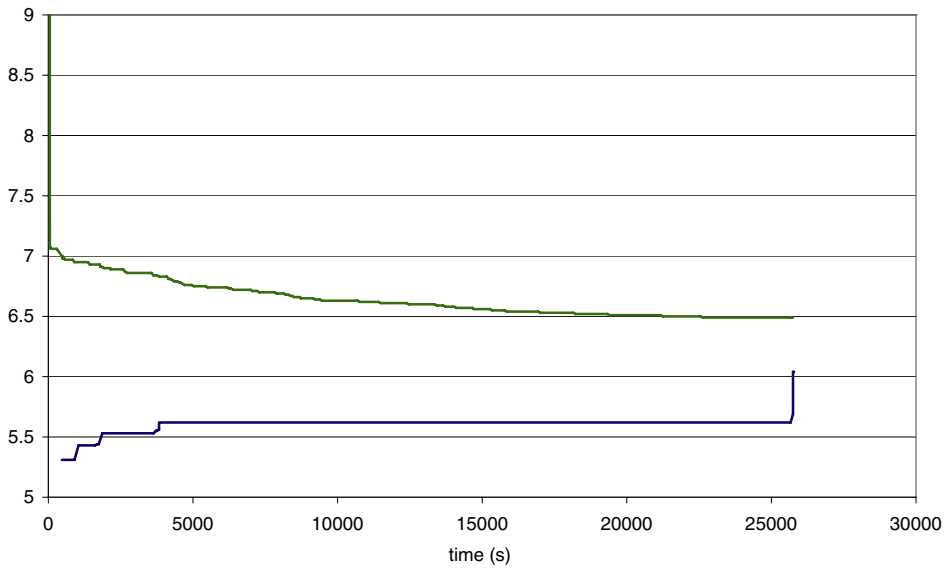

Fig. 3. Behavior of the column generation for instance 4-FullIns_5.

3.2. Branch-and-Price algorithm

In this section we compare the best performing version of the Branch-and-Price algorithm based on the edge branching with the best performing version based on the variable branching (according to our computational experience, in both cases the best version is the one accepting non-maximal columns in the master problem).

The time limit of the overall Branch-and-Price method is set to 36 000 s (this time includes the time for solving the root node but does not include the time spent by the MMT algorithm). In Table 2 we report the list of the instances for which the root node is solved within the time limit but the Branch-and-Price method exceeds the time limit (note that the value of the best incumbent solution is never improved).

In Table 3 we consider the instances which could be solved to optimality by using one of the branching methods. In the first 4 columns we report the instance characteristics (name, n, m, χ). The chromatic number χ is reported in bold for the 2 DSJC instances, which were never solved before to optimality. In the following, for each branching rule, we report the time needed to certify the optimality of the solution (note that the MMT solution is never improved) and the number of nodes of the Branch-and-Price tree.

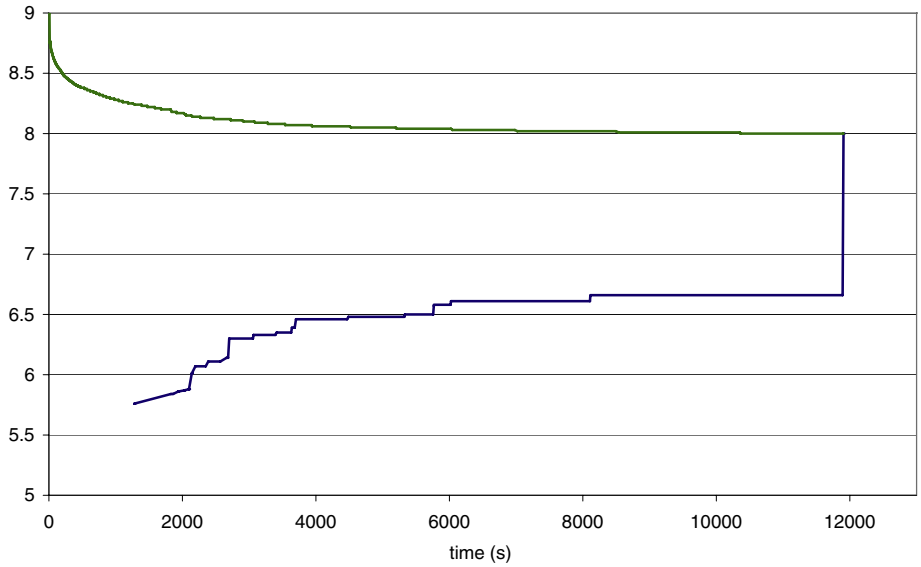

Fig. 4. Behavior of the column generation for instance abb313GPIA.

Table 2
Instances for which the root node is solved but the Branch-and-Price algorithm cannot improve on the best incumbent solution nor prove optimality.

DSJC250.9	queen15_15	1-Insertions_5	1-FullIns_5	4-FullIns_5
DSJC500.9	queen16_16	2-Insertions_4	2-FullIns_4	5-FullIns_4
latin_sq._10	myciel5	3-Insertions_3	2-FullIns_5	abb313GPIA
queen12_12	myciel6	3-Insertions_4	3-FullIns_4	
queen13_13	myciel7	4-Insertions_3	3-FullIns_5	
queen14_14	1-Insertions_4	1-FullIns_4	4-FullIns_4	

Table 3
Instances optimally solved by the Branch-and-Price algorithm.

Instance	n	m	χ	Variable branching		Edge branching	
				Nodes	Time	Nodes	Time
DSJC125.5	125	3891	**17**	281	17 929	374	*tl*
DSJC125.9	125	6961	**44**	17 634	*tl*	365	3776
queen9_9	81	1056	10	19	31	29	57
queen10_10	100	2940	11	137	681	163	944
queen11_11	121	3960	11	321	1863	4865	*tl*
myciel3	11	23	4	21	0	5	0
myciel4	20	71	5	7 957	118	573	4

The variable branching scheme is able to solve 6 out of the 7 considered instances, while the edge branching can solve only 5 out of the 7 instances. However, the small size of the test set does not allow us to conclude the superiority of one scheme over the other.

3.3. Comparison with the exact algorithms from the literature

In this section we compare the results obtained by the Branch-and-Price algorithm with the variable branching scheme[1] (called MMT-BP in the following), which integrates an effective metaheuristic procedure and a Branch-and-Price scheme, with the best performing exact algorithms from the literature for which extensive computational results exist, namely the Branch-and-Cut approach by Méndez-Díaz and Zabala [8] and the DSATUR algorithm (originally proposed by Brélatz in [24], then improved by Sewell [25], in the implementation reported in [10] available at http://mat.gsia.cmu.edu/COLOR/solvers/trick.c). Results for DSATUR are from [8]. In addition, we consider the lower bound computed by Méndez-Díaz and Zabala in [9]. We do not report the results of the LPCOLOR algorithm proposed by Mehrotra and Trick [10] since the authors only considered a subset of the instances, which were hard to solve 10 years ago, but can be solved quite easily nowadays, thanks to the evolution of the LP/ILP codes and of the computers, making a comparison with

[1] With the exception of instance DSJC125.9 which was solved with the edge branching scheme.

LPCOLOR not fair (these results are summarized in [11]). However, we mention that in [10] the authors report that their implementation of the DSATUR algorithm can solve the myciel5 instance.

Table 4 reports, in the first 2 columns, the name and the chromatic number χ of the instances. Then the table considers the results of the Branch-and-Cut algorithm BC-Col by Méndez-Díaz and Zabala [8], obtained on a Sun ULTRA 1 workstation at 140 MHz and 288 MB RAM. This machine needs 24 s to solve the benchmark instance (r500.5), thus we estimate our computer to be approximately 3 times faster for coloring problems. The authors, before starting the algorithm, preprocess the graph in order to reduce its size. In addition, they compute a maximal clique by means of a greedy algorithm and preassign a color to each vertex of the clique. Then they run code DSATUR [10] for 5 s, in order to obtain an upper bound. They consider this phase an initialization of the algorithm, which is then run with a time limit of 7200 s. The third and fourth columns of Table 4 report the best lower and upper bounds achieved within the time limit, the corresponding gap (column five), and the computing time in seconds, not including the initialization (column six). An *init* entry in column six means that the instance was solved to optimality during the initialization phase of the algorithm, thus with a computing time of at most 5 s. The following four columns report the same information for code DSATUR [10] in the tests performed by Méndez-Díaz and Zabala [8]. When one of the two algorithms can solve an instance while the other algorithm fails, they did not report the lower and upper bounds corresponding to the algorithm which failed, and thus we simply know that the gap is positive (>0).

Column eleven reports the lower bound obtained by Méndez-Díaz and Zabala in [9], where a cutting plane method is applied to three alternative formulations for the VCP, derived from the descriptive model VCP-ASS (2)–(6). We report the bound obtained by the best procedure, called $\mathcal{CP}$ in [9]. We consider only the subset of instances which were not solved during the initialization phase (greedy computation of a maximal clique and execution of DSATUR for 5 s), and for which procedure $\mathcal{CP}$ can improve the initial lower bound (represented by the cardinality of a maximal clique). The computing time expressed in seconds on the previously mentioned machine is reported in column twelve.

Finally, the last seven columns summarize the results obtained by algorithm MMT-BP. We report the best lower and upper bounds achieved within the time limit (columns thirteen and fourteen), the corresponding gap (column fifteen), and the overall computing time in seconds (column sixteen, including the initialization phase performed by MMT). It is worth to note that our computer is approximately 3 times faster than the one used for the other algorithms and that we use a time limit of 36 000 s (in addition to the time spent by MMT and reported in Table 1). In order to give a performance comparison of the different methods with similar computing times, we give as well the results obtained by our algorithm when run for at most 2400 s (which roughly correspond to 7200 s of the slower computer used in [8,9]). Clearly, this is done only for the sake of comparison: if we have a sufficient computing time available, we should not stop an algorithm which is still improving the incumbent solution. Given a time budget of 2400 s, we stop the execution of the heuristic algorithm MMT after 300 s for those instances for which MMT exceeded this time (namely, instances DSJC500.5, DSJC500.9, DSJC1000.5, DSJC1000.9 and latin_square_10), and use the remaining 2100 s for the Branch-and-Price algorithm. In the last three columns of the table we report, only for the instances which were not solved within 2400 s, the corresponding results. The computing time is not reported explicitly, being equal to 2400 s for all the considered instances.

Comparing the performance of MMT-BP with the best among the methods considered in the table, we conclude that algorithm MMT-BP can solve to optimality 20 more instances (the optimal solution value for 5 of them, namely DSJC125.5, DSJC125.9, DSJR500.1c, wap05 and wap06 was not known in the literature), while there are only 3 instances for which one of the methods in the table can prove optimality of the solution while algorithm MMT-BP fails. Concerning the gap between the lower and upper bounds, algorithm MMT-BP improves on the best known result in 43 cases and obtains a worse result in 9 cases. The improvement is especially due to the good quality of the upper bound, which improves the best one from the literature (by considering the exact algorithms) in 42 cases and is never worse. Concerning the value of the lower bound, algorithm MMT-BP improves on the best known result in 12 cases and obtains a worse result in 13 cases. When the same comparison is done with respect to the performance of MMT-BP with a time limit of 2400 s, we observe the following: MMT-BP can solve to optimality 17 more instances, and there are 3 instances for which one of the other methods can prove optimality of the solution while algorithm MMT-BP fails. Algorithm MMT-BP still improves on the best known upper–lower bound gap in 39 cases and obtains a worse result in 15 cases. The upper bound is still improved in 42 cases and it is never worse. Concerning the value of the lower bound, algorithm MMT-BP with shorter computing time improves on the best known result in 8 cases and obtains a worse result in 20 cases. These results confirm that, for hard instances, running MMT-BP for a long computing time can produce better results, in particular for what concerns the quality of the lower bounds.

Considering the different classes of instances in the table, the best results are obtained by algorithm MMT-BP for the random (DSJC), geometric random (DSJR), le, queen and wap instances, while algorithm MMT-BP gets into troubles for the Insertions and FullIns instances, for which the lower bound computed by the cutting plane methods tends to be better.

Finally, we conclude this computational section by pointing out that, by using the lower bound of value 4 provided by [8, 9], one can prove the optimality of the feasible solution of value 4 obtained by algorithm MMT-BP for instance ash958GPIA, which was previously unsolved.

4. Conclusions

In this paper we propose an exact approach for the Vertex Coloring Problem, one of the most studied NP-hard problems in combinatorial optimization. The interest for the problem is not only practical, due to the many applications which

Table 4
Comparison of exact algorithms for VCP.

Instance	χ	BC-Col [8]				DSATUR [8]				$\mathcal{CP}$ [9]		MMT-BP						
		LB	UB	%Gap	Time	LB	UB	%Gap	Time	LB	Time	LB	UB	%Gap	Time	LB	UB	%Gap
DSJC125.1	5	5	5	0.0	0.9	5	5	0.0	0.1	5	1	5	5	0.0	142.0			
DSJC125.5	17	13	20	35.0	*tl*	9	19	52.6	*tl*	12	77	17	17	0.0	18 050.8	16	17	5.9
DSJC125.9	44	42	47	10.6	*tl*	29	45	35.6	*tl*	42	354	44	44	0.0	3896.9	43	44	2.3
DSJC250.1	?	5	9	44.4	*tl*	4	9	55.6	*tl*	5	11	6	8	25.0	*tl*	5	8	37.5
DSJC250.5	?	13	36	63.9	*tl*	9	35	74.3	*tl*	14	3339	20	28	28.6	*tl*	15	28	46.4
DSJC250.9	?	48	88	45.5	*tl*	34	87	60.9	*tl*	48	3605	71	72	1.4	*tl*	71	72	1.4
DSJC500.1	?	5	15	66.7	*tl*	5	15	66.7	*tl*			5	12	58.3	*tl*	4	12	66.7
DSJC500.5	?	13	63	79.4	*tl*	9	63	85.7	*tl*	13	538	16	48	66.7	*tl*	11	48	77.1
DSJC500.9	?	59	161	63.4	*tl*	43	160	73.1	*tl*	59	5870	123	127	3.1	*tl*	44	127	65.4
DSJC1000.1	?	6	26	76.9	*tl*	5	25	80.0	*tl*			5	20	75.0	*tl*	5	20	75.0
DSJC1000.5	?	15	116	87.1	*tl*	10	114	91.2	*tl*			13	84	84.5	*tl*	13	92	85.9
DSJC1000.9	?	65	301	78.4	*tl*	53	300	82.3	*tl*	66	4546	51	224	77.2	*tl*	51	226	77.4
DSJR500.1	12	12	12	0.0	init	12	12	0.0	init			12	12	0.0	35.3			
DSJR500.1c	85	78	88	11.4	*tl*	70	88	20.5	*tl*	80	4470	85	85	0.0	288.5			
DSJR500.5	122	119	130	8.5	*tl*	103	130	20.8	*tl*	119	1211	122	122	0.0	342.2			
latin_sq._10	?	90	129	30.2	*tl*	90	129	30.2	*tl*			90	102	11.8	*tl*	90	108	16.7
le450_5a	5	5	9	44.4	*tl*	5	9	44.4	*tl*			5	5	0.0	0.3			
le450_5b	5	5	9	44.4	*tl*	5	9	44.4	*tl*			5	5	0.0	0.2			
le450_5c	5	5	5	0.0	init	5	5	0.0	init			5	5	0.0	0.1			
le450_5d	5	5	10	50.0	*tl*	5	8	37.5	*tl*			5	5	0.0	0.2			
le450_15a	15	15	17	11.8	*tl*	15	17	11.8	*tl*			15	15	0.0	0.4			
le450_15b	15	15	17	11.8	*tl*	15	16	6.3	*tl*			15	15	0.0	0.2			
le450_15c	15	15	24	37.5	*tl*	13	23	43.5	*tl*			15	15	0.0	3.1			
le450_15d	15	15	23	34.8	*tl*	13	23	43.5	*tl*			15	15	0.0	3.8			
le450_25a	25	25	25	0.0	init	25	25	0.0	init			25	25	0.0	0.1			
le450_25b	25	25	25	0.0	init	25	25	0.0	init			25	25	0.0	0.1			
le450_25c	25	25	28	10.7	*tl*	20	28	28.6	*tl*			25	25	0.0	1356.6			
le450_25d	25	25	28	10.7	*tl*	21	27	22.2	*tl*			25	25	0.0	66.6			
queen8_8	9	9	9	0.0	3.0	9	9	0.0	18.0			9	9	0.0	3.6			
queen8_12	12	12	12	0.0	init	12	12	0.0	init			12	12	0.0	0.2			
queen9_9	10	9	11	18.2	*tl*	9	10	10.0	*tl*			10	10	0.0	36.6			
queen10_10	11	10	12	16.7	*tl*	10	12	16.7	*tl*			11	11	0.0	686.9			
queen11_11	11	11	14	21.4	*tl*	11	13	15.4	*tl*			11	11	0.0	1865.7			
queen12_12	12	12	15	20.0	*tl*	12	14	14.3	*tl*			12	13	7.7	*tl*	12	13	7.7
queen13_13	13	13	16	18.8	*tl*	13	15	13.3	*tl*			13	14	7.1	*tl*	13	14	7.1
queen14_14	14	14	17	17.6	*tl*	14	17	17.6	*tl*			14	15	6.7	*tl*	14	15	6.7
queen15_15	15	15	18	16.7	*tl*	15	18	16.7	*tl*			15	16	6.3	*tl*	15	16	6.3
queen16_16	16	16	20	20.	*tl*	16	19	15.8	*tl*			16	17	5.9	*tl*	16	17	5.9
myciel6	7	5	7	28.6	*tl*	2	7	71.4	*tl*			4	7	42.9	*tl*	4	7	42.9
myciel7	8	5	8	37.5	*tl*	2	8	75.0	*tl*			5	8	37.5	*tl*	5	8	37.5
1-Insertions_4	5	5	5	0.0	2.0			>0	*tl*	3	2	3	5	40.0	*tl*	3	5	40.0
1-Insertions_5	?	4	6	33.3	*tl*	2	6	66.7	*tl*	3	0	3	6	50.0	*tl*	3	6	50.0
1-Insertions_6	?	4	7	42.9	*tl*	2	7	71.4	*tl*	3	3	3	7	57.1	*tl*	2	7	71.4
2-Insertions_4	?	4	5	20.0	*tl*	2	5	60.0	*tl*	3	0	3	5	40.0	*tl*	3	5	40.0
2-Insertions_5	?	3	6	50.0	*tl*	2	6	66.7	*tl*	3	3	3	6	50.0	*tl*	2	6	66.7
3-Insertions_3	4	4	4	0.0	1.0	4	4	0.0	5.0	3	0	3	4	25.0	*tl*	3	4	25.0
3-Insertions_4	?	3	5	40.0	*tl*	2	5	60.0	*tl*	3	0	3	5	40.0	*tl*	3	5	40.0
3-Insertions_5	?	3	6	50.0	*tl*	2	6	66.7	*tl*	3	61	2	6	66.7	*tl*	2	6	66.7
4-Insertions_3	4	3	4	25.0	*tl*	2	4	50.0	*tl*	3	0	3	4	25.0	*tl*	3	4	25.0
4-Insertions_4	?	3	5	40.0	*tl*	2	5	60.0	*tl*	3	2	3	5	40.0	*tl*	2	5	60.0
1-FullIns_4	5	5	5	0.0	0.1			>0	*tl*	4	0	4	5	20.0	*tl*	4	5	20.0
1-FullIns_5	6	4	6	33.3	*tl*	3	6	50.0	*tl*	4	0	4	6	33.3	*tl*	4	6	33.3
2-FullIns_3	5	5	5	0.0	0.1	5	5	0.0	1014.0	5	0	5	5	0.0	2.9			
2-FullIns_4	6	5	6	16.7	*tl*	4	6	33.3	*tl*	6	4	5	6	16.7	*tl*	5	6	16.7
2-FullIns_5	7	5	7	28.6	*tl*	4	7	42.9	*tl*			5	7	28.6	*tl*	5	7	28.6
3-FullIns_3	6	6	6	0.0	0.1			>0	*tl*	6	0	6	6	0.0	2.9			
3-FullIns_4	7	6	7	14.3	*tl*	5	7	28.6	*tl*	6	4	6	7	14.3	*tl*	6	7	14.3
3-FullIns_5	8	6	8	25.0	*tl*	5	8	37.5	*tl*	6	292	6	8	25.0	*tl*	5	8	37.5
4-FullIns_3	7	7	7	0.0	3			>0	*tl*	7	0	7	7	0.0	3.4			
4-FullIns_4	8	7	8	12.5	*tl*	6	8	25.0	*tl*	7	16	7	8	12.5	*tl*	7	8	12.5
4-FullIns_5	?	6	9	33.3	*tl*	6	9	33.3	*tl*			7	9	22.2	*tl*	6	9	33.3
5-FullIns_3	8	8	8	0.0	20			>0	*tl*			8	8	0.0	4.6			
5-FullIns_4	?	8	9	11.1	*tl*	7	9	22.2	*tl*	8	55	8	9	11.1	*tl*	8	9	11.1
wap01	?	41	46	10.9	*tl*	39	48	18.8	*tl*			40	43	7.0	*tl*	40	43	7.0
wap02	?	40	45	11.1	*tl*	39	46	15.2	*tl*			40	42	4.8	*tl*	40	42	4.8
wap03	?	40	56	28.6	*tl*	40	55	27.3	*tl*			40	47	14.9	*tl*	40	47	14.9
wap04	?	40	50	20.0	*tl*	20	48	58.3	*tl*			40	44	9.1	*tl*	40	44	9.1

Table 4 (*continued*)

Instance	χ	BC-Col [8]				DSATUR [8]				$\mathcal{CP}$ [9]		MMT-BP						
		LB	UB	%Gap	Time	LB	UB	%Gap	Time	LB	Time	LB	UB	%Gap	Time	LB	UB	%Gap
wap05	50	50	51	2.0	*tl*	27	51	47.1	*tl*			50	50	0.0	293.2			
wap06	40	40	44	9.1	*tl*	33	45	26.7	*tl*			40	40	0.0	175.0			
wap07	?	40	46	13.0	*tl*	23	46	50.0	*tl*			40	42	4.8	*tl*	40	42	4.8
wap08	?	40	47	14.9	*tl*	23	45	48.9	*tl*			40	42	4.8	*tl*	40	42	4.8
qg_order30	30	30	30	0.0	init	30	30	0.0	init			30	30	0.0	0.2			
qg_order40	40	40	42	4.8	*tl*	40	42	4.8	*tl*			40	40	0.0	2.9			
qg_order60	60	60	63	4.8	*tl*	60	63	4.8	*tl*			60	60	0.0	3.8			
fpsol2_i_1	65	65	65	0.0	0.6	65	65	0.0	0.1	65	8	65	65	0.0	10.6			
fpsol2_i_2	30	30	30	0.0	1.2	30	30	0.0	0.1	30	1	30	30	0.0	11.2			
fpsol2_i_3	30	30	30	0.0	1.1	30	30	0.0	0.1	30	1	30	30	0.0	10.0			
mug88_1	4	4	4	0.0	11.0			>0	*tl*			4	4	0.0	9.6			
mug88_25	4	4	4	0.0	184.0	4	4	0.0	4756.0			4	4	0.0	10.6			
mug100_1	4	4	4	0.0	60.0			>0	*tl*			4	4	0.0	14.4			
mug100_25	4	4	4	0.0	60.0			>0	*tl*			4	4	0.0	12.0			
ash331GPIA	4	4	4	0.0	51.0	4	4	0.0	0.7	4	46	4	4	0.0	45.9			
ash608GPIA	4	4	4	0.0	692.0	4	4	0.0	3.0	4	692	4	4	0.0	2814.8	3	4	25.0
ash958GPIA	4	4	5	20.0	*tl*	3	5	40.0	*tl*	4	4236	3	4	25.0	*tl*	3	4	25.0
abb313GPIA	?	8	10	20.0	*tl*	6	10	40.0	*tl*			8	9	11.1	*tl*	7	9	22.2
will199GPIA	7			>0.0	*tl*	7	7	0.0	1.2			7	7	0.0	80.7			
inithx.i.1	54	54	54	0.0	init	54	54	0.0	init			54	54	0.0	21.0			
inithx.i.2	31	31	31	0.0	init	31	31	0.0	init			31	31	0.0	9.2			
inithx.i.3	31	31	31	0.0	init	31	31	0.0	init			31	31	0.0	9.9			
mulsol.i.1	49	49	49	0.0	init	49	49	0.0	init			49	49	0.0	0.2			
mulsol.i.2	31	31	31	0.0	init	31	31	0.0	init			31	31	0.0	4.7			
mulsol.i.3	31	31	31	0.0	init	31	31	0.0	init			31	31	0.0	0.2			
mulsol.i.4	31	31	31	0.0	init	31	31	0.0	init			31	31	0.0	0.2			
mulsol.i.5	31	31	31	0.0	init	31	31	0.0	init			31	31	0.0	6.0			
school1	14	14	14	0.0	init	14	14	0.0	init			14	14	0.0	0.4			
school1_nsh	14	14	14	0.0	init	14	14	0.0	init			14	14	0.0	17.0			
zeroin.i.1	49	49	49	0.0	init	49	49	0.0	init			49	49	0.0	3.8			
zeroin.i.2	30	30	30	0.0	init	30	30	0.0	init			30	30	0.0	4.4			
zeroin.i.3	30	30	30	0.0	init	30	30	0.0	init			30	30	0.0	4.5			
anna	11	11	11	0.0	init	11	11	0.0	init			11	11	0.0	3.6			
david	11	11	11	0.0	init	11	11	0.0	init			11	11	0.0	0.2			
huck	11	11	11	0.0	init	11	11	0.0	init			11	11	0.0	0.2			
jean	10	10	10	0.0	init	10	10	0.0	init			10	10	0.0	0.2			
games120	9	9	9	0.0	init	9	9	0.0	init			9	9	0.0	0.2			
miles250	8	8	8	0.0	init	8	8	0.0	init			8	8	0.0	5.0			
miles500	20	20	20	0.0	init	20	20	0.0	init			20	20	0.0	3.7			
miles750	31	31	31	0.0	init	31	31	0.0	init			31	31	0.0	0.2			
miles1000	42	42	42	0.0	0.2	42	42	0.0	0.1	42	0	42	42	0.0	0.2			
miles1500	73	73	73	0.0	0.1	73	73	0.0	0.1	73	0	73	73	0.0	0.1			

can be modelled as coloring problems, but also theoretical and related to the intrinsic difficulty of the problem from the computational point of view.

The proposed approach considers the Set Covering formulation of the problem and is based on a column generation procedure and a Branch-and-Price scheme. Our contribution includes an original Tabu Search algorithm for the solution of the slave problem arising in the column generation procedure, an alternative branching scheme which is compared with a classical one from the literature, and an efficient integration of the Branch-and-Price method with a metaheuristic algorithm from the literature, which produces high quality solutions and a large set of columns which are used to speed-up the column generation procedure.

The approach is computationally tested on a large set of DIMACS instances, representing the standard test-bed for evaluating the performance of coloring algorithms. The experiments show the effectiveness of the proposed algorithm, which is able to solve, for the first time to proven optimality, five of the benchmark instances of the literature, and to reduce the optimality gap of many others.

Acknowledgements

The authors are grateful to two anonymous referees for their careful reading and useful comments.

References

[1] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, 1979.
[2] F.T. Leighton, A graph coloring algorithm for large scheduling problems, Journal of Research of the National Bureau of Standards 84 (1979) 489–503.
[3] N. Zufferey, P. Amstutz, P. Giaccari, Graph colouring approaches for a satellite range scheduling problem, Journal of Scheduling 11 (2008) 263–277.

[4] D. de Werra, An introduction to timetabling, European Journal of Operational Research 19 (1985) 151–162.
[5] F.C. Chow, J.L. Hennessy, The priority-based coloring approach to register allocation, ACM Transactions on Programming Languages and Systems 12 (1990) 501–536.
[6] A. Gamst, Some lower bounds for a class of frequency assignment problems, IEEE Transactions on Vehicular Technology 35 (1986) 8–14.
[7] T.K. Woo, S.Y.W. Su, R. Newman Wolfe, Resource allocation in a dynamically partitionable bus network using a graph coloring algorithm, IEEE Transactions on Communications 39 (2002) 1794–1801.
[8] I. Méndez-Díaz, P. Zabala, A branch-and-cut algorithm for graph coloring, Discrete Applied Mathematics 154 (2006) 826–847.
[9] I. Méndez-Díaz, P. Zabala, A cuttting plane algorithm for graph coloring, Discrete Applied Mathematics 156 (2008) 159–179.
[10] A. Mehrotra, M.A. Trick, A column generation approach for graph coloring, INFORMS Journal on Computing 8 (1996) 344–354.
[11] E. Malaguti, P. Toth, A survey on vertex coloring problems, International Transactions in Operational Research 17 (2010) 1–34.
[12] F. Hermann, A. Hertz, Finding the chromatic number by means of critical graphs, ACM Journal of Experimental Algorithmics 7 (2002) 1–9.
[13] A. Hertz, D. de Werra, Using tabu search techniques for graph coloring, Computing 39 (1987) 345–351.
[14] D.S. Johnson, C.R. Aragon, L.A. McGeoch, C. Schevon, Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning, Operations Research 39 (1991) 378–406.
[15] L. Davis, Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1998.
[16] C.A. Morgenstern, Distibuted coloration neighborhood search, in: D.S. Johnson, M.A. Trick (Eds.), Cliques, Coloring, and Satisfiability: 2nd DIMACS Implementation Challange, 1993, in: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1996, pp. 335–358.
[17] P. Galinier, J.K. Hao, Hybrid evolutionary algorithms for graph coloring, Journal of Combinatorial Optimization 3 (1999) 379–397.
[18] N. Funabiki, T. Higashino, A minimal-state processing search algorithm for graph coloring problems, IEICE Transactions on Fundamentals E83-A (2000) 1420–1430.
[19] P. Galinier, A. Hertz, N. Zufferey, An adaptative memory algorithm for the *k*-coloring problem, Discrete Applied Mathematics 156 (2008) 267–279.
[20] I. Blöchliger, N. Zufferey, A reactive tabu search using partial solutions for the graph coloring problem, Computers & Operations Research 35 (2008) 960–975.
[21] M. Plumettaz, D. Schindl, N. Zufferey, Ant local search and its efficient adaptation to graph colouring, Journal of the Operational Research Society (2010) (in press).
[22] E. Malaguti, M. Monaci, P. Toth, A metaheuristic approach for the vertex coloring problem, INFORMS Journal on Computing 20 (2008) 302–316.
[23] R. Brown, Chromatic scheduling and the chromatic number problem, Management Science 19 (1972) 456–463.
[24] D. Brélatz, New methods to color the vertices of a graph, Communications of the ACM 22 (1979) 251–256.
[25] E.C. Sewell, An improved algorithm for exact graph coloring, in: D.S. Johnson, M.A. Trick (Eds.), Cliques, Coloring, and Satisfiability: 2nd DIMACS Implementation Challange, 1993, in: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1996, pp. 359–373.
[26] P. Hansen, M. Labbé, D. Schindl, Set covering and packing formulations of graph coloring: algorithms and first polyhedral results, Discrete Optimization 6 (2009) 135–147.
[27] A. Caprara, M. Fischetti, P. Toth, A heuristic method for the set covering problem, Operations Research 47 (1999) 730–743.
[28] P.R.J. Östergård, A new algorithm for the maximum-weight clique problem, Nordic Journal of Computing 8 (2001) 424–436.
[29] I.M. Bomze, M. Budinich, P.M. Pardalos, M. Pelillo, The maximum clique problem, in: D.Z. Du, P.M. Pardalos (Eds.), Handbook of Combinatorial Optimization, suppl. vol. A, 1999, pp. 1–74.
[30] A. Grosso, M. Locatelli, W. Pullan, Simple ingredients leading to very efficient heuristics for the maximum clique problem, Journal of Heuristics 14 (2008) 587–612.
[31] A.A. Farley, A note on bounding a class of linear programming problems, including cutting stock problems, Operations Research 38 (1990) 922–923.
[32] J.M. Valério de Carvalho, Using extra dual cuts to accelerate column generation, INFORMS Journal on Computing 17 (2005) 175–182.
[33] A.A. Zykov, On some properties of linear complexes, Matematicheskij Sbornik 24 (1949) 163–188.
[34] D.S. Johnson, M.A. Trick (Eds.), Cliques, Coloring, and Satisfiability: 2nd DIMACS Implementation Challange, 1993, in: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1996.