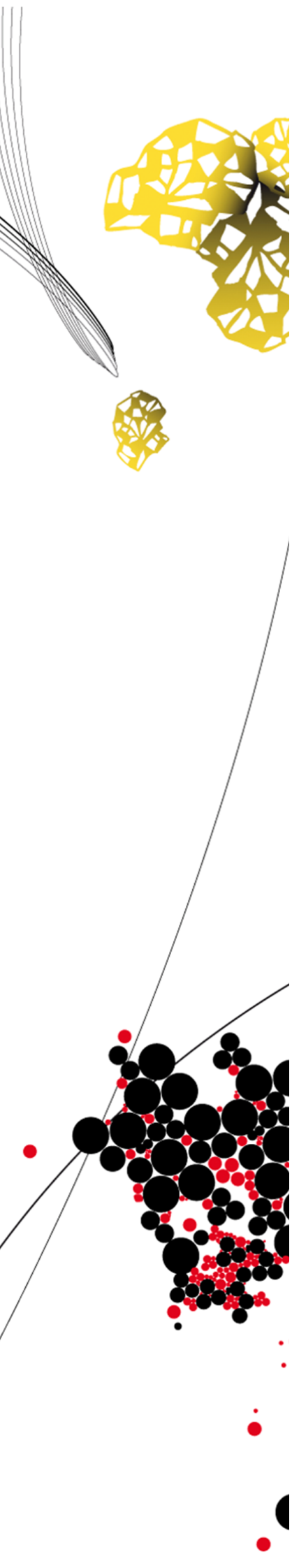


UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics & Computer Science

A branch-price-and-cut algorithm for graph coloring

Roelof Petrus van der Hulst

M.Sc. Thesis
August 2021

Supervisor:
dr. M. Walter

Graduation committee:
prof. dr. M.J.Uetz
dr. M. Walter
dr. K. Proksch

Discrete Mathematics and
Mathematical Programming
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Abstract

Branch-and-price approaches for graph coloring using the set covering formulation have been shown to be effective, particularly in computing strong lower bounds. This thesis proposes the addition of both maximally-violated mod-k cutting planes and odd-cycle cutting planes to extend the existing branch-and-price algorithms. Algorithms to efficiently separate these classes of cutting planes are proposed. A combinatorial algorithm for the modified pricing algorithm is proposed, and the resulting branch-price-and-cut algorithm is tested on the DIMACS instances. The branch-price-and-cut algorithm has similar performance as the branch-and-price algorithm when applying cuts at the root node. Maximally violated mod-k cutting planes are shown to be effective in increasing the lower bound of the root node for *insertion* graphs, and odd-cycle cuts are shown to be effective in increasing the lower bound of dense DIMACS instances.

Various branching strategies from the literature are also surveyed, and a new branching strategy is proposed which outperforms the existing branching strategies. Using this new branching strategy, we are able to solve all tested graphs with less than 100 vertices, solving the *myciel6* instance, all *FullIns* and 3 *insertion* graphs using a branch-and-price algorithm for the first time. The open instance *5-FullIns_4* is solved by the new branch-and-price algorithm.

Preface

This thesis was written in 2021 during the COVID pandemic. This meant there were little opportunities to meet in person, and that I often had to work in my student room. While this was not how I initially envisioned my thesis time, I thoroughly enjoyed working on this thesis and I am proud of it. I would like to thank a few people who have helped make that possible. First and foremost, I would like to thank Matthias Walter for his feedback, enthusiasm and guidance in the past year, not only for this project, but also for supervising my internship at DAT.mobility. I would also like to thank my family, my friends, my roommates at Huize Orient, and Judith, who have all supported me throughout my study and with this thesis.

Contents

Chapter 1

Introduction and related works

GRAPH COLORING, also known as VERTEX COLORING, is a well known problem in Graph Theory. Given a graph $G = (V, E)$, a color $c(v)$ is assigned to each vertex $v \in V$, such that no two adjacent vertices connected by an edge are assigned the same color. More formally, a *valid coloring* of a graph G is a coloring for all $v \in V$ such that $c(u) \neq c(v), \forall (u, v) \in E$. In graph coloring, the objective is to use a minimal amount of colors. The minimum number of colors of a graph is denoted by $\chi(G)$, and is also called the chromatic number of G. Graph coloring is well known to be NP-hard, as shown by Karp [39].

Graph coloring is a fundamental NP-hard problem with many applications. Discrete scheduling problems can often be posed as graph coloring problems[25, 74]. For example, let each vertex represent an event (e.g. a school class), with edges connecting two vertices if they are in conflict (e.g. they require the same teacher). Then in a valid coloring of the graph, all events with the same color can happen simultaneously, and a minimum color solution uses a minimal number of time-slots. Several applications regarding timetabling exist [9, 69]. Similarly, graph coloring is used as a model for the register allocation problem [16]. Applications also exist for frequency assignment [62] and resource allocation problems [71].

Another good reason to investigate the graph coloring problem is that it seems to be one of the most challenging NP-hard problems. Even for graphs with as few as 95 vertices, research has been struggled to prove the exact value of $\chi(G)$ using state-of-the-art algorithms [45]. In this light, new techniques applicable to graph coloring could be worthwhile to use in other problems and have a broader impact.

1.1 Related work

The literature on Graph Coloring is extensive. It can be separated in two different categories, exact and heuristic approaches. Exact approaches attempt to find an optimal solution and guarantee its validity. Heuristic approaches also attempt to find solutions, but do not guarantee that the solutions they find are optimal. Typically, these heuristic approaches trade off the quality of a solution for a much faster running time.

There is a large amount of papers which heuristically try to find valid colorings. Most notable are tabu search and local search methods [6, 21, 37, 38, 54]. Two papers outside of this scope have obtained strong results; one extracts independent sets [33] and another uses simulated (quantum) annealing [64], although more research has been done into this direction [61]. For a more elaborate discussion and evaluation of the heuristic coloring literature, see [63]. The solutions found by these approaches pose valid upper bounds on the coloring number of a

given graph.

The literature on exact methods is less extensive. Three main types of formulations are commonly used in literature.

First are the 'classical' integer programming formulations. These typically assign a binary variable $x_{v,c}$ for each color c and vertex v. These formulations typically obtain weaker lower bounds and have problems with symmetry between color classes. Attempts to use the classical formulation typically revolve around breaking these symmetries by adding cutting planes or by strengthening the formulation to remove symmetries. In a branch-and-cut algorithm, Mendez-Diaz and Zabala [46, 47] add 5 different types of cutting planes to the classical formulation and introduce two ordering models which break symmetry. The asymmetric representatives formulation is another attempt to reduce symmetry by Campelo, Campos and Correa [10], and they also introduce several types of cutting planes. Conraz, Fugini and Malaguti [15] show that this formulation is also effective in practice, solving more instances than the cutting plane approach of Mendez-Diaz and Zabala [46].

A second class of methods tries to model the problem as a satisfiability (SAT) problem with boolean clauses. Van Gelder [65] introduces three formulations using propositional logic. Zhou et al. [73] also use propositional logic combined with a learning approach. Bouhmala and Granmo [7] use learning automata and random walks with a satisfiability encoding. In a more unique approach, Galinier, Hertz and Derosier [18] attempt to find χ-critical subgraphs in order to determine the chromatic number, which is effective for some classes of graphs. Hebrard and Katsirelos [35] use similar techniques using a SAT solver. Although finding χ-critical subgraphs can be useful for computing strong lower bounds, many of the more difficult tested instances were not χ-critical, making the application of this technique somewhat limited [18].

Last are branch-and-price algorithms. These formulations model each color as a stable set. Then, using a partitioning or covering problem formulation, they try to minimize the number of stable sets. Since the number of stable sets of a graph may be exponential in the size of the graph ($O(3^{\frac{n}{3}})$)[48], column generation is used in order to create a tractable problem. Columns are generated by the pricing algorithm. The strength of these formulations is primarily in their strong lower bounds. As variables are associated with stable sets, no two adjacent vertices can be given the same color in the LP relaxation of these formulations. This makes the linear relaxation of these formulations stronger than that of classical models, as they may allow for this.

The most researched branch-and-price algorithm uses the set covering formulation. This formulation uses only *maximal* stable sets. Mehrotra and Trick [45] first introduced a branch-and-price algorithm using the set covering formulation. Malaguti, Monaci and Toth [44] improve on these results by generating a large number of initial columns using a metaheuristic algorithm at the start of the column generation process. The used algorithms are adapted for numerical safety by Held et al. [36], who also propose a more complex and effective pricing algorithm. Gualandi and Malucelli [30] use constraint programming to solve the pricing problems, and introduce an augmented pricing method which attempts to find integral solutions for stronger upper bounds. The set covering formulation was also investigated by Hansen et al. [32], who attempt to add cutting plane procedures and introduce two preprocessing rules. Chapter 2 explains the set covering formulation in further detail.

The set partitioning formulation was also investigated by Hansen et al. [32]. They show it obtains results on par with the set covering formulation. They also attempted to add cutting plane procedures based on cliques in the *conflict graph* of stable sets, but results from these

experiments were not promising.

Various exact enumeration algorithms also exist. The DSATUR algorithm [45][8] in particular is widely used by many papers to compare the performance of linear programming or SAT formulations.

A common technique for integer programming is to add cutting planes based on polyhedral results of the problem formulation. In his thesis, Schindl introduces some polyhedral results for the set covering formulation of Graph Coloring [59]. Hansen, Labbé and Schindl [32] try to apply these results in practice, but do not have much success. They show that odd-cycle cuts can yield moderate improvements in the branch-and-bound tree size when added appropriately, emphasizing that further research may improve on these results. Additionally, they briefly report on the effect of adding cuts based on small graphs, which did not yield promising new results. To our knowledge, this is the only existing work of literature which investigates a branch-price-and-cut approach for graph coloring. Further possibilities for cutting planes are discussed in Chapter 3.

1.2 Research Goals and contribution

As many heuristic methods are already known, this research investigates exact integer linear programming formulations using column generation, the branch-and-price algorithms. These algorithms obtain strong lower bounds, and this way we may hope to prove that known upper bounds are indeed optimal. For this purpose, the set covering formulation is used, as it is most commonly used in literature. We then attempt to add cutting planes to the set covering formulation, in order to obtain stronger lower bounds.

Implementing cutting planes poses a challenge when combined with pricing. In particular, as the reduced cost of the variables changes, the structure of the pricing problem changes as well. These modifications can often increase the computational effort in the pricing problem. This research aims to resolve these issues by proposing and testing effective algorithms for pricing. This way, we hope to be able to efficiently add cutting planes in order to obtain stronger bounds for graph coloring problems using the set covering formulation. Furthermore, after adding these cutting planes, we aim to obtain a better understanding of the cutting planes' impact on the branch-price-and-cut algorithm.

1.3 Outline

In Chapter 2 the set covering formulation and its details are discussed. Chapter 3 discusses the methods necessary to separate odd-cycle and mod-k cutting planes. In Chapter 4 an algorithm for the modified pricing problem is formulated. Chapter 5 describes various differing branching strategies, and compares these on a small set of instances. Finally, we discuss our results and make recommendations for further research in Chapter 7.

Chapter 2

Set covering formulation

The set covering formulation was first described in detail by Mehrotra and Trick [45]. This section is simply an explanation of their work, as we use their work as the basis of this thesis. Most ideas in this section are based on [30, 32, 36] and [44].

2.1 Set covering formulation

In the set covering formulation (2.1), we use the variables x_S, where S is a stable set in the given graph $G = (V, E)$. A stable set is a set $S \subseteq V$ such that there are no edges between two vertices within S. More formally we have that for all $u, v \in S$, $(u, v) \notin E$. For a valid coloring, each color can be associated to a stable set.

Then, in the set covering formulation, $x_S = 1$ if and only if the stable set S represents a color class for the optimal coloring of the graph. Note that for a solution to (2.1) a vertex v is allowed to be in multiple color classes. We can always pick one of these colors to end up with a final solution. Then, the formulation can be restricted to only use inclusion-wise maximal stable sets, as any solution to (2.1) remains valid when we expand one of the color classes with a vertex. Let $\mathcal{S}_{\max}$ be the set of all maximal stable sets. The graph coloring problem can then be formulated as:

$$\chi(G) = \text{Minimize} \quad \sum_{S \in \mathcal{S}_{\max}} x_s \tag{2.1a}$$

$$\text{subject to} \quad \sum_{S \in \mathcal{S}_{\max} : v \in S} x_S \geq 1 \qquad \forall v \in V \tag{2.1b}$$

$$x_S \in \{0, 1\} \qquad \forall S \in \mathcal{S}_{\max} \tag{2.1c}$$

This is a valid formulation of the graph coloring problem [45].

In a branch-and-bound application, we seek to find lower bounds to (2.1). This is done by taking the *linear relaxation* of the integer programming formulation, relaxing the variables x_S to be able to take values in a continuous $[0, 1]$ interval. Then, we can use the Simplex algorithm (see [60] for more information) to solve the resulting Linear Program (LP) in practice. Let $\mathcal{S} \subseteq \mathcal{S}_{\max}$. Then the Restricted Master Problem (RMP) can be formulated as in the system (2.2).

$$\chi_f(G) = \text{Minimize} \quad \sum_{S \in \mathcal{S}} x_s \tag{2.2a}$$

$$\text{subject to} \quad \sum_{S \in \mathcal{S} | v \in S} x_S \geq 1 \qquad \forall v \in V \tag{2.2b}$$

$$x_S \in [0,1] \qquad \forall S \in \mathcal{S} \tag{2.2c}$$

Although we can try to solve the RMP using all variables so that $\mathcal{S} = \mathcal{S}_{\max}$, this is not feasible in general. The total number of variables $|\mathcal{S}_{\max}|$ can be exponential in the size of the graph with $O(3^{\frac{|V|}{3}})$ [48], which slows down the computation by too much. Instead, we restrict the columns to use the smaller subset $\mathcal{S}$.

After solving the RMP, we then have to consider if expanding our set of columns $\mathcal{S}$ would lead to a better solution for the RMP, as better columns may exist. This is done by using column generation. In column generation, the RMP is solved using linear programming. Then in the *pricing problem*, one decides if any columns exist which could possibly decrease $\chi_f(G)$, and add these columns if necessary. This is crucial as we otherwise cannot claim to have solved the RMP.

The pricing problem can be decided based on the reduced cost of the variables. By linear programming, if a variable x_S with $S \notin \mathcal{S}$ has negative reduced cost adding the variable to the RMP may improve the objective. Similarly, if x_S has zero or positive reduced cost, it cannot improve the objective of RMP. In the pricing problem, we then must find a negative reduced cost variable x_S, or prove that none exists. For the RMP (2.2), the pricing problem is a maximum weighted stable set problem.

Let π_v be the dual values in the current LP solution associated with constraints (2.2b). Then we can formulate the Maximum Weighted Stable Set problem (MWSS) as follows.

$$\text{Maximize} \quad W(y) = \sum_{v \in V} \pi_v y_v \tag{2.3a}$$

$$\text{subject to} \quad y_u + y_v \leq 1 \qquad \forall (u,v) \in E \tag{2.3b}$$

$$y_v \in \{0,1\} \qquad \forall v \in V \tag{2.3c}$$

Solving the system (2.3) gives us a solution vector y which is a stable set within the graph G. If for the objective (2.3a) we have that $W(y) > 1$, then we can conclude that we have found a stable set with negative reduced cost. We can then add this new stable set to $\mathcal{S}$ and solve the Restricted Master Problem again. If $W(y) \leq 1$, we can conclude that no negative reduced cost stable set exists, and thus we can then certify that the solution value as defined by the linear relaxation of the Restricted Master Problem is optimal. In column generation, we can then repeatedly add one or more columns to the Restricted Master Problem and solve it again, terminating when no more negative reduced columns exist.

The optimal solution value of the Restricted Master Problem is called the fractional chromatic number and denoted by $\chi_f(G)$. The fractional chromatic number is a lower bound to the Master Problem [60], so that we have $\chi_f(G) \leq \chi(G)$. In this report we also use the notation $\chi_c(G)$, that we define as the optimal solution value of RMP at the root branch-and-bound node after adding cutting planes. Intermediate solutions to the RMP that are not yet proven optimal by the pricing problem are denoted by z_{RMP}. Note that z_{RMP} is not necessarily equal to $\chi_f(G)$, as reduced cost columns may be added to the Reduced Master Problem still. Generic lower bounds for $\chi(G)$ are denoted as $\underline{\chi}(G)$.

The strength of the set covering formulation lies primarily in its strong lower bounds. The explanation for this is that using stable sets as variables implies that each clique in G is colored with the exact amount of colors needed, also within the RMP. This corresponds to adding all clique inequalities to 'classical' formulations, which is one of the most important classes of cutting planes used for these formulations [47].

For each pricing iteration the minimal reduced negative cost c^* and an associated stable set vector y^* can be found by solving the pricing problem to optimality. Then, one can define a lower bound using the optimal value from the maximum weighted stable set problem (see [43, 44]) together with z_{RMP}. Let us denote this lower bound by $\underline{\chi_p}$.

$$\underline{\chi_p} = \frac{z_{RMP}}{1 - c^*} = \frac{z_{RMP}}{W(y^*)} \tag{2.4}$$

Note that bound (2.4) can only be used when the pricing problem is solved to optimality or when a valid upper bound $\overline{W}$ such that $\overline{W} \geq W(y^*))$ is found.

An important note is that the Maximum Weight Stable Set problem is equivalent to the Maximum Weight Clique problem by simply taking the complement of the edges $\bar{E} = \{(u, v) \notin E \mid u, v \in V\}$, and by then considering the maximum weight clique problem on the graph $\bar{G} = (V, \bar{E})$. Both problems are among the most studied NP-hard problems. In Chapter 4, we will further discuss the pricing problem and algorithms used to solve it.

Although we may solve the linear relaxation of the Restricted Master Problem, a large gap between $\chi_f(G)$ and $\chi(G)$ may still exist. This problem is usually solved using a branch-and-bound algorithm. In branch-and-bound, the problem is recursively split up into two easier subproblems, creating a branch-and-bound tree. Any integral solution found in a branch-and-bound node is valid for the original problem and thus constitutes a good upper bound. The lower bound of a branch-and-bound node is the minimum of the lower bound of its two children. This lower bound can be computed by solving the Restricted Master Problem again for this branch-and-bound node. Some of the algorithmic details on the branch-and-bound algorithm are described in Section 2.1.2.

2.1.1 Preprocessing

In the literature, two preprocessing rules specific to graph coloring have been introduced by Hansen et al. [32]. These preprocessing rules aim to reduce the size of the graph to be colored. We use the notation $G[V'] = (V', E')$ to denote the induced graph of the vertices V', with $E' = \{(u, v) \in E | u, v \in V'\}$. Both preprocessing rules are based on expanding valid colorings of subgraphs. Given some vertex $v \in V$ and a valid coloring for $G[V \setminus \{v\}]$, we then argue that a color is available for v or that we can assign a color greedily to v. Then, we can remove this vertex from the relevant coloring problem, reducing the size of the program.

The first rule introduced by Hansen et al. [32] is based on *dominated* vertices. Given two distinct vertices $u, v \in V, u \neq v$, it is said that u dominates v if and only if $N(v) \subseteq N(u)$. Note this also implies $(u, v) \notin E$. Then given any coloring for $G[V \setminus \{v\}]$, we can always assign the color of u to v. This will be valid, as in a valid coloring for $G[V \setminus \{v\}]$ all vertices in $N(u)$ must receive a different color as u, which implies all colors in $N(v)$ are different from the color of u. Thus, when removing any dominated vertex v from a graph G, we know that

$\chi(G) = \chi(G[V \setminus \{v\}])$.

The second preprocessing rule introduced by Hansen et al.[32] uses knowledge of a lower bound $\underline{\chi}(G)$ on $\chi(G)$. When considering a vertex v with degree $d(v) < \underline{\chi}(G) - 1$ with a given optimal coloring for $G[V \setminus \{v\}]$, we then know that always at least one color in the set of colors $\{1, 2, ..., \underline{\chi}(G)\}$ must be available for it. If a given subgraph G' is known with $\chi(G') = \underline{\chi}(G)$, then we can also safely remove vertices $v \notin G'$ with $d(v) = \underline{\chi}(G) - 1$. For example, if a graph has a single large clique of size n, K_n, all vertices $v \in K_n$ have $d(v) = n - 1$, but removing a vertex from K_n would delete a vertex from the clique, lowering the coloring number. Thus, in this case, we can only remove a vertex v with degree $n - 1$ if $v \in V \setminus K_n(V)$, as $\chi(K_n) = n$ implies that $\chi(G)$ would still be unaltered from using the greedy argument.

A lower bound $\underline{\chi}(G)$ can be computed by calculating the size of the maximal clique in G. Lower bounds also naturally arise when solving the pricing problem to optimality, as given by equation (2.4), particularly when a branch-and-bound node is solved to optimality.

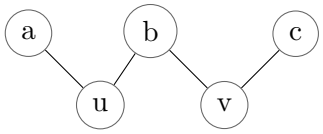
Then, given an original graph G, we can reduce it to a graph G' by iteratively removing all vertices which satisfy one of the above rules until no more vertex can be removed. As this can be done in polynomial time, it is well worth to apply this reduction to a graph G. Algorithm 1 illustrates the algorithm used.

Algorithm 1 Preprocessing algorithm

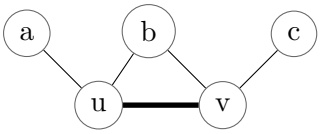
```
1: function PREPROCESSING(G,χ(G), (optional) clique K such that χ(K) = χ(G))
2:     G' = G
3:     loop
4:         for v ∈ V(G') do
5:             if d(v) < χ(G) − 1 or (d(v) = χ(G) − 1 and v ∉ K_n) then
6:                 G' = G'[V \ {v}]
7:             end if
8:         end for
9:         for u, v ∈ V(G') do
10:            if d(u) ≤ d(v) and N(u) ⊆ N(v) then
11:                G' = G'[V \ {v}]
12:            end if
13:        end for
14:        if No vertex was removed then return G'
15:        end if
16:    end loop
17: end function
```

2.1.2 Branching rule

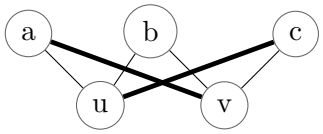
In a branch-and-bound algorithm, we need a procedure to separate the problem into two subproblems. The branching rule used by Trick and Mehrotra [45], which was first proposed by Zykov [75], is the most used branching rule for the set covering formulation. It is equivalent to the well known Ryan/Foster branching rule for graph coloring [58]. Mehrotra and Trick make the observation that traditional branching rules are not very suitable. These branching rules typically split up the problem into two subproblems by fixing a single variable x_S to 0 and 1 in the two subproblems respectively. However, this can be problematic as it implies stable sets must be forbidden in one of the branches, which alters the pricing problem (2.3) (see [44]).



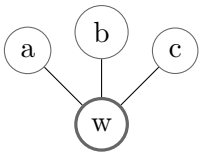
(a) The original graph



(b) The original graph with DIFFER branching on vertices u and v



(c) The original graph with a SAME constraint on vertices u and v, without removing a vertex



(d) The original graph with SAME branching on vertices u and v, contracting them to a new vertex w.

Figure 2.1: An example of Zykov's branching rule for a simple graph. Changed edges or vertices are marked in bold.

Instead, one can use a branching approach specific to graph coloring that does not alter the problem structure of the pricing problem.

Consider two vertices u, v such that $u, v \notin E$. Then we can create two subproblems. In the first subproblem we require that u and v are given different colors. This can be done by adding the edge (u, v), e.g. setting $G_{DIFFER} = (V, E \cup (u, v))$. In the second subproblem, we require that u and v are assigned the same color. We can do this by *contracting* the vertices u and v. Here, we remove u and v from the graph, and replace them with a new vertex w which has $N(w) = N(u) \cup N(v)$. We denote the resulting graph as G_{SAME}.

The SAME constraint is sometimes implemented differently. Instead of unifying the two vertices, edges are added to the vertices in $N(u)$ and $N(v)$ such that $N(u) = N(v)$. Then, implicitly we could remove one of u or v since it is dominated, but for other reasons it may be beneficial to keep the relevant vertex in the graph. Suppose we are in a branch of the branch-and-bound tree with a $SAME(u, v)$ constraint. Then, if at a later point in the branch-and-bound tree edges an edge (u, w) is added for some vertex $w \in V$, we also need to add the edge (v, w) if it does not yet exist, in order to ensure that the SAME constraint is still implicitly satisfied by the graph, e.g. that $N(u) = N(v)$ still. Figure 2.1 shows an example of the branching rule and the two different SAME implementations for a simple case.

In order to select the relevant two vertices to branch on, different procedures have been tried. Mehrotra and Trick [45] pick some vertex $u \in S$, where S is the most fractional column, and then pick a second column S' and choose a second vertex $v \in (S' \cup S) \setminus (S' \cap S)$. This way, in each branch, at least one of S and S' is invalidated. We call the procedure or function used to pick the branching vertices the *branching strategy*. In Chapter 5 several different branching strategies are investigated with the Zykov branching rule.

When the new branch-and-bound nodes have been generated, we can propagate existing stable sets which have already been generated to the node. This way, we do not need to start column generation from scratch in every branch-and-bound node. In particular, given $\mathcal{S}$ and two branching vertices u and v, we can check for each stable set $S \in \mathcal{S}$ if they violate the given branch. For G_{DIFFER}, we can initialize the new branch-and-bound node with all stable sets $S \in \mathcal{S}$ that contain at most one of u and v, e.g. we can initialize the initial set of stable sets

Instance	Branch-and-bound nodes	$\lvert\mathcal{S}\rvert$ in root node	$\lvert\mathcal{S}\rvert$ after solving
myciel5	3619	73	452
DSJC125.9	355	230	388

Table 2.1: Distribution of the number of generated stable sets over the branch-and-bound nodes

$\mathcal{S}_{DIFFER}$ as $\mathcal{S}_{DIFFER} = \{S \in \mathcal{S} || S \cap \{u, v\}| \leq 1\}$. We can do something similar for G_{SAME} and initialize the new branch-and-bound with all stable sets which either have both or neither of u and v. Then, we can set $\mathcal{S}_{SAME} = \{S \in \mathcal{S} || S \cap \{u, v\}| \neq 1\}$.

One advantage of having a coloring problem as a subproblem structure is that Algorithm 1 can be applied at every branch-and-bound node. Hansen et al. [32] have observed that applying these reductions at every branch-and-bound node reduces the computation time for most instances. We observed similar results in our preliminary tests. Note we do not use the optional clique argument for further branch-and-bound nodes, as this requires solving the maximal clique problem again, which is NP-hard.

It is worth noting that the removal of vertices of G does not affect the validity of the stable sets that have been computed. Thus, we can still share the set of columns which was generated in the child nodes, provided that they are disabled if they violate the branching constraints. However, a stable set generated in a graph of a node at a larger depth might not be maximal in a branch-and-bound node that has more vertices from the original graph. This would be a shame, as we then cannot use these columns in different parts of the branch-and-bound tree.

To change this, we can extend stable sets in subproblems from the branch-and-bound tree to also be inclusion-wise maximal in the original graph. We do this by using a greedy strategy which adds the vertices deleted from the original graph in an arbitrary order, whilst still satisfying the branching constraints. We can satisfy these branching constraints by only adding edges, and not removing the relevant vertex in a copy of the graph. For all DIFFER constraints, we only need to add one edge, and for each SAME constraint we can add edges without removing vertices (as described above). Then, a simple greedy procedure on this resulting graph will give a stable set which satisfies the branching constraints. In our implementations, we used a greedy procedure with a random vertex ordering, to generate a larger 'variety' of stable sets.

Using the above procedures, the stable sets can be transferred between branch-and-bound nodes efficiently, and all generated stable sets are also valid and inclusion-wise maximal in the root node. Together, this helps reduce the computation time significantly when doing branch-and-price computations. In Table 2.1 one can see that although many stable sets are generated in the root node, only a few stable sets generated in all other branch-and-bound nodes. For most nodes, after initialization with the initial (transferred) set of columns, only a few stable sets are priced in before the branch-and-bound node is solved. For both instances in Table 2.1, we even need fewer stable sets than branch-and-bound nodes after solving the root node; this means many branch-and-bound nodes are immediately solved to optimality after only a single run of the pricing algorithm.

Malaguti et al. [44] have done experiments with variable branching, and have found that in their case, it performs similarly to edge branching. However, as it complicates the maximum weight stable set problem, and prevents us from applying Algorithm 1 at every branch-and-bound node, we decided not to use this branching rule. Indeed, using variable branching, one would need to prohibit the generation of some stable sets, which makes the pricing problem more difficult, as the structure is then altered based on the branching decisions. In contrast,

Zykov's branching rule does not alter the pricing problem, but rather the graph itself, which more easily allows for reductions based on graph coloring arguments. The variable branching is explained more in depth by Malaguti et al. in their paper [44].

Chapter 3

Separation methods

In integer programming, cutting planes are a tool that is often used in order to speed up the branch-and-bound procedure. The core idea is to consider the linear relaxation of a formulation, and to then strengthen it by adding inequalities that are violated by its optimal solution.

Considering the Restricted Master Problem (2.2), we then seek to add classes of cutting planes to it in order to strengthen the formulation, and to hopefully speed up the computation. One important requirement here is that the generated cutting planes are compatible with column generation. In practice this means that for the generated cutting plane row, we must know what coefficient to add for each newly generated stable set. If we do not do this, we may re-generate variables which are "cut off", which could lead to an infinite pricing and separation loop [19].

One clear candidate for cutting planes in this respect are *Chvátal-Gomory cutting planes* [13, 28]. Consider the polyhedron $P = \{x \in \mathbb{R}^n : Ax \geq b, x \geq 0\}$ with $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^n$. Chvátal-Gomory cuts are cutting planes which are defined by a multiplier vector $\lambda \in \mathbb{R}^m$ with $0 \leq \lambda_i \leq 1$. This gives a row inequality of the form $\lambda^T Ax \geq \lceil \lambda^T b \rceil$, where we require that $\lambda^T A \in \mathbb{Z}^n$, e.g. the coefficients of the new row are integral. Then, when a new column a_{n+1} is generated for A in pricing, the new coefficient in the cutting plane is simply $\lceil \lambda^T a_{n+1} \rceil$. Here we can use the ceiling operator to ensure that the new row coefficient is integral.

Chvátal-Gomory cuts are a very generic framework. There have been several efforts and investigations in different applications of these cuts. Mod-k cuts were investigated by Caprara et al. in [12]. For these cuts, one considers only the multipliers $\lambda \in \{0, \frac{1}{k}, \frac{2}{k}, ..., \frac{k-1}{k}\}^m$ for defining a cut. Mod-2 cuts were examined as a special case in another paper by Caprara and Fischetti [11]. Hansen et al. applied mod-2 cuts to Graph Coloring in their paper [32], using an approach related to odd-cycle cuts which is explained in Section 3.2. For these results, we show some elaborations and improvements compared to the results as posed by Hansen.

Hansen et al. [32] also attempted to add cuts based on small subgraphs $G[V']$ with known $\chi(G[V'])$. One can efficiently detect these for odd hole graphs. However, this did not seem fruitful, as even adding thousands of inequalities based on these small graphs only lead to a small improvement in objective, and the authors also show that these inequalities do not yield a good description of the convex hull for the small instance *myciel3*.

Although other cutting plane methods exist, they are difficult to apply to the Graph Coloring Problem. For set covering programs, only a few other cutting plane methods are known.

Nobili and Sassano introduced inequalities based on 'k-projection' [51, 52]. Unfortunately,

these are not suitable with column generation as they require knowledge of the entire problem matrix, and thus all columns, in order to separate cuts.

For the set covering problem, Balas and Ho [4][3] introduced *conditional cuts*, which are cuts that are only valid under the assumption a better primal solution exists. Unfortunately, these cuts are also not very well suited for graph coloring. In order to separate a cut, a row with exactly 3 coefficients of 1 and zero otherwise is required. This is extremely unlikely in graph coloring, as it requires that $|\{S \in \mathcal{S} : v \in S\}| = 3$. For a typical application this is near impossible, as after solving the root node we have observed only $|\mathcal{S}| \geq |V|$, with for sparser graphs $|\mathcal{S}| \gg |V|$. The typical maximal stable set $S \in \mathcal{S}$ has size $|S| > 3$, even for dense graphs; in this light, the precondition is extremely unlikely to hold. Only for very dense instances one could hope that these conditions hold, but these instances are also typically considered to be much easier to solve. Conditional cuts thus do not seem to be promising for graph coloring applications.

3.1 Mod-k Cuts

Caprara and Fischetti [12] have studied Chvátal-Gomory cuts for which for a given integer $k \geq 2$, the coefficients $\lambda \in \{0, \frac{1}{k}, ..., \frac{k-1}{k}\}$. Here, we present an adaptation of these cuts for the set covering formulation of graph coloring.

In the separation problem for mod-k cuts the violation of the mod-k cut is maximized. Adapting Caprara and Fischetti's work [12], we can formulate this problem in a way relevant to the set covering formulation for graph coloring. Consider $A \in \mathbb{Z}^{m \times n}$ and $b \in \mathbb{Z}^m$. Let $P = \{x \in \mathbb{R}^n | Ax \geq b\}$, and $P_I = \{x \in \mathbb{Z}^n | Ax \geq b\}$ be the linear relaxation and integer convex hull respectively.

Suppose we have a solution $x^* \in P$ and the slack vector $s^* = b - Ax^*$. We let $\mu = k\lambda$ be integer. Then, finding violated mod k cuts amounts to the following optimization problem.

$$\begin{aligned}
&\text{Minimize} && \delta^* = s^*\mu - (k - \theta) && \text{(3.1a)}\\
&\text{subject to} && A^T\mu \equiv 0 \pmod{k} && \text{(3.1b)}\\
& && b^T\mu \equiv \theta \pmod{k} && \text{(3.1c)}\\
& && \mu \in \{0, 1, ..., k-1\}^m && \text{(3.1d)}\\
& && \theta \in \{1, 2..., k-1\} && \text{(3.1e)}
\end{aligned}$$

Note that the objective represents the violation of the found cut. Then, any cut with a strictly negative objective value is violated by the current LP solution x^*.

The above problem can also be solved as an integer program, as the congruences can be modeled by introducing an additional variable with coefficient k. For $k = 2$ the separation problem is equivalent to finding a minimum weight member of a binary clutter, which is NP-hard [11]. Thus, we generally cannot expect to solve this problem exactly.

3.1.1 Maximally Violated Mod-k Cuts

One subclass of solutions to this optimization problem is *maximally violated* mod-k cuts. For these cuts, all rows with nonzero slack are not considered, such that if $s_i^* > 0$ for the i'th row of A, $\mu_i = 0$. As $\lambda^T A$ is integral, and the right hand side b is a multiple of $\frac{1}{k}$, we can look for cuts which have the best possible objective by fixing $\theta = 1$. Searching cuts under this assumption

turns the optimization problem into a decision problem.

Since for maximally violated mod-k cuts the objective is fixed, all that remains is to find a feasible solution which satisfies the system (3.1b)–(3.1e). For prime k , this can be easily done, as we can then simply solve a system of equations over the prime field $\mathbb{Z}/k\mathbb{Z}$, which can be done using Gaussian elimination. For non-prime k, some additional work needs to be done, as the corresponding system of equations is not in a prime field and we cannot use Gaussian elimination. Caprara and Fischetti also remark this case is still easy to solve [12], but do not show in depth how to solve these issues when k is non-prime.

Let us elaborate on how to separate maximally violated mod-k cuts efficiently for arbitrary $k \geq 2$. Here, we assume that the prime decomposition of k is known. Then, we can use techniques from number theory to find solutions. For prime k, the problem is simple, but for composite k more complicated tools are necessary.

Consider an arbitrary system of equations $Ax = b \bmod k$, with $a_{ij}, b_i \in \mathbb{Z}$ for all rows i and columns j. For prime k, we are solving the problem over the finite prime field $\mathbb{Z}/k\mathbb{Z}$. This can be done efficiently by using the Wiedemann algorithm, see [70] for more elaboration. The Wiedemann algorithm also benefits from the sparsity of the matrix, making it well suited for integer programming applications.

Next, there is the case of $k = p^q$ for some prime p and integer power q. In this case, one assumes that the integer solution is a polynomial of p; this can be done by sequentially 'lifting' the solution to a polynomial of a higher degree. Algorithm 2 shows how this can be implemented in practice: it is taken from [41].

Algorithm 2 An algorithm to solve a system of equations $Ax = b \bmod p^q$. If no solution is found and $i > 0$, a solution $x \bmod p^i$ is returned instead.

```
1: function SolveModK(A,b,p,q)
2:     i = 0
3:     b_0 = 0
4:     x = 0
5:     while i < q do
6:         solve Ax_i = b_i mod p
7:         if x_i infeasible then return
8:         end if
9:         b_{i+1} = (b_i − Ax_i)/p
10:        x = x + p^i x_i
11:        i = i + 1
12:    end while
13: end function
```

Note that by updating the basis, this algorithm obtains solutions for increasingly higher powers of p. Even when it terminates early, if $i > 0$ we then still have a solution of $Ax = b \bmod p^i$.

Given a prime decomposition of k, all of its individual prime factors are coprime with each other. Suppose we want to solve a system of equation given by A, b for some $k = k_1 k_2$ where k_1 and k_2 are coprime and we have solutions $Ax_1 = b \bmod k_1$ and $Ax_2 = b \bmod k_2$. Then, by applying the Chinese Remainder Theorem [17], we can efficiently find x such that $Ax = b \bmod k$. Here, each coefficient of x is obtained by using the Chinese Remainder Theorem on the

relevant coefficients of x_1 and x_2 mod k.

For finding maximally violated mod-k cuts, there is a structure that can be exploited to find maximally violated cuts more efficiently. In particular, we know that the right hand side $b \in \{0,1\}^m$. This can be particularly useful, because for $k = k_1 k_2$ composite, we know that $b \equiv 0 \bmod k$ if and only if $b \equiv 0 \bmod k_1$ and $b \equiv 0 \bmod k_2$ hold. Similarly, $b \equiv 1 \bmod k$ holds if and only if $b \equiv 1 \bmod k_1$ and $b \equiv 1 \bmod k_2$ holds. As $\theta = 1$ is fixed, the latter holds for the last row in (3.1c) and all other rows (3.1b) have a right hand side of zero. Thus, when trying to find a mod k cut for k composite, all coefficients of b remain unchanged in the subproblems for k_1 and k_2. However, this is *exactly* the problem one solves to find a maximally violated mod k_1-cut. Thus, we only need to solve for each prime factor once, if we desire to find cuts for composite numbers which have them as divisors.

Although one could suspect that cuts with composite k are weaker or dominated, this is not necessarily the case from our computational experience. One problem instance which highlights this well is *4-FullIns_4*. The linear relaxation at the root node has objective value $\chi_f(G) \approx 6.33$, and after finding and adding a mod-2 and mod-3 cut and pricing in new stable sets, we improve to $\chi_c(G) \approx 6.56$. Adding the resulting mod-6 cut found by applying the Chinese Remainder Theorem to the two found cuts and performing pricing, improves the bound to $\chi_c(G) = 7$.

3.1.2 Modified Pricing problem

In order to be able to apply cutting planes to the set covering formulation, we also need to be able to price in new columns. The addition of cutting planes changes the associated pricing problem by changing the reduced cost of each variable. Here, we present a mixed integer linear programming formulation of the modified pricing problem. In Chapter 4 we introduce a more advanced combinatorial algorithm for the modified pricing problem.

Let H denote a mod-k cutting plane, with $\mathcal{H}$ the set of all mod-k cutting planes. Let π_v be the dual solution values corresponding to set covering constraints for vertex $v \in V$, and let μ_H be the dual solution values corresponding to a mod-k cutting plane H. Additionally, we have the multipliers of a cutting plane $\lambda_H \in \{0, \frac{1}{k_H}, .., \frac{k_H - 1}{k_H}\}^{|V|}$. Here we use the substitution $\rho_H = \lambda_H k_H$. Then, taking into the account the dual multipliers when pricing, the pricing problem becomes as follows:

$$\text{Maximize} \qquad W(y) = \sum_{v \in V} \pi_v y_v + \sum_{H \in \mathcal{H}} \mu_H \left\lfloor \frac{\rho_H^T y}{k_H} \right\rfloor \tag{3.2a}$$

$$\text{subject to} \qquad y_u + y_v \leq 1 \qquad \forall (u,v) \in E \tag{3.2b}$$

$$y_v \in \{0,1\} \qquad \forall v \in V \tag{3.2c}$$

In order to obtain a linear and more tractable problem, we introduce new variables z_H and r_H such that $z_H = \lceil \frac{\rho_H^T y}{k_H} \rceil = \frac{\rho_H^T y + r_H}{k_H}$. Note that $z_H \in \mathbb{Z}^+$ and that we can require that $r_H \in \{0, 1, ..., k_H - 1\}$. Let ρ_H^v denote the coefficient for vertex $v \in V$ of ρ_H for cutting plane

$H \in \mathcal{H}$. We obtain the following formulation:

$$\begin{aligned}
&\text{Maximize} && \sum_{v \in V} \pi_v y_v + \sum_{H \in \mathcal{H}} \mu_H z_H && && (3.3a)\\
&\text{subject to} && y_u + y_v \leq 1 && \forall (u,v) \in E && (3.3b)\\
& && y_v \in \{0,1\} && \forall v \in V && (3.3c)\\
& && k_H z_H = \sum_{v \in V} \rho_H^v y_v + r_H && \forall H \in \mathcal{H} && (3.3d)\\
& && z_H \in \mathbb{Z}^+ && \forall H \in \mathcal{H} && (3.3e)\\
& && r_H \in \{0, 1, ..., k_H - 1\} && \forall H \in \mathcal{H} && (3.3f)
\end{aligned}$$

Note that since ρ_H and k_H are known when solving the pricing problem, this is an integer program. By construction, z_H gives the priced coefficient for cutting plane H. Although this formulation is easy to read, all variables r_H can be removed from the linear program and equation (3.3d) can be weakened to the following:

$$k_H z_H \leq \sum_{v \in V} \rho_H^v y_v + k_H - 1 \qquad \forall H \in \mathcal{H} \qquad (3.3g)$$

We can argue this as z_H is maximized with positive coefficients μ_H, and since r_H is not in the objective that this is sufficient. Note that we can then simply set $r_H = k_H - 1$, to its maximum, which allows us to eliminate it from the program.

Important to note is that the lower bound (2.4) can still be used as before, but now using $W(y)$ as defined in (3.2a), as the lower bound uses the reduced cost.

Lastly, an important detail to discuss about the modified pricing problem is that of *lazy upper variable bounds*. When considering the original Restricted Master Problem (2.2), the variable upper bounds $x_S \leq 1$ are implicitly satisfied; if $x_S \geq 1$, all of the constraints associated with the vertices $v \in S$ in (2.2b) are feasible, and the objective will never increase x_S past 1 as it is minimized in the objective (2.2a). This concept is also known as that of *lazy bounds* within SCIP. If inequalities from cutting planes are added the above reasoning does not work, as not all inequalities with non-zero coefficients for x_S are necessarily feasible if $x_S = 1$. Thus, in this case the upper bound inequalities $x_S \leq 1$ need to be added explicitly to the Linear Program. This may somewhat slow down the computation speed of the linear program when using cutting planes. Note that the dual multipliers of these upper bounds do not enter the above pricing problem, as the priced in coefficient for each new stable set in all existing inequalities is simply zero by definition.

3.1.3 Selecting k

One practical question which remains is how to select k when deciding which mod k cuts to investigate. In this section, we detail a heuristic which selects a range of k's to be considered. When investigating this issue for Mycielski graphs, we noticed a pattern; maximally violated mod-k cuts were only found for those k where k was a prime factor of the rational representation of $\chi_f(G)$. For the Mycielski graphs M_i, an analytical formula for $\chi_f(M_i)$ is known [40], and these cuts matched the denominators of the rational representation of $\chi_f(M_i)$ exactly.

Although the denominator of the rational representation of the Mycielski graph is known, for arbitrary graphs we cannot hope to find these easily. In particular, the denominator can grow very large very quickly. For example, M_7 has a denominator of roughly $2.7 \cdot 10^{23}$ with

$|V| = 191$ [23, 40]. Thus, we need a different approach. Since each stable set has a coefficient of 1 in the objective, we can also consider the optimal LP solution x^* for this purpose. Then, consider each variable value x^*_S. Given a procedure which can estimate $n, d \in \mathbb{Z}$ such that $x^*_S \approx \frac{n}{d}$, we can then simply take the Least Common Multiple of all denominators d to obtain an estimate of the denominator of the objective value. Since we need to decompose this number again into its prime factors, this can be done without needing to generate the very large Least Common Multiple explicitly, by keeping a list of prime factors and their multiplicity.

The integer programming solver used in this research, SCIP [26], has procedures for finding $n, d \in \mathbb{Z}$ with a large denominator limit D such that $d < D$. In our computations, we used $D = 2^{16}$, as this takes better advantage of modern computer hardware when solving systems of equations using the Wiedemann Algorithm. Additionally, cuts were typically found more often for small primes k, where $k < 100$ (see Section 6.3.2 for details and results). Thus, searching for cuts with very large denominators did not seem very productive.

3.2 Odd-cycle cutting planes

In this section we will explain how odd-cycle cutting planes can be used and found for the set covering formulation of graph coloring. In practice, this can be seen as a method to separate mod-2 cutting planes. This approach is almost identical to the one described by Hansen et al [32]; we will note it whenever we add new results which are our own work. Hansen et al. build their argument primarily on the work done by Caprara and Fischetti [11], who rely on the results of Gerards and Schrijver [27]. Caprara and Fischetti formulate the following theorem.

Theorem 3.2.1 (Caprara and Fischetti [11])**.** *The separation problem* (3.1) *for mod-*2 *(*$k = 2$*) cutting planes can be solved in polynomial time if* A^T *is an EPT matrix.*

A matrix A^T is *EPT* if A has at most 2 odd coefficients for each row. Note this is not a definition, but a useful sufficient condition. As we have a binary matrix and our stable sets are typically of size $|S| > 2$, this is not directly applicable. However, one can find odd-cycle cutting planes by *weakening* the rows of the matrix A.

Consider the constraints (2.2b) as a binary matrix. A cutting plane can be found by using Chvátal's procedure for cutting planes, by choosing a subset of rows, with $\lambda_i = \frac{1}{2}$ if row i is included in the subset, and $\lambda_i = 0$ otherwise. In the set covering formulation, each row corresponds to a vertex in the original graph. We will abuse notation and use H to denote this vertex subset in the original graph, corresponding to the relevant rows. Odd-cycle cuts can then be seen as a special kind of mod-2 cuts.

In order to formalize the odd-cycle cuts, we construct the *conflict graph*. The conflict graph G_c is defined to have a vertex for each stable set, and an edge if two stable sets have a non-empty intersection. More formally we define the conflict graph as

$$G_c = (\mathcal{S}, \{\{S, S'\} | S \cap S' \neq \emptyset, \forall S, S' \in \mathcal{S}\}) \tag{3.4}$$

Now consider a cycle C in G_c. Then for each edge $(S, S') \in C$ we can choose a vertex $v \in G$ such that $v \in S \cap S'$. Each edge $(S, S') \in C$ then has a corresponding row defined by the relevant vertex $v \in S \cap S'$. We then use H to denote the set of vertices corresponding to the edges for the cycle C, such that $\lambda_v = \frac{1}{2}$ if and only if $v \in H$.

If we consider a feasible solution $x^\star$, then we can compute for a cycle whether or not inequality associated with our cycle C is violated. Should it be violated, we can add it to our linear program to strengthen it.

First we will discuss a weaker case of these cuts. Here, the original rows of the LP are weakened by adding constraints $x_S \geq 0$ to the rows. For an edge (S, S') we consider the normal row for a vertex $v \in S \cap S'$ to be simply the inequality (2.2b).

$$\sum_{S \in \mathcal{S} | v \in S} x_S \geq 1$$

Then, we can weaken the inequality by increasing the coefficients of all sets which are not in edge in the conflict graph, corresponding to this vertex. Considering all cycles, this expands the single inequality from above into a set of inequalities:

$$x_S + x_{S'} + 2 \sum_{S \in \mathcal{S} \setminus \{S,S'\} : v \in S} x_S \geq 1, \forall S, S' \in \mathcal{S} | v \in S \cap S' \tag{3.5}$$

Now, we use A and b as defined by the matrices after doing the above procedure for every row. As all the stable sets in the cycle are counted twice, and all other coefficients are even, we then know that $\lambda A \in \mathbb{Z}^n$ as, all coefficients are then even before dividing λ by 2. Then, we can trivially round the right hand side $\lceil \lambda b \rceil = \lceil \frac{|H|}{2} \rceil$. Then, the minimum cost of the cycle C is simply the sum of the costs of each edge. Note that because we do not need to round the entries for A, that the edge costs are independent. Given a fractional solution $x^\star$, we can then find minimum-weight odd cycles by defining the weight of the edge to be

$$w((S, S')) = x^\star_S + x^\star_{S'} - 1 + 2 \min_{v \in S \cap S'} \sum_{S'' \in \mathcal{S} \setminus \{S,S'\} : v \in S''} x^\star_{S''}, \tag{3.6}$$

Note that this is simply the same as choosing $v \in S \cap S'$ such that the corresponding constraint has minimal slack. Then the total weight of a cycle C equals $w(C) = 2\lambda A x^\star - |C|$, where λ now simply indicates which rows are in the cycle. If we then find a cycle C such that $w(C) = \sum_{e \in E(C)} w(e) < 1$, the corresponding mod-2 inequality is violated, as from the basic argument by Chvátal and Gomory we must have that $\lambda A x^\star \geq \lceil \lambda b \rceil = \frac{|C|+1}{2}$.

Thus, using the distance as defined above, any odd-length cycle in G_c with weight smaller than 1 implies a cutting plane that cuts off the fractional solution $x^\star$. Finding a minimum weight odd cycle (if it exists) can be done in polynomial time in size of the conflict graph [32].

We can compute the minimum weight cycle by considering the bipartite graph G'_c that has two copies of the stable sets $\mathcal{S} = \mathcal{S}'$ as its nodes. The edges of G'_c are defined as $E(G'_c) = \{(S, S') \in \mathcal{S} \times \mathcal{S}' \ : \ S \cap S' \neq \emptyset\}$. Then, finding a minimum weight odd cycle starting from some stable set S in the conflict graph G_c is equivalent to finding a shortest path from $S \in \mathcal{S}$ to $S \in \mathcal{S}'$ in G'_c. We can find the minimum weight odd cycle for G_c by repeating the shortest path procedure for every stable set $S \in \mathcal{S}$. Using Dijkstra's algorithm, this puts the complexity of finding a minimum weight odd-cycle on $O(|V||E| + |V|^2 \log |V|)$, which for our case becomes $O(|\mathcal{S}|^3)$.

3.2.1 Limited to fractional variables

Hansen et al. claim the following result in Theorem 3.2.2, but do not prove it explicitly [32]. Here, we provide a proof.

Theorem 3.2.2 (Hansen et al, [32])**.** *The length of an odd cycle C in the conflict graph G_c is greater than or equal to $|2x^\star_S - 1|$ for any stable set $S \in V(C)$.*

Proof. First, we have that for any edge (S_i, S_j) within $E(C)$ that $w((S_i, S_j)) \geq |x^\star_{S_i} + x^\star_{S_j} - 1| = |1 - x^\star_{S_i} - x^\star_{S_j}|$. If $x^\star_{S_i} + x^\star_{S_j} - 1 \geq 0$, this fact can be derived by observing the remaining terms are also positive. For $x^\star_{S_i} + x^\star_{S_j} - 1 < 0$, we can show this fact as well, using that $x^\star_{S_i} + x^\star_{S_j} - 1 + \min_{v \in S_i \cap S_j} \sum_{S'' \in \mathcal{S} \setminus \{S_i, S_j\} : v \in S''} x^\star_{S''} \geq 0$ as the slack of the associated row is positive.

$$\begin{aligned} w((S_i, S_j)) &= x^\star_{S_i} + x^\star_{S_j} - 1 + 2 \min_{v \in S_i \cap S_j} \sum_{S'' \in \mathcal{S} \setminus \{S_i, S_j\} : v \in S''} x^\star_{S''} \\ &\geq x^\star_{S_i} + x^\star_{S_j} - 1 + 2(1 - x^\star_{S_i} - x^\star_{S_j}) \\ &= 1 - x^\star_{S_i} - x^\star_{S_j} \end{aligned}$$

Then, let S_i be the i'th vertex in the cycle. Then, we obtain the following by alternating signs and adding them together, as in general, we know that $|a| + |b| \geq |a + b|$

$$\begin{aligned} w(C) =& \sum_{e \in C} w(e) \\ \geq& |x^\star_{s_1} + x^\star_{s_2} - 1| + |x^\star_{s_2} + x^\star_{s_3} - 1| + ... + |x^\star_{s_{|C|-1}} + x^\star_{s_{|C|}} - 1| + |x^\star_{s_{|C|}} + x^\star_{s_1} - 1| \\ =& |x^\star_{s_1} + x^\star_{s_2} - 1| + |1 - x^\star_{s_2} - x^\star_{s_3}| + ... + |1 - x^\star_{s_{|C|-1}} - x^\star_{s_{|C|}}| + |x^\star_{s_{|C|}} + x^\star_{s_1} - 1| \\ \geq& |2x^\star_{s_1} - 1| \end{aligned}$$

Note, that we can start this proof from any vertex $s \in V(C)$ and the argument still holds. Note we know the outcome of the alternating signs by the parity of $|C|$. This concludes our proof. □

Since $|2x^\star_s - 1| = 1$ if $x^\star_s = 1$ or $x^\star_s = 0$, this implies the weight of a cycle containing a non-fractional variable is always greater or equal than 1, ensuring that this cycle is not violated. This means that we only need to consider all fractional variables when constructing the conflict graph. Since in many practical examples, the number of fractional variables for an optimal solution $x^\star$ can be orders of magnitude smaller than the current amount of columns in the LP, $|\mathcal{S}|$, this can significantly reduce the computational burden for calculating the minimum weight odd-length cycle.

3.2.2 Strong odd-cycle cuts

Strong odd-cycle cuts are similar to the weaker odd-cycle cuts as described in Section 3.2. The major difference, is that we do not weaken the row inequalities before adding them together. Then, we consider the matrices A and b to be the inequalities (2.2b) together with the lower bound constraints $x_S \geq 0$. If we consider A as the matrix just defined by the row constraints (2.2b), we cannot expect that $\lambda A \in \mathbb{Z}^n$. During column generation, we typically have $|\mathbb{S}| > |V|$, and then each coefficient x_S in the produced row inequality would need to be even by coincidence when obtained from the row sum of at most $|V|$ rows. Instead, we can round the coefficient for each x_S to be even by adding the lower or upper bound constraint. This is an application of the approach described by [11]. In this case, we are simply separating mod-2 cutting planes. Any weakening done using fractional variables then makes it a cutting plane which is not maximally violated.

Hansen et al. only considered the lower bound rounding in their paper [32]. This is a sensible decision, as it is very rare for $x^\star_S \geq \frac{1}{2}$ to occur in a solution for the Restricted Master Problem. We too use this scheme.

When pricing a new set S' with some existing cutting plane H, we know that $x^*_{S'} = 0$ at the time the cutting plane H was defined. Then we always use the lower bound weakening for newly priced stable sets in the cutting plane coefficient.

Suppose we find that some set of vertices H defines a weak-cycle cut. Then, we can consider the sum the inequalities (3.5) for this cycle as part of its weight. Note, these inequalities are dominated by the inequalities (2.2b). If we instead consider their sum in a mod-2 cut, and add the lower bound constraints $x_S \geq 0$ if the resulting coefficient is odd so that we can divide by 2, we must always obtain an inequality with violation at least as big as defined by the odd-cycle procedure. The weak inequalities are rounded at every intermediate step, and thus must be coefficient wise larger. Thus, for each violated weak odd-cycle cutting plane, an associated stronger mod-2 cut exists which is also violated. From now on, when we consider the odd-cycle cuts, we will use the strong mod-2 cuts as described here.

3.2.3 Modified pricing problem

As the odd-cycle cuts are simply mod-2 cuts, the modified pricing problem for strong odd-cycle cuts is equivalent to the modified pricing problem for mod-k cuts in system (3.2a)–(3.2c). Indeed we have $k = 2$ for the relevant cutting planes which originate from the odd-cycle cuts, with $\rho^v_H = 1$ if and only if $v \in H$.

In their implementation, Hansen et al. use a linear approximation, ignoring the ceiling function in equation (3.2a) for greedy algorithms. If that fails, they run a simple branch-and-bound algorithm. In Chapter 4 we formulate a more advanced branch-and-bound algorithm based on the algorithm developed by Held et al. in [36], and explain some greedy procedures.

3.2.4 Polyhedral results

Our odd-cycle cuts satisfy the necessary condition formulated and proven by Schindl [59], proof for which is presented in Theorem 3.2.3. Here, $Chv(V', k)$ is the inequality which is derived by doing Chvátal's procedure for the vertices $v \in V'$ and dividing the resulting equation by k before rounding up, e.g. the mod-k cut with $\lambda_v = \frac{1}{k}$ if and only if $v \in V'$ and $\lambda_v = 0$ otherwise.

Theorem 3.2.3 (Schindl [59])**.** *If $Chv(V', k)$ defines a facet, then $\lceil \frac{|V'|}{k} \rceil > \omega(G[V'])$.*

We prove that the condition posed by Theorem 3.2.3 always holds for odd-cycle cuts in Theorem 3.2.4.

Theorem 3.2.4. *All odd-cycle cuts satisfy $\lceil \frac{|V'|}{k} \rceil > \omega(G[V'])$*

Proof. Note in our case, we have that $k = 2$ and $|V'| = |H|$ odd. Now, consider our odd-cycle procedure. Let $M \subseteq H$ be a maximal clique of $G[H]$, e.g. $\omega(G[H]) = |M|$. Let $v, w \in M$ be two vertices in the maximal-size clique. Since M is a clique, the edge $(v, w) \in E$ exists. This implies that no stable set contains both v and w.

Then considering an edge (S, S') in our stable cycle, we have that $v \in S$ and $v \in S'$ by construction for some $v \in S \cap S'$. This implies S' and S do not contain any other vertex in

M but v. Then, considering the next edge $(S'S,'')$ in our cycle, we know the only possible candidate $u \in S' \cap S'' \cap M$ is v (assuming that $v \in S''$), as $v \in S'$ and S' can contain no other vertex. But, in the odd-cycle procedure, $u \in S' \cap S'' \setminus \{v\}$ by construction as each row is only picked once. Thus, we must then conclude that $S' \cap S'' \cap M = \emptyset$ is empty. As this argument holds for any two consecutive edges of the conflict graph, this gives that $|H| \geq 2|M|$. However, since $|H|$ is odd by construction, we then know that $|H| \geq 2|M| + 1$ must hold.

This implies that $\lfloor \frac{|H|}{2} \rfloor \geq |M| = \omega(G[H])$ must hold, which implies (by oddness of $|H|$), that $\lceil \frac{H}{2} \rceil > \omega(G[H])$. Thus, the odd-cycle cuts satisfy the necessary condition as formulated by Schindl to be facet defining. □

3.3 Cutting planes with larger Chvátal Rank

In Sections 3.1 and 3.2 we discussed mod-k and odd-cycle cuts based on the inequalities (2.2b), the original problem matrix. In this section, we will discuss how to extend these ideas to be able to include cuts using arbitrary rows valid for the Restricted Master Problem. This would allow us to define cutting planes based on 'new' rows which have been generated from cutting planes.

This idea is connected with the notion of Chvátal-Rank for Chátal-Gomory cuts. Cuts which are defined on the original inequalities (2.2b) are said to have Chvátal rank 1. These cuts only require a single rounding step to derive. Then, if one finds a new Chvátal-Gomory cut for the system which includes the original inequalities (2.2b) and the derived Chvátal rank 1 inequalities, this cut is said to have Chvátal-Rank 2. Note that here, two 'levels' of rounding steps were required to derive the newly added row; once for all rank 1 inequalities, and once for all rank-2 inequalities. Higher rank inequalities can be defined in exactly the same manner. A more rigorous definition and discussion of Chvátal-Rank and can be found in integer programming textbooks, see for example [60].

Before we are able to separate cuts with Chvátal rank greater than 1, we need to tackle two challenges. First, the separation methods need to be able to efficiently separate higher rank cuts. Secondly,the pricing problem needs to be adjusted to deal with these cuts, and algorithms for the pricing problem should still be efficient when dealing with higher rank cuts.

3.3.1 Separation of maximally violated mod-k cutting planes

In our separation procedure for mod-k cutting planes, we used generic notation, e.g. matrices A and b . Adding rows to A from mod-k cutting planes or odd-cycle cutting plane procedures is then easy, as all results still hold when this is done. The only major modification is that due to the fact that we now no longer have a binary vector b, we may not be able to directly combine cuts using the Chinese Remainder Theorem, but we might need to recompute the solution of one of the prime factors.

Adjusting the separation problem for mod-k cutting planes then simply amounts to adding these rows, which become columns in the system of equations which is solved mod k by transposition. With more columns, it is more likely the system of equations yields feasible solutions, which then gives us more solutions and also more cutting planes.

3.3.2 Separation of odd-cycle cutting planes

In Section 3.2 we explained how to separate odd-cycle cuts for the original problem matrix. Adding general rows from cutting planes to this procedure breaks several assumptions. However, the same underlying ideas can still be used; by first weakening each row to have at most 2 odd coefficients, one can still formulate a polynomial time algorithm for separating cutting planes using Theorem 3.2.1 [11, 27].

First, we can again construct a conflict graph over the variables, but this time also taking the generated rows into account. Assume a set of rows M exists of the form $a_m x \geq b$. For each row a_m, let T_m be the set of stable sets with odd coefficients, more formally, $T_m = \{S \in \mathcal{S} : a_m^S \equiv 1 \mod 2\}$. Then we can again define the conflict graph G_c, but this time using all rows to define edges; an edge (S, S') only exists if a row m exists such that $S' \in T_m$ and $S \in T_m$. More formally we have $V(G_c) = \mathcal{S}$, with $E(G_c) = \{(S, S') \in \mathcal{S}^2 \text{ and } \exists m | S \in T_m, S' \in T_m\}$. Note that this definition yields the exact same conflict graph as definition (3.4) when only the problem inequalities (2.2b) are considered as rows.

We can still consider cycles in the undirected conflict graph G_c to attempt finding cutting planes. However, unlike before, we now have coefficients b_m for which $b_m \neq 1$. When considering the inequality found by a sum of set of rows M, the constant coefficient of this row becomes $\sum_{m \in M} b_m$. In order to apply the rounding argument for Chvátal Gomory cuts, we then need to find a cycle such that the corresponding inequality has $\sum_{m \in M} b_m \equiv 1 \mod 2$. We can solve this issue by differentiating *even* and *odd* rows, e.g. rows with $b_m \equiv 0 \mod 2$ and rows with $b_m = 1 \mod 2$.

Consider the undirected conflict graph G_c', which has two copies of all vertices , e.g. $V(G_c') = \mathcal{S}_0 \cup \mathcal{S}_1$, where $\mathcal{S} = \mathcal{S}_0 = \mathcal{S}_1$. Then, we consider edges by the oddness of their right hand side coefficient; adding edges corresponding to odd rows for the subsets $\mathcal{S}_0 \times \mathcal{S}_1$, and edges corresponding to even rows to $\mathcal{S}_0 \times \mathcal{S}_0$ and $\mathcal{S}_1 \times \mathcal{S}_1$. To be complete, define $M_i(S, S') = \{m \in M : a_m^S \equiv 1 \mod 2, a_m^{S'} \equiv 1 \mod 2, b_m \equiv i \mod 2\}$ for $i = 0$ and $i = 1$. Then the edges can be defined as follows:

$$\begin{aligned} E(G_c') = &\{(S_0, S_1) \in \mathcal{S}_0 \times \mathcal{S}_1 : M_1(S_0, S_1) \neq \emptyset\} \cup \\ &\{(S_0, S_1), \in \mathcal{S}_0 \times \mathcal{S}_0 : M_0(S_0, S_1) \neq \emptyset\} \cup \\ &\{(S_0, S_1), \in \mathcal{S}_1 \times \mathcal{S}_1 : M_0(S_0, S_1) \neq \emptyset\} \end{aligned}$$

Consider an arbitrary edge (S_0, S_1). The weight of this edge follows from $M_i(S_0, S_1)$ in a natural way. We can again apply the rounding argument from [11], adding variable lower bounds to each odd coefficient to make them even (except for the edge). Let $g_m(S_0, S_1)$ be the row a_m after rounding:

$$g_m^S(S_0, S_1) = \begin{cases} a_m^S & \text{if } a_m^S \equiv 0 \mod 2 \vee S = S_0 \vee S = S_1 \\ a_m^S + 1 & \text{otherwise} \end{cases}$$

Then the weight of an edge can again be defined by minimizing the resulting slack of this row for a given solution x^*:

$$W((S_0, S_1)) = \begin{cases} \min\limits_{m \in M_1(S_0, S_1)} g_m^T(S_0, S_1) x^* - b_m & \text{if } (S_0, S_1) \in \mathcal{S}_0 \times \mathcal{S}_1 \\ \min\limits_{m \in M_0(S_0, S_1)} g_m^T(S_0, S_1) x^* - b_m & \text{if } (S_0, S_1) \in \mathcal{S}_0 \times \mathcal{S}_0 \\ \min\limits_{m \in M_0(S_0, S_1)} g_m^T(S_0, S_1) x^* - b_m & \text{if } (S_0, S_1) \in \mathcal{S}_1 \times \mathcal{S}_1 \end{cases} \tag{3.7}$$

Note that $g_m(S_0, S_1)$ can again contain at most 2 odd entries (S_0 and S_1), so Theorem 3.2.1 applies. Otherwise, we cannot find a polynomial time algorithm to separate the mod-2 cutting planes. The above definitions are consistent with the previously defined odd-cycle cutting planes, such that one recovers the approach from Section 3.2 when considering only the inequalities (2.2b).

Then, as before, we can simply search for a shortest weight path over G'_c from some set S_0 to its corresponding set S_1 to find a cycle in G_c. For a cycle, the associated rows can be recovered by simply substituting 'arg min' for 'min' in definition (3.7) .

Then, a path in $C \in G'_c$ from a stable set $S \in \mathcal{S}_0$ to itself in $\mathcal{S}_1$ corresponds to a cycle in G_c. If the total weight of the path is less than 1, a violated odd-cycle inequality is found, e.g. by rounding the constant coefficient, the resulting inequality would become invalid. This can again be done using Dijkstra's algorithm, achieving a worst-case time of $O(|\mathcal{S}|^3)$. Additionally, note all row coefficients in the computed row $\sum_{m,(S_0,S_1)\in C} g_m(S_0, S_1)$ are even, so that we can safely divide by 2 and round to obtain a mod 2 cut.

Note that similarly to before, we have only described the 'weak' odd-cycle inequalities. We must have that $R(\sum_{m\in C} a_m)^T x^* \leq (\sum_{m,(S_0,S_1)\in C} g_m(S_0, S_1))^T x^*$, where $R(a)$ represents the rounding operation on odd coefficients of the row a, increasing them by one. Doing this, we can potentially find a stronger odd-cycle inequality, and we can use the framework for mod-k inequalities to formulate it as a mod-2 inequality.

One final difference with the original approach and for higher order cuts is that Theorem 3.2.2 is not valid anymore in this new context. Being able to limit the search to fractional variables is important for computation time, as the number of fractional variables is smaller or equal to the number of rows in the current LP, which is typically much smaller than $|\mathcal{S}|$. Of course, we can still choose to simply not separate using integer variables. Theorem 3.3.1 formulates conditions under which separating non-fractional variables is not useful for finding cyles with an odd number of edges.

Theorem 3.3.1. *Consider an cycle $C \in G_c$ with $|C|$ odd. If the condition*

$$s^*_m + \sum_{S\in T_m\setminus\{S_0,S_1\}} x^* \geq x^*_{S_0} + x^*_{S_1} - 1$$

holds for all $S_0, S_1 \in T_m$, for all rows $m \in M$, then $W(C) \geq 1$ if the cycle C goes through a stable set S with $x^*_S = 0$ or $x^*_S = 1$.

Proof. First, we consider the edges of cycle C and the associated LP rows m and stable sets S_0, S_1, and rewrite it to a form similar to one in Theorem 3.2.2.

$$\begin{aligned} W(C) &= \sum_{(m,(S_0,S_1))\in E(C)} g_m^T(S_0, S_1)x^* - b_m \\ &= \sum_{(m,(S_0,S_1))\in E(C)} x^*_{S_0} + x^*_{S_1} - 1 + g_m^T(S_0, S_1)x^* - x^*_{S_0} - x^*_{S_1} - (b_m - 1) \end{aligned}$$

Then, if $g_m^T(S_0, S_1)x^* - x^*_{S_0} - x^*_{S_1} - (b_m - 1) \geq 0$ for all rows in the cycle, we can apply the proof of Theorem 3.2.2 to the remaining terms. This condition can be rewritten by substituting in $g_m^T x^* = a_m^T x^* + \sum_{S\in T_m\setminus\{S_0,S_1\}} x^*$, and substiuting in the row slack $s^*_m = a_m^T x^* - b_m$

$$
\begin{aligned}
g_m^T(S_0, S_1)x^* - x^*_{S_0} - x^*_{S_1} - (b_m - 1) \geq 0 \\
a_m^T x^* + \sum_{S \in T_m \setminus \{S_0, S_1\}} x^* - x^*_{S_0} - x^*_{S_1} - (b_m - 1) \geq 0 \\
s^*_m + \sum_{S \in T_m \setminus \{S_0, S_1\}} x^* \geq x^*_{S_0} + x^*_{S_1} - 1
\end{aligned}
$$

Then if this holds for all rows $m \in M$ it also holds for each cycle C of odd length. The rest of the argument is similar to the argument in the proof of Theorem 3.2.2. Note we only need to additionally require that $|C|$ is odd.

$$
\begin{aligned}
W(C) &= \sum_{(m,(S_0,S_1)) \in E(C)} x^*_{S_0} + x^*_{S_1} - 1 + g_m^T(S_0, S_1)x^* - x^*_{S_0} - x^*_{S_1} - (b_m - 1) \\
&\geq \sum_{(m,(S_0,S_1)) \in E(C)} x^*_{S_0} + x^*_{S_1} - 1 \\
&\geq |2x^*_{S_0} - 1| \quad \forall S_0 \in V(C)
\end{aligned}
$$

From the last equation it becomes clear that if $S_0 \in V(C)$ with $x^*_{S_0} = 1$ or $x^*_{S_0} = 0$, then $W(C) \geq 1$, which concludes our proof.

□

The condition of Theorem 3.3.1 is strong in the sense that many LP optima satisfy it in practice; it is uncommon for $x^*_S \geq \frac{1}{2}$ to occur in any LP optima, which is clearly required at least once in order to even make the right hand side positive. It is also easy to check, as for each row m we only need to check for the two sets $S_0, S_1 \in T_m$ with largest x^* to verify the condition for all pairs of sets in T_m. This provides a decent justification for checking only the fractional variables.

3.3.3 Pricing

Higher order cuts also change the pricing problem; the modified pricing problem (3.2a)–(3.2c) is only valid for rank-1 cuts. A rank-1 cut then combines the coefficients of the original inequalities and rounds up to obtain the coefficient in the rank-1 cut. Similarly, rank-n cuts need the coefficients in the original system of inequalities and all cuts of rank $n-1$ or lower to compute their coefficient.

For rank-1 cuts the coefficient computed for the cutting plane is part of the objective in (3.2a). Given a solution vector y and a cutting plane H, this *computed coefficient* γ_H is then defined as $\gamma_H = \lceil \frac{\rho_H^T y}{k_H} \rceil$. Note how this is simply a sum of the coefficients λ_H and then rounded up. This procedure can be extended to include the cutting planes by considering extending λ_H to also contain multipliers for cutting plane rows.

Let $\mathcal{H}_n \subseteq \mathcal{H}$ be the set of all cutting planes with Chvátal rank n, and let $\theta_{H',H} \in \{0, \frac{1}{k_H}, ..., \frac{k_H - 1}{k_H}\}$ be the mod-k multiplier of cutting plane H' in the cutting plane H. We define the computed coefficient γ_H of a cutting plane H of rank N as

$$
\gamma_H = \left\lceil \frac{\rho_H^T y}{k_H} + \sum_{n=1}^{N-1} \sum_{H' \in \mathcal{H}_n} \theta_{H',H} \gamma_{H'} \right\rceil \tag{3.8}
$$

Then, the modified pricing problem with large rank Chvátal cuts can be defined naturally as

$$\text{Maximize} \quad \sum_{v \in V} \pi_v y_v + \sum_{H \in \mathcal{H}} \mu_H \gamma_H \tag{3.9a}$$

$$\text{subject to} \quad y_u + y_v \leq 1 \qquad \forall (u,v) \in E \tag{3.9b}$$

$$y_v \in \{0,1\} \qquad \forall v \in V \tag{3.9c}$$

Although this is a sufficient formulation, it is not clear if we can efficiently compute γ_H for a cutting plane H of rank N as it depends on computed coefficients γ'_H, where the rank of H' is smaller than N.

$$\sum_{v \in V} \pi_v y_v + \sum_{n=1}^{N} \sum_{H \in \mathcal{H}_n} \mu_H \gamma_H \tag{3.10}$$

By ordering the cutting planes by Chvátal rank as in (3.10), we can guarantee that each coefficient γ_H only needs to be computed once in order to evaluate the objective for a given vector y. This is important for efficiency in combinatorial algorithms to solve the pricing problem.

Chapter 4

The pricing problem

In this chapter we will discuss the pricing problem, and in particular, how to tackle the modified pricing problems that arise as a result of adding cutting planes as discussed in Chapter 3. The effectiveness of a branch-price-and-cut algorithm typically depends on the algorithm for the pricing problem, as the pricing problem accounts for the majority of the spent time in typical branch-and-price applications [30, 36]. The pricing algorithm must decide if negative reduced cost columns exist, and if so, provide one or multiple negative reduced cost columns and add them to set of columns in the Restricted Master Problem. For the (modified) pricing problem for graph coloring (3.2a)–(3.2c), this is equivalent to deciding if a stable set y exists that has a weight $W(y) > 1$, providing a relevant column y with $W(y) > 1$ if one exists.

4.1 Background

As discussed in Chapter 2, the pricing problem for the set covering formulation is a maximum weight stable set problem. By taking the complement graph $G(V, \bar{E})$, the problem can also be formulated as a maximum (vertex) weight clique problem. Extensive literature is available on these problems, with many different heuristic and exact approaches. The maximum weight stable set problem is well known to be NP-hard [39], and no good approximation schemes are known either. Indeed, Håstad has shown that for every real number $\epsilon > 0$, no polynomial time algorithm that approximates the solution to within $O(n^{1-\epsilon})$ can exist, unless P = NP [34].

Given a stable set S, one can perform a *(1,2) swap* by replacing a vertex $v \in S$ with two vertices $w_1, w_2 \in V$ if $\pi_{w_1} + \pi_{w_2} > \pi_v$. Using efficient data structures, (1,2) swaps can be detected in polynomial time [1, 36]. In a similar way one can also find $(2, k)$ swaps, where two vertices $u, v \in S$ are replaced by a set of vertices $w_1, w_2, ..., w_n \in N(u) \cup N(v)$ if $\sum_{i=1}^{n} \pi_{w_i} > \pi_u + \pi_v$. Malaguti and Toth implement (1,2) swaps in a tabu search framework, where the initial set S is randomly generated [44]. Held et al. use both (1,2) swaps and (2,k) swaps, initializing them with the results of greedy algorithms using several different orderings [36]. Many more heuristics exist among which a genetic algorithm [50], though most are variations of local search [55, 56] or tabu search algorithms[5, 72].

Although heuristics are useful in finding solutions quickly, we desire to solve the maximum weight stable problem exactly and thus need exact algorithms to do so. Combinatorial algorithms are widely used in literature for this purpose. For dense graphs, the combinatorial CLIQUER algorithm by Ostergard [53] is deemed as one of the best available algorithms for the maximum weight stable set problem by Held et al. [36]. Held et al. also introduce a similar combinatorial algorithm [36]. More implementations exist, more detail and explanations can be found in [42, 67]. Most of the combinatorial algorithms work well as a result of the branching

rule formulated by Balas and Xue [2].

Other exact solutions exist as well; many branch-and-cut methods (see [57]) and a few branch-and-price methods [68] have also been used successfully to solve the maximum weight stable set problem exactly. However, typically combinatorial algorithms seem to be faster and more successful in solving the maximum weight stable set problems which arise from graph coloring, when comparing the results of both Held et al.[36] and Hansen et al. [32] with other branch-and-price algorithms using integer programming solvers [30, 44].

4.1.1 Numerical safety

Numerical safety can be a serious concern when solving graph coloring problems [36]. Although software which solves Linear Programming problems exactly exists, its performance is orders of magnitude worse than that of Linear Programming solvers which use floating-point arithmetic [36]. However, choosing a floating-point arithmetic solver comes with the drawback that we do not know the exact value of all found primal solutions x^* and the corresponding dual values π. When solving the pricing problem only a floating-point approximation $\pi_f \approx \pi$ is available for use. This can be problematic, as based on π_f it may be impossible to decide for a given column y if $W(y) > 1$ due to numerical noise in π_f.

In order to overcome this problem, Held et al. proposed converting the floating point weights π_f to an integer representation π_i [36]. As integer arithmetic can be done exactly and cheaply by computers, it is then desirable to do computation on π_i instead of π_f. Noting that $\pi_f \in [0,1]^{|V|}$, we can then project the vector π_f to a range of integers $\pi_i \in [0, I_{\max}]^{|V|}$ by picking some scale factor K and defining the following transformation:

$$\pi_{i,v} = \lfloor K\pi_{f,v} \rfloor \quad \forall v \in V \tag{4.1}$$

From the above transformation, we can deduce that $\frac{\pi_i}{K} \leq \pi_f$, giving us a good lower approximation of π_f. In particular this allows us to conclude that if $\pi_i^T y > K$ for some vector y, then $W(y) > 1$. One crucial element here is that at no point in our computations we must get intermediate integer values which are outside of the range $[I_{\min}, I_{\max}]$ which is supported by the computer hardware. This can be done by picking K smartly:

$$K = \frac{\min\{-I_{\min}, I_{\max}\}}{\sum_{v \in V} \pi_{f,v}} \tag{4.2}$$

Then, any intermediate sum using the coefficients π_i will fall in $[I_{\min}, I_{\max}]$, while K is still made as large as possible so to minimize the difference between bounds obtained from π_f compared to π_i. In the algorithms for the pricing problem in this chapter, we also use the conversion in equation (4.1) and pick K as in equation (4.2), where we used 64-bit integers and pick $I_{\min}$ and $I_{\max}$ accordingly.

4.1.2 Stabilization and early branching

Held et al. use a technique to reduce dual weights in their branch-and-price algorithm [36]. After solving the Restricted Master Problem, in pricing we can consider the lower bound (2.4). If during pricing we can prove that no column y exists such that $W(y) > \frac{z_{RMP}}{\lfloor z_{RMP} \rfloor}$, then we know that the integral bound $\lceil z_{RMP} \rceil$ will remain unchanged for the rest of the column generation. This can be seen as an early branching procedure. However, instead of increasing the lower bound for the maximum weight stable set procedure, we can also simply lower the dual weights

π, the objective coefficients of the pricing problem, by a total amount of $\frac{z_{RMP}}{\lfloor z_{RMP} \rfloor} - 1$. This can also be done effectively in exact arithmetic [36]. Held et al. investigated two different methods that decide which dual weights are reduced. Uniform rounding reduces all dual weights by an equal amount, whereas in *neighbourhood rounding* a vertex $v \in V$ is picked such that the dual weights are only reduced for vertices in $v \cup N(v)$. Both methods are shown to be effective in reducing the number of pricing algorithm calls [36].

Gualandi and Malucelli [30] also use a method that imposes a stronger lower bound on the maximum weighted stable set problem. They base their bound on knowledge of a lower bound κ of $\chi(G)$ and the lower bound given by (2.4). They first solve the decision problem to find a column y such that $W(y) > \frac{z_{RMP}}{\kappa}$. If one finds such a column y, then by (2.4) the lower bound from this pricing iteration cannot improve on κ. In this scenario, it can thus be unrewarding to solve the pricing problem to optimality as it does not improve the lower bound. In the case that no column y exists such that $W(y) > \frac{z_{RMP}}{\kappa}$, Gualandi and Malucelli simply fall back to solving the maximal weight stable set problem with the original lower bound of 1. They show that it can be valuable to solve the pricing problem to optimality in this case, as it allows them to infer a stronger lower bound using (2.4). This bound still comes at a cost, as solving the full pricing problem to optimality can be expensive, particularly for sparser and larger graphs.

Although early branching is effective and reduces the number of columns that needs to be generated, it is not very suitable when combined with cutting planes. One can still call separation procedures on non-optimal LP points, but rows may be generated which are not invalid for the LP optimum. Additionally, no or few cuts may be found compared to running these separation procedures at the LP optimum [49]. For the purpose of this research, we thus decided not to use early branching, and rather get a better idea of the strength of the cutting planes in isolation.

4.2 An algorithm for the modified pricing problem

In this section we will present a branch-and-bound algorithm for the modified pricing problem (3.2a)–(3.2c). Here, we only consider the rank-1 cuts, as we unfortunately had no time to extend the pricing algorithm to the higher rank results from Section 3.3. This algorithm is essentially identical to that of Held et al. [36], with the major difference being that the current algorithm has been adapted to take cutting planes into account. First, we will introduce some methods which can be used to bound the modified pricing problem on subgraphs in Section 4.2.1. Then, we will use these bounds to introduce a branch-and-bound algorithm in Section 4.2.2.

4.2.1 Bounds for the modified pricing problem

We consider the modified pricing problem for a subset of the vertices $V' \subseteq V$. Here, we restrict ourselves to the induced graph $G[V'] = (V', E')$, where $E' = \{(u, v) \in E : u, v \in V'\}$. Let us define $W(V')$ to be the total pricing weight of V'. More concretely, we have the following definition, which is simply equal to the objective as defined in equation (3.2a). Note that the dual values π and μ are non-negative.

$$W(V') = \sum_{v \in V'} \pi_v y_v + \sum_{H \in \mathcal{H}} \mu_H \left\lfloor \frac{\sum_{v \in V'} \rho_H^v y_v}{k_H} \right\rfloor \tag{4.3}$$

First we will introduce an upper bound procedure, which is done by finding a feasible solution to the dual problem and employing weak duality. Consider the formulation formed by (3.3a)–(3.3c) with (3.3e)–(3.3g). We can strengthen the edge inequalities (3.3c) by replacing them with the stronger clique inequalities. Let $\mathcal{Q}$ be the set of all maximal cliques of $G[V']$. Then we replace the inequalities (3.3c) with the following set of inequalities:

$$\sum_{v \in Q} y_v \leq 1 \qquad \forall Q \in \mathcal{Q} \tag{4.4}$$

Note this is still a valid formulation of the modified maximum weight stable set problem for $G[V']$, as it still ensures each clique can have at most one vertex in the optimal stable set. We consider then the linear relaxation in the system (4.5a)–(4.5e), where we divide the inequality (3.3g) by k_H for reasons which shall later become apparent. Note the upper bounds $y_v \leq 1$ can be left out as they are dominated by the clique inequalities (4.4)

$$\text{Maximize} \quad \sum_{v \in V'} \pi_v y_v + \sum_{H \in \mathcal{H}} \mu_H z_H \tag{4.5a}$$

$$\text{subject to} \quad \sum_{v \in Q} y_v \leq 1 \qquad \forall Q \in \mathcal{Q} \tag{4.5b}$$

$$y_v \geq 0 \qquad \forall v \in V' \tag{4.5c}$$

$$z_H \leq \frac{k_H - 1}{k_H} + \sum_{v \in V'} \frac{\rho_H^v y_v}{k_H} \qquad \forall H \in \mathcal{H} \tag{4.5d}$$

$$z_H \geq 0 \qquad \forall H \in \mathcal{H} \tag{4.5e}$$

Then, let us consider the linear relaxation of this formulation, and take its dual. By weak duality, any valid solution to the dual of the linear relaxation is an upper bound to the objective value of the linear relaxation in (4.5a) [60].

Let λ_Q be the dual value corresponding to an inequality from (4.5b), and let ν_H be a dual value corresponding to an inequality in (4.5d). The dual of the linear program (4.5a)–(4.5e) can then be stated as follows:

$$\text{Minimize} \sum_{Q \in \mathcal{Q}} \lambda_Q + \sum_{H \in \mathcal{H}} \nu_H \frac{k_H - 1}{k_H} \tag{4.6a}$$

$$\text{subject to} \sum_{Q \in \mathcal{Q}: v \in Q} \lambda_Q \geq \pi_v + \sum_{H \in \mathcal{H}} \nu_H \frac{\rho_H^v}{k_H} \qquad \forall v \in V' \tag{4.6b}$$

$$\nu_H \geq \mu_H \qquad \forall H \in \mathcal{H} \tag{4.6c}$$

$$\lambda_Q \geq 0 \qquad \forall Q \in \mathcal{Q} \tag{4.6d}$$

For the dual program (4.6a)–(4.6d), we can argue that in an optimal solution $\nu_H = \mu_H$ for all $H \in \mathcal{H}$, as we have $\mu_H \geq 0$, and increasing ν_H in inequality (4.6b) can only increase λ_Q, which also has a positive objective coefficient. More formally, assume a valid optimal solution exists for which $\nu_H > \mu_H$ for some $H \in \mathcal{H}$. By setting $\nu_H = \mu_H$, the solution must still be feasible, as this can only increase the slack of inequalities (4.6b), and as the objective coefficient $\frac{k_H - 1}{k_H}$ is always positive, this violates the assumption that our solution was optimal. Thus, a solution can only be optimal if $\nu_H = \mu_H$.

After substituting $\nu_H = \mu_H$, we are simply left with a weighted clique covering problem with variables λ_Q and weights on the vertices as given by inequalities (4.6b). We find feasible solutions to this problem by employing a greedy algorithm for this clique covering problem. As mentioned before, any feasible solution provides a feasible upper bound on $W(V')$. Note that by dividing by k_H, the contribution of the cutting planes is scaled in the objective (4.6a) and in inequalities (4.6b), reduces the impact of the cutting plane weights.

Next, we formulate two lemmas that aid us in further proofs and algorithm development by providing simple bounds on a subset of vertices.

Lemma 4.2.1 (Upper bound, subadditivity)**.** *Given two sets $A \subseteq V$ and $B \subseteq V$, we have $W(A) + W(B) \geq W(A \cup B)$.*

Proof.

$$
\begin{aligned}
W(A \cup B) &= \sum_{v \in A \cup B} \pi_v + \sum_{H \in \mathcal{H}} \mu_H \left\lceil \frac{\sum_{v \in A \cup B} \rho_H^v y_v}{k_H} \right\rceil \\
&= \sum_{v \in (A \setminus B)} \pi_v + \sum_{u \in B} \pi_u + \sum_{H \in \mathcal{H}} \mu_H \left\lceil \frac{\sum_{v \in A \cup B} \rho_H^v y_v}{k_H} \right\rceil \\
&\leq \sum_{v \in A} \pi_v + \sum_{u \in B} \pi_u + \sum_{H \in \mathcal{H}} \mu_H \left\lceil \frac{\sum_{v \in A \cup B} \rho_H^v y_v}{k_H} \right\rceil \\
&\leq \sum_{v \in A} \pi_v + \sum_{u \in B} \pi_u + \sum_{H \in \mathcal{H}} \mu_H \left\lceil \frac{\sum_{v \in A} \rho_H^v y_v}{k_H} + \frac{\sum_{v \in B} \rho_H^v y_v}{k_H} \right\rceil \\
&\leq \sum_{v \in A} \pi_v + \sum_{u \in B} \pi_u + \sum_{H \in \mathcal{H}} \mu_H \left(\left\lceil \frac{\sum_{v \in A} \rho_H^v y_v}{k_H} \right\rceil + \left\lceil \frac{\sum_{v \in B} \rho_H^v y_v}{k_H} \right\rceil \right) \\
&= \sum_{v \in A} \pi_v + \sum_{H \in \mathcal{H}} \mu_H \left\lceil \frac{\sum_{v \in A} \rho_H^v y_v}{k_H} \right\rceil + \sum_{u \in B} \pi_u + \sum_{H \in \mathcal{H}} \mu_H \left\lceil \frac{\sum_{v \in B} \rho_H^v y_v}{k_H} \right\rceil \\
&= W(A) + W(B)
\end{aligned}
$$

□

Lemma 4.2.2 (Lower bound)**.** *Given two subsets $A \subseteq V$ and $B \subseteq V$, we have $W(A \cup B) \geq \sum_{v \in (A \setminus B)} \pi_v + W(B)$.*

Proof.

$$
\begin{aligned}
W(A \cup B) &= \sum_{v \in A \cup B} \pi_v + \sum_{H \in \mathcal{H}} \mu_H \left\lceil \frac{\sum_{v \in A \cup B} \rho_H^v y_v}{k_H} \right\rceil \\
&= \sum_{v \in (A \setminus B)} \pi_v + \sum_{u \in B} \pi_u + \sum_{H \in \mathcal{H}} \mu_H \left\lceil \frac{\sum_{v \in A \cup B} \rho_H^v y_v}{k_H} \right\rceil \\
&\geq \sum_{v \in A \setminus B} \pi_v + \sum_{u \in B} \pi_u + \sum_{H \in \mathcal{H}} \mu_H \left\lceil \frac{\sum_{v \in B} \rho_H^v y_v}{k_H} \right\rceil \\
&\geq \sum_{v \in A \setminus B} \pi_v + W(B)
\end{aligned}
$$

□

Note that both lemmas hold for disjoint sets in particular. If Lemma 4.2.2 is invoked with two disjoint subsets $A, B \subseteq V$, this then implies that $W(A \cup B) \geq \sum_{v \in A} \pi_v + W(B)$.

4.2.2 A combinatorial branch-and-bound algorithm for the modified pricing problem

Based on the bounds derived in Section 4.2.1 we can construct a combinatorial branch-and-bound algorithm. The branch-and-bound algorithm works with 3 sets of vertices. We have F, the set of free vertices that we can still pick from, and S, the current stable set. A set of excluded vertices X is also used; these are vertices that are explored by different subtrees, and do not need to be checked again. In order to generate valid stable sets, the algorithm keeps some invariants on these sets. First, S, F and X are always disjoint so that $S \cap F = S \cap X = F \cap X = \emptyset$. Secondly, there is no edge from any vertex in S to any vertex in F, $(S \cap N(f) = \emptyset, \forall f \in F)$, so that any vertex $f \in F$ may be added to S and S will remain a valid stable set. These invariants are ensured by updating the sets S, F and X whenever branching is done. Algorithm 3 shows a (simplified) version of the implemented algorithm. It uses two pruning rules to prune redundant recursive calls, and 3 branching rules are scanned in order to select branching vertices.

Algorithm 3 Branch and Bound algorithm for the modified MWSS

```
1: function ModifiedMWSS(S,F,X)
2:     LB = max(LB, W(S))
3:     if F = ∅ and W(S) > 1 then
4:         SaveSolution(S)
5:         return
6:     end if
7:     if ∃x ∈ X such that π_x ≥ W((S ∪ F) ∩ N(x)) then
8:         return
9:     end if
10:    C = WeightedCliqueCover(G[F])
11:    if Weight(C) ≤ LB − W(S) then
12:        return
13:    end if
14:    B = {b_1, b_2, ..., b_p} = BranchVertices(C,S,F,X)
15:    for i = 1 to p do
16:        F_i = F \ (N(b_i) ∪ {b_i, b_{i+1}, ..., b_p})
17:        ModifiedMWSS(S ∪ {b_i},F_i, X)
18:        X = X ∪ {b_i}
19:    end for
20: end function
21: ModifiedMWSS(∅,V,∅)
```

The validity of the first pruning rule (lines 7-9 of Algorithm 3) can be seen as an application of Theorem 4.2.3.

Theorem 4.2.3 (Pruning rule)**.** *If there exists a vertex $x \in X$ such that $\pi_x \geq W((S \cup F) \cap N(x))$, then from any solution of the current subtree $S' \subseteq S \cup F$ we can construct a solution $S'' = S' \setminus N(x) \cup \{x\}$ such that $W(S'') \geq W(S')$.*

Proof.

$$\begin{aligned} W(S'') &= W(\{x\} \cup (S' \setminus N(x))) \\ &\geq \pi_x + W(S' \setminus N(x)) \\ &\geq \pi_x + W(S') - W(S' \cap N(x)) \\ &\geq \pi_x + W(S') - W((S \cup F) \cap N(x)) \\ &\geq W(S') \end{aligned}$$

In the first two steps Lemma 4.2.2 and Lemma 4.2.1 are used. Then, we use the definition of S'. The last step simply uses the condition of the theorem. □

The first branching rule also uses intuition from the above; for each $x \in X$ where $\pi_x \geq W(S \cap N(x))$, we must use some $f \in (F \cap N(x))$ in the maximal stable set $S' \subseteq (S \cup F)$, as otherwise we would have $\pi_x \geq W(S \cap N(X)) = W((S \cup F) \cap N(x))$ which would imply that taking the branch would not improve the current stable set by the above pruning rule. Thus we can branch on all branching candidates $f \in F \cap N(x)$ if $\pi_x \geq W(S \cap N(x))$.

The second pruning rule (lines 10-13 of Algorithm 3) tries to provide an upper bound for any stable set $S' \subseteq S \cup F$ of the current subtree. We know that $W(S') \leq W(S) + W(F)$, and by computing an upper bound to $W(F)$ using a greedy clique covering heuristic, we can compute an upper bound $W(S) + \bar{W}(F)$ on the weight of the solutions that can be obtained from the current subtree. If this upper bound is smaller than or equal to the lower bound then we know the current subtree cannot yield any improving solutions.

Algorithm 4 A greedy algorithm for the weighted clique cover problem

```
1: function WeightedCliqueCover(G(V, E), vertex weights w)
2:     W̄(V) = 0
3:     while ∃v ∈ V : w_v > 0 do
4:         v = arg min_{u∈V}{w_u : w_u > 0}
5:         Greedily build the clique Q ⊆ {u ∈ N(v) ∪ {v} : w_u > 0}
6:         Set W̄(V) = W̄(V) + w_v
7:         Set w_u = w_u − w_v, ∀u ∈ Q
8:     end while
9: end function
```

A greedy algorithm to compute the weighted clique covering bound is presented in Algorithm 4, and is identical to the one used by Held et al. [36]. We order by weight, preferring to pick vertices with smaller weights first.

The second branching rule is derived from the second pruning rule: instead of computing the full weight of $W(F)$, we can also stop the greedy covering algorithm early with a covering C when $W(S) + \bar{W}(F) \leq LB$ is first violated (not performing the weight reduction steps for the 'violating' vertex). Then, all vertices $\{f \in F : w_f > 0\}$ can become the branching vertices, as we need to add at least one of them in order to ensure that the clique covering bound does not prune away the current subtree. This branching rule is also used by Held et al. [36] and other combinatorial algorithms [2, 42, 67], and is key to the performance of the algorithm. It was first proposed by Balas and Xue [2].

Theorem 4.2.4 (Third branching rule)**.** *If a vertex $f \in F$ exists such that $\pi_f \geq W(F \cap N(f))$, then for any stable set $S' \subseteq S \cup F$ with $f \notin S'$ we can construct a stable set $S'' = S' \setminus N(f) \cup \{f\} \subseteq (S \cup F)$ such that $W(S'') \geq W(S')$*

Proof. Suppose there is a maximal stable set $S' \subseteq (S \cup F)$ such that $f \notin S'$. Then, we can consider the set $S'' = \{f\} \cup (S' \setminus N(f))$. Then, we can claim the following:

$$\begin{aligned} W(S'') &= W(\{f\} \cup (S' \setminus N(f))) \\ &\geq \pi_f + W(S' \setminus N(f)) \\ &\geq \pi_f + W(S') - W(S' \cap N(f)) \\ &\geq W(S') + \pi_f - W((S \cup F) \cap N(f)) \\ &= W(S') + \pi_f - W(F \cap N(f)) \\ &\geq W(S') \end{aligned}$$

The first and second steps are done by applying Lemma 4.2.2 and Lemma 4.2.1. In the third step, we use that $S \cap N(f) = \emptyset$ in our algorithm, by definition. The last step uses the condition of the theorem.

□

The last branching rule is formalized in Theorem 4.2.4. Here, we can see that we simply create a single branch for a vertex $f \in F$ for which $\pi_f \geq W(F \cap N(f))$, which by Theorem 4.2.4 always contains a solution better or equal than that of all other branches.

The set of branching vertices is selected based on the size of the set, as using fewer branching vertices means we can typically terminate the algorithm more quickly. Thus, the third branching rule is preferred over the first, which is (most of the time) preferred over the second branching rule. For the first branching rule, the vertices are sorted by decreasing degree, whilst for the second branching rule, they are sorted decreasing by the remaining weight in the clique covering algorithm.

Before running Algorithm 3 we can reduce the size of the graph by removing any vertex $v \in V$ for which $\pi_v = 0$ and $\{H \in \mathcal{H} : \mu_H > 0, \rho_H^v > 0\} = \emptyset$ hold, as these vertices cannot contribute to the weight of the stable set. Stable sets found can be maximized over these zero weight vertices arbitrarily, as is also discussed in Section 2.1.2.

Algorithm 3 can be terminated whenever a stable set with $W(y) > 1$ is found. This is crucial for its performance, as this is often much cheaper than solving the pricing problem to optimality [36]. However, only adding a single column is not always productive, as one may want to generate multiple at once so that fewer calls to Algorithm 3 are necessary. Additionally, if Algorithm 3 is close to proving optimality of the found column y, then it could be worthwhile to wait, so that the lower bound from (2.4) can be used. Thus, we adapted the following stopping condition; whenever a stable set y with $W(y) > 1$ is first found, we terminate Algorithm 3 if $10|V|$ more calls to function *ModifiedMWSS* are made after finding this stable set y. In practice, this gave us a good combination of giving Algorithm 3 enough chances to find better columns and perhaps prove the lower bound in equation (2.4), but with the benefit of relatively early termination in most cases.

4.3 Implementation

4.3.1 Column Initialization

In order to find an initial set of columns, the tabu search algorithm as described by Galinier and Hertz [24] was used. The tabu coloring algorithm for a k coloring starts with an arbitrary

distribution of colors, and exchanges the colors of two vertices if this leads to fewer *violated* edges, which have the same color on both ends. A tabu list of forbidden exchanges is kept which aims to forbid short term cycling and makes it possible to escape local optima. We used the same parameters as in the open-source SCIP implementation [26]; using the notation from [24], we set $L = 50$ and $\lambda = 0.9$. Then, we search an initial set of columns by first using a greedy algorithm, which produces a k-coloring. Then, the tabu search algorithm is run with an iteration limit of 10^5 exchanges for successively smaller k, until no coloring is found within the iteration limit. The smallest coloring found is then used as the initial set of columns, after being greedily expanded using an arbitrary ordering to create maximal independent sets.

4.3.2 Greedy improvements

In Section 4.1, we mention the use of (1,2) and (2,k) swaps in order to improve solutions found using greedy methods. This can give a 'local' search like improvement. Although these work well when the pricing problem is linear, they cannot be used in the context where the weight of a set is nonlinear. However, we can linearize the weight of a vertex quite easily, by ignoring the ceiling operator in (3.2a). Then in this linear approximation, the weight of each vertex $v \in V$ can be defined as

$$w(v) = \pi_v + \sum_H \mu_H \frac{\rho_{H,v}}{k_H} \tag{4.7}$$

Held et al. [36] initialized the swapping procedure using greedy methods. We observed that the local search procedure initialized by greedy methods did not work well for most instances, only finding the very first columns after initialization of the column generation algorithm. Instead, we pass any solution found by Algorithm 3 into a local search algorithm which iteratively tries to improve the solution using these swapping procedures. The resulting column is added if it has negative reduced cost. Note that this is not guaranteed, as the linear approximation in equation (4.7) may cause us to find worse columns. However, this can potentially be a cheap way to find better columns quickly, which may reduce the number of calls to Algorithm 3.

Chapter 5

Branching

In Section 2.1.2 we explained the usage of the Zykov branching rule to create subproblems which are also graph-coloring problems. In this chapter, we will primarily focus on how to pick the two vertices u and v to branch on, and how this choice affects solution speed. Although this is not the primary goal of this research, branching is an important part of branch-price-and-cut algorithms. The importance of branching is often not discussed in existing papers, and no comparison of different branching strategies exists in the current literature. Thus, we decided to run a small experiment in order to decide which branching strategy is 'best'.

5.1 Branching strategy

A few different methods to pick the branching vertices have been suggested in a literature. Mehrotra and Trick [45] used the following strategy; from the column of the most fractional stable set S, pick some vertex $u \in S$. Then, from a second column S' pick a second vertex $v \in (S' \cup S) \setminus (S' \cap S)$. This is the most popular branching rule, and several papers have used this or small variations on it in their research. Malaguti et al. also used this rule [44], and Gualandi and Malucelli [30] use the vertex degree to break ties when picking the second vertex v.

Held et al. [36] suggest picking the edge using the following formula, which tries to compute the degree to which two stable sets are considered to be the 'same' color in the current LP solution. Consider arbitrary vertices $u \in V$, $v \in V \setminus \{v\}$ such that $(u, v) \notin E$.

$$p(u,v) = \frac{\sum\limits_{S \in \mathcal{S}: u,v \in S} x^*_S}{\frac{1}{2}(\sum\limits_{S \in \mathcal{S}: u \in S} x^*_S + \sum\limits_{S \in \mathcal{S}: v \in S} x^*_S)} \tag{5.1}$$

Note that $p(v, w) \in [0, 1]$ is well defined. A factor of 55% was determined empirically to be most practical in Held's application.

$$\arg \min_{u,v \in V:(u,v) \notin E} |p(u,v) - 0.55| \tag{5.2}$$

A different suggestion was done by Hansen et al. [32]. They pick u and v by maximizing $|N(u) \cap N(v)|$.

$$\arg \max_{u,v \in V:(u,v) \notin E} |N(u) \cap N(v)| \tag{5.3}$$

A rationale for this branching rule is that in G_{DIFFER} when the edge (u, v) is added, triangles are formed with all vertices in $N(u) \cap N(v)$, thus creating $|N(u) \cap N(v)|$ new triangles in the graph (see Figure 2.1). As the stable set formulation implicitly satisfies the clique inequalities belonging to these cliques, one could conjecture that more stable sets are eliminated and a better lower bound is achieved for the subproblems G_{DIFFER} when $|N(u) \cap N(v)|$ is large.

Although the above branching edge selection methods are the ones we could find in literature, we have tried a few new ones when testing. First, considering Hansen's rule in equation (5.3), the rationale we provide only really holds for the branch G_{DIFFER}. Considering the interpretation of G_{SAME} where the vertex is not removed (see Figure 2.1), we see that $|(N(u) \cup N(v)) \setminus (N(u) \cap N(v))|$ edges are added to G_{SAME} after branching. Combining the two rules, it makes sense to then also consider $|N(u) \cup N(v)|$ as a metric which acts somewhat favorably for both branches. This strategy is presented by equation (5.5).

$$\arg \max_{u,v \in V:(u,v) \notin E} |N(u) \cup N(v)| \tag{5.4}$$

We also introduce a variant to this which weighs slightly more in favor of the G_{DIFFER} branch, using $|N(u)| + |N(v)| = |N(u) \cup N(v)| + |N(u) \cap N(v)|$ as its metric. A new triangle, forcing three vertices to belong to different stable sets, may be deemed as a 'stronger' property than just adding a single edge, thus not making evenly weighing them equally very logical.

$$\arg \max_{u,v \in V:(u,v) \notin E} |N(u)| + |N(v)| \tag{5.5}$$

Additionally, we were curious about using more information from the LP. The most fractional variable approach from Mehrotra and Trick [45] and the approach from Held et al. (equation (5.2)) both use information from the primal LP. However, information from the dual values π_v from the constraints in equation (5.6) also maps directly to the vertices of the graph. Thus, we also tried a rule which simply maximizes $\pi_u + \pi_v$.

$$\arg \max_{u,v \in V:(u,v) \notin E} \pi_u + \pi_v \tag{5.6}$$

Additionally, a variant of strategy (5.5) was introduced which breaks ties based on the value of $\pi_u + \pi_v$.

$$\arg \max_{u,v \in V:(u,v) \notin E} |N(v)| + |N(w)| + \frac{\pi_u + \pi_v}{\sum_{w \in V} \pi_w} \tag{5.7}$$

An important note is that all of the above branching strategies, with exception of Mehrotra's strategy [45], may not violate the LP. More elaborately, when considering an edge (u, v) they do not explicitly guarantee that there exist at least two fractional stable sets $S, S' \in \mathcal{S}$ for which $u \in S$, $v \in S \Delta S'$ and $x^*_S > 0, x^*_{S'} > 0$. If no stable sets satisfying these conditions exist in the problem, then the current LP solution is not invalidated in any of the two branches as all variables $S \in \mathcal{S}$ with $x^*_S > 0$ are transferred to both new branches. This solution is then again immediately optimal, as adding new edges to a graph G cannot decrease $\chi_f(G)$, and we obtain the exact same bound as before, making this a 'useless' branching step. Fortunately, all procedures can be adjusted to only consider all the non-edges $\bar{E}'$ for which at least one pair exists, e.g. $\bar{E}' = \{(u, v) \notin E$ such that $\exists S, S' \in \mathcal{S} : x^*_S > 0, x^*_{S'} > 0, u \in S, v \in S \Delta S'\}$.

5.2 Experiments

In order to get an idea of the strengths and weaknesses of all existing and suggested branching rules, we ran experiments on a few select instances. Our goal here is not to determine the 'best'

branching rule, but rather to get an idea of the performance of each branching rule in different circumstances, so only a few instances were considered. Here, we use the same configuration for testing as described at the start of Chapter 6.

We computed the results on a select few instances which are all reported to have branch-and-bound trees of non-trivial size in the literature using branch-and-price algorithms. These are the instances *myciel4*, *myciel5*, *queen9_9*,*DSJC125.9*,*DSJC250.9* and *4-FullIns_4*. These graphs have varying densities and difficulties. A further explanation of the origin and the structure of these instances can be found in Section 6.2.

We denote the different branching strategies as follows. The most fractional variable strategy from Mehrotra and Trick is denoted by FRAC. The strategy by from Held et al from equation (5.2) is denoted by HELD. Hansen's strategy (equation (5.3)) is denoted by IS, and the union strategy from equation (5.4) is denoted UN. The combined strategy from equation (5.5) is denoted ISUN, with the version with dual tie breakers in equation (5.7) denoted as ISUND. The dual strategy in equation (5.6) is denoted by DUAL. In order to compare the branching strategies, we also consider a random branching strategy, which picks $u, v \in V$ with uniform probability, subject to the constraint that $(u, v) \notin E$. This strategy denoted as RAND.

In this experiment, the branch-and-bound nodes are processed in order of lower bound, processing nodes with the smallest lower bound first. This is done, so that even when the branch-and-bound process terminates early, we can compare the obtained lower bounds and argue about the quality of a strategy for that particular instance. The default branching rules of SCIP do not necessarily process the nodes in this order, which may paint an inaccurate picture of the strength of the bounding strategy.

For these tests, a time limit of 1 hour was used. If the instance is not solved within this limit, this is denoted by '*tl*' in the time column; otherwise, the time is reported in seconds. Additionally, the best lower bound $\underline{\chi(G)}$ and the number of solved branch-and-bound nodes are reported. The most important metric for measuring the effectiveness of the branching rules is the number of branch-and-bound nodes needed to solve a problem. The strategy with the least number of branch-and-bound nodes is highlighted in bold. Table 5.1 shows the outcome of this experiment.

Instance	$\chi(G)$	FRAC			HELD			DUAL			RAND		
		time	$\underline{\chi(G)}$	nodes	time	$\underline{\chi(G)}$	nodes	time	$\underline{\chi(G)}$	nodes	time	$\underline{\chi(G)}$	nodes
myciel4	5	0.25	5	241	1.61	5	1529	0.09	5	95	0.7	5	863
myciel5	6	*tl*	5	241455	*tl*	4.22	24503	55.71	6	19457	*tl*	4.31	34501
4-FullIns_4	8	0.21	8	293	*tl*	7	240204	0.05	8	21	120.98	8	59249
queen9_9	10	4.06	10	**19**	5.35	10	29	6.42	10	43	5.73	10	35
DSJC125.9	44	1.16	44	**75**	1.14	44	91	4.30	44	449	1.33	44	93
DSJC250.9	72	1309.45	72	44407	1193.76	72	37979	839.68	72	20869	809.58	72	27821

Instance	$\chi(G)$	IS			UN			ISUN			ISUND		
		time	$\underline{\chi(G)}$	nodes	time	$\underline{\chi(G)}$	nodes	time	$\underline{\chi(G)}$	nodes	time	$\underline{\chi(G)}$	nodes
myciel4	5	0.10	5	93	0.08	5	77	0.10	5	77	0.08	5	**65**
myciel5	6	48.31	6	18295	5.98	6	3711	4.83	6	3565	4.35	6	**3531**
4-FullIns_4	8	0.06	8	41	0.05	8	19	0.05	8	**13**	0.05	8	**13**
queen9_9	10	4.47	10	**19**	4.99	10	27	4.29	10	23	4.52	10	23
DSJC125.9	44	1.82	44	185	6.73	44	535	3.41	44	371	4.19	44	475
DSJC250.9	72	720.17	72	**20045**	*tl*	70.83	16816	966.53	72	26731	883.10	72	23965

Table 5.1: Branch-and-bound results for 8 different branching strategies on a select few instances

Note that the implementations of the different branching rules are not particularly efficient, which can explain the difference in timing for different branching rules, particularly for the smaller instances. However, most of the above branching rules can be implemented efficiently by computing the needed information once and updating it for every branch-and-bound node, or by keeping e.g an index of all sets which contain a pair of vertices. In the tested implementations, a lot of information is recomputed at each branch-and-bound node, such as the set of possible edge branching candidates. Thus, one should not consider the time, but primarily the number of used branch-and-bound nodes as the main metric here.

A few interesting observations can be made from Table 5.1. On the three sparse and smaller instances *myciel4*, *myciel5* and *4-FullIns_4*, the existing strategies HELD and FRAC perform poorly, creating much larger branch-and-bound trees than the neighborhood strategies IS,UN,ISUN and ISUND. The last three of these strategies perform particularly well, needing significantly smaller branch-and-bound trees for all 3 instances. The DUAL strategy is also viable, having results highly similar to the IS strategy for the sparser instances.

All tested methods perform well on the symmetric and somewhat dense *queen9_9* instance, needing only a few branch-and-bound nodes. Interesting to note is that for this highly symmetric instance, DUAL seemed to perform comparatively worse, being outperformed by the random branching strategy.

For the two dense instances *DSJC125.9* and *DSJC250.9* we obtain varying results. For the larger instance *DSJC250.9*, the branch-and-bound tree is smallest using the IS or DUAL strategy. Interestingly, the two strategies from literature FRAC and HELD are both outperformed by random branching, but there is no order-size difference in branch-and-bound tree. The high density of this instance may explain this; there is comparatively fewer vertex pairs which can be branched on, of which only some are violated by the LP solution. For the smaller instance DSJC125.9 these two strategies interestingly perform best using only 75 and 91 branch-and-bound nodes, respectively.

One surprising result, is that the branching strategy HELD (Equation (5.2)) seems to perform very poorly, even being outperformed by random branching on some instances. This does not match the results from Held et al. [36], who reported needing only 3225 branch-and-bound nodes for the instance DSJC250.9, which is significantly lower than any of the strategies tested in Table 5.1. A potential explanation for this result is that Held et al. perform a dual stabilization procedure which also allows for early branching (see Section 4.1 from [36]), whereas we do not. One could also attribute this observed difference to the LP solver or differences between the generated columns when solving the pricing problem, or the order in which branches were taken.

The results for the FRAC strategy are similar to the results found in [30, 44, 45], with the size of the branch-and-bound tree never varying by more than an order. The results from Hansen et al. [32] mostly match our results for the IS strategy, with minimal differences in the number of branch-and-bound nodes. The only significant difference is for *myciel5*, where they report needing only 4290 branch-and-bound nodes, compared to the 18295 needed in our case. Again, differences in the LP solver or in the solution of the pricing problem could explain this discrepancy.

Table 5.2 displays the percentage of interior branch-and-bound nodes for which the last taken branching decision is of type SAME. In a leaf node of the branch-and-bound node with graph G', it is proven for the RMP that $\lceil \chi_f(G') \rceil = \chi(G)$. One can argue that if the percentage of nodes of a type is higher in the interior, the branching on that type is weaker relative to

Instance	FRAC	HELD	DUAL	RAND	IS	UN	ISUN	ISUND
myciel4	42.86	42.99	36.96	41.16	46.67	35.14	35.14	35.48
myciel5	50.52 *	50.00*	38.05	49.98*	46.84	35.44	36.50	36.51
4-FullIns_4	53.79	44.65*	44.44	34.30	31.58	12.50	0	0
queen9_9	37.50	46.15	10.0	18.75	25.00	8.33	20.00	0
DSJC125.9	50.00	47.73	23.32	28.89	31.87	19.17	30.98	24.15
DSJC250.9	43.11	47.94	21.66	28.91	42.28	39.38*	36.29	36.25

Table 5.2: The percentage of interior branch-and-bound nodes which are of type SAME, excluding the root node

the other branching type, as it significantly increases χ_f less often to conclude the subtree is optimal. Note that runs highlighted with a star did not finish within the time limit. As a result, their percentage may be off from the percentage obtained when this instance is solved to optimality.

A first observation is that the percentages in Table 5.2 are almost never larger than 50%; this indicates that more of the interior nodes are DIFFER. This observation matches the reasoning for the ISUN strategy presented in the previous section, which argues based on the number of edges added in the graph that the SAME branch should be relatively stronger; apparently, SAME branches are more likely to become leaves, and thus must have stronger lower bounds.

We can also argue the UN, ISUN and ISUND strategies indeed select stronger SAME branches, as for both *myciel4* and *myciel5* less interior nodes are of type SAME than for the remaining 5 strategies. Similarly, one could argue that the IS strategy somewhat favors the 'DIFFER' branch as it typically has more SAME nodes in its interior than the RAND strategy.

Lastly, we can also observe that the number of branch-and-bound nodes is not tightly bound to the percentage of interior SAME nodes. For the dense graphs DSJC125.9 and DSJC250.9, various strategies can create branch-and-bound trees of similar size, but the interior of this trees can contain vastly different percentages of SAME nodes.

In the rest of this research the strategy ISUN (5.5) was used, as it is a balanced branching rule which performs well for both sparse and dense instances.

Chapter 6

Experiments

In this chapter, the set-up and outcomes of experiments to test the methodologies from Chapters 2–5 are described. First, we will describe the configurations tested and relevant details in Section 6.1. Section 6.2 contains descriptions of the tested DIMACS instances. Finally, in Section 6.3 we describe the obtained results and compare the described branch-cut-and-price algorithm with a branch-and-price algorithm.

6.1 Implementation

The methodology and algorithms as discussed in Chapters 2 – 5 are implemented in C++ using the SCIP framework [26]. The LinBox library [22, 29] was to solve systems of linear equations over finite fields in order to separate mod-k cuts. Gurobi [31] was used as the Linear Programming Solver. All benchmarks have been done on a 3.0 GHz Intel Xeon Gold 5217 CPU on a system with 64 Gb of RAM memory. The code was compiled using gcc and the "-O3 -march=native" flags.

In order to test the effect of mod-k cutting planes and odd-cycle cutting planes, we run several experiments. We were unable to implement the results from Section 3.3 due to a lack of time, so only rank-1 cuts will be considered in the experiments done.

One of the crucial parameters for separating procedures is deciding when they are called. Separating cuts at every branch-and-bound node may be too costly computationally, and drive up the computation time. Additionally, the larger number of cutting planes may slow down the LP solver and/or the pricing algorithm. We tested three main configurations; separating cutting planes at every branch-and-bound node, separating cutting planes only at the root node and no separation. For the experiments which separate cutting planes at the root node, we tested mod-k and odd-cycle cuts also separately in order to get an better idea of their respective strengths. This yields the 5 configurations C_1-C_5 as shown in Table 6.1.

Configuration	C_1	C_2	C_3	C_4	C_5
mod-k cutting planes	no	yes	no	yes	yes
odd-cycle cutting planes	no	no	yes	yes	yes
separated at	never	root node	root node	root node	every node

Table 6.1: Configurations tested

In their tests for the odd-cycle cutting planes, Hansen et al. [32] separate and add cutting

planes only when a branch-and-bound node is 'close' to being solved. For a given branch-and-bound node, they only separate odd-cycle cutting planes if the difference between the local bounds UB_n, LB_n for a branch-and-bound node n is smaller than 1.1, e.g. $UB_n - LB_n \leq 1.1$. Here, we do not adapt such a choice, but are rather more interested in the strength of the cuts at the root node. The rationale for this is twofold. First, we have observed large branch-and-bound trees, even for relatively 'small' sparse instances such as *myciel5* in Chapter 5. Secondly, the time to solve the root branch-and-bound node can become very large for more difficult and unsolved instances, sometimes taking weeks to solve [36]. These two arguments combined make it unlikely that branch-and-bound finds significant improvements for these difficult instances, and make adding cutting planes at the root node a more attractive alternative.

In the case both (maximally violated) mod-k cutting planes and odd-cycle cuts were used, we first run mod-k cut separation. As this is a more general class of cutting planes, we expect that this separator might find stronger cutting planes. If any mod-k cutting planes are found, the odd-cycle separation procedure is then skipped, and the Restricted Master Problem then solved again. Only if the mod-k separator finds no new cutting planes, we attempt to find odd-cycle cutting planes. The first reason for this is that mod-k cutting planes are more general and from our experience typically stronger (more elaboration below). The second reason to delay odd-cycle cuts is that after adding the 'weaker' odd-cycle cutting planes, the procedure to select k described in Section 3.1.3 seemed to be less effective. Fewer non-zero coefficients of the optimal LP solution vector x^* were close to 'nice' fractions such as $\frac{1}{2}$, and the found lower bounds were weaker in preliminary experiments as less mod-k cuts were separated.

The branching strategy used is the ISUN strategy from equation (5.5), as argued in Chapter 5 based on preliminary experiments.

6.2 Instances

Most of the experiments in the literature have been carried out using the DIMACS instances. These are a set of instances of various origins and difficulty. In this section we describe the characteristics and difficulty of each type of graph.

The *myciel* graphs are based on the Mycielski transformation. Mycielski graphs have clique number $\omega(G) = 2$, and the fractional coloring number $\chi_f(G)$ for these graphs has a known analytical expression [40]. A Mycielski graph M_m has $\chi(M_m) = m + 1$ and is thus $m + 1$ colorable. Mycielski graphs typically have low density and are some of the most difficult types of graph to color because of the large gap between $\chi_f(G)$ and $\chi(G)$ that they exhibit. This typically implies these graphs require relatively large branch-and-bound trees.

Insertion graphs (*n_Insertion_m*, *n_FullIns_m*) are created by taking Mycielski graphs M_m and inserting vertices and edges to keep the same density. Similar to Mycielski graphs, proving optimality can be difficult due to the large gap between $\chi_f(G)$ and $\chi(G)$, and these graphs are difficult to solve in practice.

Queen graphs (*queenm_n*) are graphs which model the n queens problem on an $m \times n$ chessboard. For $n \times n$ queen graphs it is known they are n-colorable if and only if $n \equiv 1 \mod 6$ or $n \equiv 5 \mod 6$ [14]. These graphs are challenging, particularly due to their large number of symmetries.

Another set of challenging instances are the DSJC graphs (*DSJC.n.p*). These are graphs

with n vertices, where for each pair of vertices $u, v \in V$, $P(\{(u,v) \in E\}) = p$ independently of other edges in G. This is also known as an Erdős-Rényi Graph. DSJR graphs (*DSJR.n.y*) are geometric graphs, which have n vertices which are randomly distributed over the unit square, with edges connecting two vertices if the resulting edge length is shorter than $0.y$. Graphs which end with a c are complemented.

Matrix partitioning graphs (ending with *GPIA*) are graphs related to matrix partitioning problems which try to find sparse Jacobian matrices. These are typically somewhat large and difficult to solve, but are often solved in the root node due to a small gap between $\chi_f(G)$ and $\chi(G)$.

Optical network graphs (starting with *wap*) are large and difficult instances based on real-life optical network design problems. These graphs have been found difficult to color due to the large number of columns needed to do so [36].

Latin squares instances (*qg.order*,*latin_square_10*) are based on latin square problems. These pose difficult coloring problems, although they typically have small gaps.

Leighton graphs (*le450_x*) are difficult graphs with coloring number $\chi(G) = x$. Column generation for these graphs typically *stalls*, but these graphs have the lower bound $\omega(G) = x$ which can be easily computed.

Flat graphs (*flat_n_c*) are difficult instances with n vertices which are colorable with c colors.

In addition to the above, we also test the four large and difficult instances *C1000.9*, *C2000.5*, *C2000.9* and *C4000.5* , which have (nearly) the same naming scheme and generation procedure as DSJC graphs.

There is also a number of easier graphs. These are nearly all solved with the initial set of columns, and Mehotra and Trick have shown their coloring numbers exactly in short periods of time [45]. In our research, these graphs are all solved by initial heuristics, and proven to be optimal in the root node. These graphs are:

- Register allocation graphs (ending with *.i.n* for $n = 1, 2, 3$) are derived from real-life register allocation problems
- Book graphs based on name occurences in popular books. (*anna*,*david*,*homer*,*huck*,*jean*)
- The two scheduling graphs *school1* and *school1_nsh*
- *miles* are graphs from geometric graphs based on real road networks
- The *games120* graph which is based on football teams, where an edge is inserted if two teams played each other.
- *mug* graphs are 4 colorable graphs with $\omega(G) = 3$.

Note that we only report our results on these easier instances for the branch-and-price algorithm (see Table 7.1 in the appendix) for comparison with the literature and to verify the correctness of the implemented code.

6.3 Results

In this section, we discuss the outcomes of the experiments as described in Section 6.1. The complete results for all instances using the C_1-C_5 configurations can be found in the appendix in Tables 7.1–7.5. Note that we leave out results for cutting planes on all instances that were solved in the root node by the branch-and-price algorithm, as cuts are not necessary to solve these instances. In the following sections, we will highlight some of the found results in more depth, and conclude the outcomes of our experiments based on them.

We denote the number of separated odd-cycle and mod-k cuts separated for a single instance by $|\mathcal{H}|_{\text{oc}}$ and $|\mathcal{H}|_{\text{modk}}$ respectively.

6.3.1 Branch-and-price

First, the branch-and-brice algorithm, configuration C_1, was run without cutting planes. A few new results were obtained, likely primarily due to the ISUN branching strategy we use in equation (5.5). In this paper, we compare our results with the existing branch-and-price implementations [30, 32, 36, 44, 45]. Results from other approaches are only included for the best Upper Bounds and for instances where $\chi(G)$ has been proven.

Table 6.2 shows all newly obtained results. Instances for which $\chi(G)$ was first proven using a branch-and-price algorithm have $\chi(G)$ highlighted in bold. If the lower bound was improved, but the instance was not yet solved to optimality, the lower bound is highlighted in bold. Our algorithm was for the first time able to solve all *FullIns* instances compared to other branch-and-price algorithms. For the instance *5-FullIns_4* optimality was proven for the very first time using any algorithm to our knowledge, solving this instance. Additionally, we compute a $\chi_f(G) = 389.6$ for the instance *C2000.9* for the first time, giving a strong lower bound of $\underline{\chi(G)} = 390$.

Instance	$\|V\|$	$\|E\|$	$\chi(G)$	time (s)	B&B nodes	$\underline{\chi(G)}$	Best UB	Found UB
1-FullIns_4	93	593	**5**	0.07	21	5	5	5
1-FullIns_5	282	3247	**6**	3.44	2025	6	6	6
1-Insertions_4	67	232	5 [46]	tl	78729	**4**	5 [46]	5
1-Insertions_5	202	1227	?	tl	504	**3.342**	6 [46]	6
2-FullIns_4	212	1621	**6**	0.1	21	6	6	6
2-FullIns_5	852	12201	**7**	5.06	3879	7	7	7
2-Insertions_4	149	541	?	tl	1563	**3.149**	5 [46]	5
3-FullIns_4	405	3524	**7**	0.06	19	7	7	7
3-FullIns_5	2030	33751	**8**	1.93	1475	8	8	8
3-Insertions_3	56	110	**4**	9.6	699	4	4	4
4-FullIns_4	690	6650	**8**	0.13	13	8	8	8
4-FullIns_5	4146	77305	**9**	5.73	3561	9	9	9
4-Insertions_3	79	156	**4**	437.77	2183	4	4	4
5-FullIns_4	1085	11395	**9**	0.11	15	9	9	9
C2000.9	2000	1799532	?	tl	2	**389.6**	400 [64]	446

Table 6.2: New results on the DIMACS instances using a branch-and-price algorithm

Using the branch-and-price algorithm, we observed that the branch-and-bound tree of the instances *myciel6* and *1-Insertions_4* were somewhat likely to terminate within a reasonable time. For both of these instances, the initial coloring heuristic is able to find the optimal solution, but they have a large gap between $\chi_f(G)$ and $\chi(G)$ which needs to be closed by branch-and-bound. Thus, we ran both these instances with a longer time limit of 24 hours. Both instances were

solved in the time limit; relevant results can be found in Table 6.3, which shows the number of branch-and-bound nodes and the time taken in minutes.

Instance	$\chi(G)$	time (m)	B&B nodes
myciel6	7	224	911149
1-Insertions_4	5	526	219567

Table 6.3: Results for the instances *myciel6* and *1-Insertions_4* using a time limit of 24 hours.

Note the large size of the branch-and-bound tree in both cases. Both instances were already solved by [30], but not using a branch-and-price algorithm. They needed branch-and-bound trees in their constraint programming application with more than 10^6 branch-and-bound nodes for both instances, which highlights these results as well.

Combining the results from Table 6.3 with the results from Table 6.2 and Appendix 7.1, the branch-and-price algorithm was able to solve all instances with fewer than 100 vertices. Worth noting is that most *FullIns* instances have only been solved by critical subgraph detection [18]. This is a specialized technique to detect Mycielski subgraphs, which specifically does very well on the *insertion* set of instances, but performs worse on other types of graphs. We are able to solve all the *FullIns* instances within 10 seconds using a more general branch-and-bound method, which is somewhat faster than the results presented by Desrosiers et al. [18], who need more than 10 seconds on 4 *FullIns* instances.

From these results, we can conclude that the ISUN strategy is effective for solving some of the more difficult *Insertion* graphs.

6.3.2 Mod-k cutting planes

Over all instances, 554 maximally violated mod-k cutting planes were found in the root node using configuration C_2. Figure 6.1 shows the distribution for k for all cuts with $k \leq 100$. From all cutting planes, 161 (29.1%) have prime k, with only 13 cutting planes having $k > 100$. and others were found by further iterations of Algorithm 2 or by applying the Chinese Remainder Theorem to two found cuts. This implies most cuts with $k > 100$ are composite and derived from cuts with $k \leq 100$.

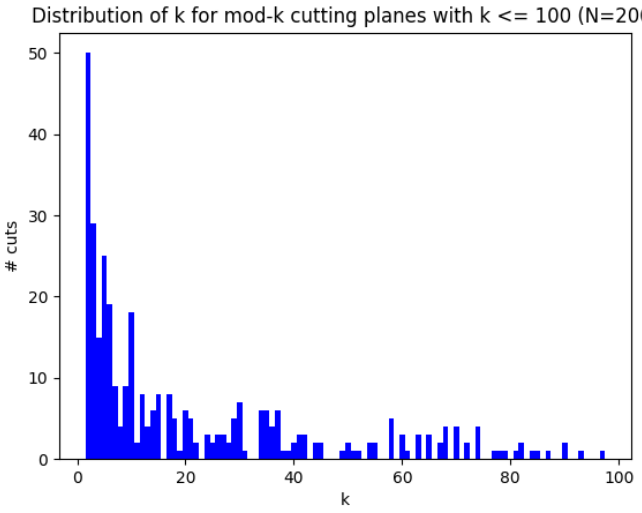


Figure 6.1: Distribution of k where $k \leq 100$ for found maximally violated mod-k cuts in the root node of all DIMACS instances

One thing which stood out from the test results in Table 7.2 is the apparent strength of mod-k cutting planes on *insertion* graphs. Here, often in the root node, the fractional bound is rounded up by adding the mod-k cut. Table 6.4 highlights a few of these results. For the instance *1-FullIns_5* adding mod-k cuts even improves the integer lower bound. Upon investigation of these cuts, we noticed that cuts were often on specific subgraphs of the considered graph. For example, for the instance *2-Insertions_3* a single mod-k cut is found for a subset $V' \in V$ of size $|V'| = 9$, with $\lambda_v = \frac{1}{4}$ if $v \in V'$. This cut effectively requires the condition $\chi(G[V']) \geq 3$ (note that $3 = \lceil \frac{9}{4} \rceil$). Coincidentally, when coloring the graph one finds that $\chi(G[V']) = 3$, so that the linear relaxation of the complete graph G also satisfies the integer bound on the coloring number for this subgraph. Similar cuts are found for many of the other insertion instances, where the mod-k cut finds a subset $V' \subseteq V$ which requires that $\lceil \chi_f(G) \rceil \geq \chi(G[V'])$.

Instance	$\|V\|$	$\|E\|$	$\chi(G)$	time (s)	B&B nodes	LB	$\chi_f(G)$	$\chi_c(G)$	$\|\mathcal{H}\|_{\text{modk}}$	Found UB
1-FullIns_4	93	593	5	0.08	15	5	3.633	4.0	18	5
1-FullIns_5	282	3247	6	3.38	1625	6	3.909	4.066	70	6
1-Insertions_4	67	232	?	tl	79290	4	2.774	3.0	17	5
2-FullIns_4	212	1621	6	0.06	25	6	4.485	5.0	7	6
2-FullIns_5	852	12201	7	6.98	4567	7	4.708	4.8	27	7
2-Insertions_3	37	72	4	0.31	147	4	2.423	3.0	19	4
2-Insertions_4	149	541	?	tl	1513	3.111	2.56	3.0	18	5
3-FullIns_4	405	3524	7	0.06	19	7	5.392	5.5	7	7
3-FullIns_5	2030	33751	8	3.46	2245	8	5.578	6.0	34	8
3-Insertions_3	56	110	4	5.7	505	4	2.334	3.0	4	4
3-Insertions_4	281	1046	?	tl	131	3	2.438	3.0	4	5
4-FullIns_4	690	6650	8	0.1	13	8	6.329	7.0	16	8
4-FullIns_5	4146	77305	9	7.82	3851	9	6.487	6.531	113	9
5-FullIns_4	1085	11395	9	0.13	15	9	7.283	8.0	14	9

Table 6.4: Root bound improvements on the DIMACS instances for configuration C_2

In order to obtain more insight into which mod-k cutting planes contribute to these large increases, we consider cuts which are in the Linear Program at the end of column generation (in the simplex tableau). These cuts can be considered strong in some sense, as they need to be kept in the LP to ensure the larger fractional coloring value. Although cut selection is not

well understood yet [20], a few common heuristics exist. Some of these heuristics, such as the *distance* or the *relative violation* metrics, try to select cuts with large violation but which are somewhat sparse, so that they have a small norm $|a|$ and right hand side $|b|$. In Figure 6.2 we use the norm $|a|$ as a measure of sparsity and compare cuts in and outside of the LP for the instances *DSJC500.9* and *2-Insertions_3*.

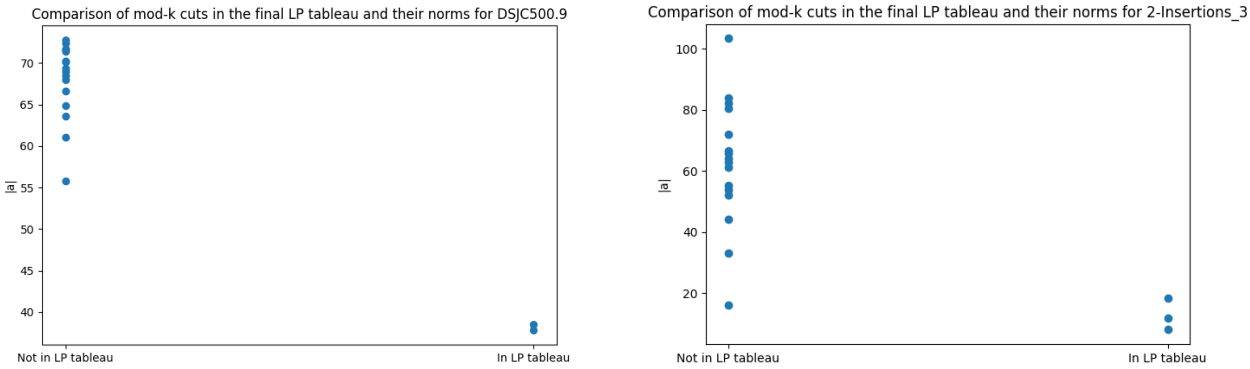


Figure 6.2: Relationship between the norm and presence in the final LP tableau for mod-k cutting planes for the instances *DSJC500.9*,*2-Insertions_3*

From Figure 6.2, we can visually observe a clear correlation between the cuts which are in the LP and their norm $|a|$, observing that cuts with smaller norm seem to be in the LP tableau more often. These results somewhat agree with the metrics from literature, in the sense that sparser cuts with smaller norms $|a|$ are deemed 'stronger'. For other instances, we observed similar results. Note in Figure 6.2 that the large difference in density of both graphs seems to have little effect on the obtained results.

6.3.3 Odd-cycle cutting planes

Using configuration C_3, we observe from Table 7.3 that odd-cycle cutting planes sometimes improve the lower bound at the root node slightly. For example, for the dense instances *DSJC1000.9* and *DSJC250.9* improvements of the order size 0.1 are found. This is in line with the results from Hansen et al. [32], who report a similar increase in bound. For most sparse instances odd-cycle cutting planes seem to be worse in strengthening lower bounds, with an unimproved bound for many instances, even if cutting planes are found. For the instance *DSJC125.1*, odd-cycle cuts are found, but they only increase the bound by approximately $1.3 \cdot 10^{-4}$, for example. This issue is also discussed in [32], where they suggest many symmetric and similar solutions exist and that adding cutting odd-cycle cutting planes does not help much in this regard.

Using both types of cutting planes at the root node seems to be bound-wise, mostly equivalent to using just maximally violated mod-k cutting planes. Only for high density instances such as *DSJC250.9* and *DSJC1000.9*, we see that odd-cycle cutting planes seem to increase the lower bound by more. One observation which can explain this is that the number of odd-cycle cutting planes is often much larger, making it more likely at least one of these cutting planes is 'good'.

For maximally violated mod-k cuts, we considered the cuts which where in the final simplex tableau of the root node. We looked at this statistic for odd-cycle cutting planes as well, but

found that for none of our instances this ever occurred. Instead, we use a similar metric, where we consider the number of LP iterations for which the odd-cycle cut row was in the simplex tableau. One could argue that a cutting plane is stronger if it is in the LP for longer in some sense, or at least suspect a correlation to occur. Figure 6.3 compares the norm $|a|$ of a cutting plane with the number of iterations it was in the LP in the root node for the instance *DSJC125.9*. In the root node of this instance, 146 odd-cycle cuts are separated, which increase the root node solution from $\chi_f(G) \approx 42.742$ to $\chi_c(G) \approx 42.796$.

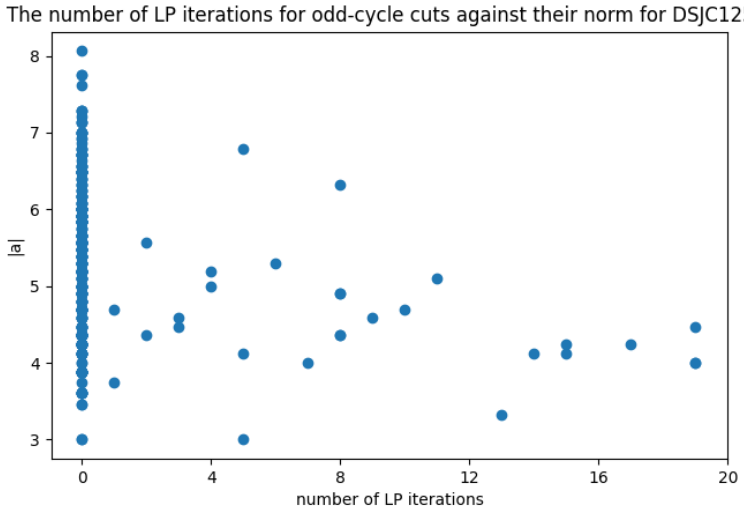


Figure 6.3: Relationship between the norm and number of active LP iterations for odd-cycle cutting planes in the instance *DSJC125.9*

From Figure 6.3 we can observe that many odd-cycle cutting planes never enter the LP. This is not unexpected, as many cuts can be generated at the same time. Secondly, we can observe that for the cuts which enter the LP that there is a slight preference for cuts with smaller norms. For other instances, we observed results similar to those in Figure 6.3. This matches our observations for mod-k cuts, where there seems to be a preference for cuts with smaller norms.

6.3.4 Comparison of branch-cut-and-price with branch-and-price

In this section, we will compare all of the configurations and draw conclusions on the effectiveness of our branch-price-and-cut algorithm. Table 6.5 shows the number of instances solved by branch-and-bound for all configurations C_1–C_5, excluding instances which are solved by C_1 at the root node. The configurations solve approximately the same set of instances, showing that the branch-price-and-cut algorithm is competitive with the existing branch-and-price algorithms. In Table 6.6, we compare the amount of time and the number of branching nodes each configuration needs in order to solve 19 out of the 20 solved instances.

Configuration	C_1	C_2	C_3	C_4	C_5
Solved instances	20	20	20	20	19

Table 6.5: Number of solved instances for each configuration out of the 75 instances which are not solved by C_1 at the root node

	C_1		C_2		C_3		C_4		C_5	
Instance	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
1-FullIns_4	0.07	21	0.08	15	0.06	25	0.07	15	0.07	7
1-FullIns_5	3.44	2025	3.38	1625	3.41	1909	3.63	1713	4.28	1249
2-FullIns_4	0.10	21	0.06	25	0.06	23	0.06	23	0.06	13
2-FullIns_5	5.06	3879	6.98	4567	5.28	4103	5.67	4003	37.31	3799
3-FullIns_4	0.06	19	0.06	19	0.06	19	0.06	19	0.07	13
3-FullIns_5	1.93	1475	3.46	2245	1.90	1437	3.21	2069	7.63	1655
4-FullIns_4	0.13	13	0.10	13	0.08	13	0.08	13	0.09	13
4-FullIns_5	5.73	3561	7.82	3851	6.18	3843	7.69	3737	17.55	2635
5-FullIns_4	0.11	15	0.13	15	0.11	15	0.12	15	0.13	15
2-Insertions_3	0.52	237	0.31	147	0.52	219	0.18	91	0.37	127
3-Insertions_3	9.60	699	5.70	505	13.59	871	5.10	495	4.76	445
4-Insertions_3	437.77	2183	175.5	1365	235.18	1739	91.64	1109	164.93	1129
DSJC125.9	5.57	355	5.90	395	5.73	209	5.95	209	8.84	133
DSJR500.5	880.12	467	395.18	259	290.19	123	431.92	287	*tl*	59
myciel3	0.13	5	0.16	1	0.07	5	0.09	1	0.10	1
myciel4	0.28	77	0.33	77	0.22	77	0.17	77	0.37	63
myciel5	6.27	3619	5.44	3581	4.81	3511	7.73	3555	25.98	3657
queen9_9	4.35	23	4.46	21	4.00	21	3.87	19	4.35	17
queen11_11	666.94	856	6.32	5	7.82	5	103.98	78	38.79	14

Table 6.6: Comparison of time taken for solved instances for configurations C_1-C_5

A few observations can be made from Table 6.6. With the exception of *queen11_11* and *DSJR500.5*, the upper bound found during column initialization for all these instances is equal to $\chi(G)$. For these instances, we thus only need to use a branch-and-bound tree to improve the lower bounds in order to prove optimality of the found upper bound. Considering the time taken, we can observe that adding cutting planes has a mixed impact on the size of the branch-and-bound tree. For some instances, such as *3-Insertions_3*,*4-Insertions_3* and *queen9_9*, a clear decrease in time and branch-and-bound nodes can be observed when comparing strategy C_1 to C_2, C_3 and C_4. When mod-k cuts are used, *myciel3* is even solved in the root node. However, there are also instances where cutting planes seem to have an averse effect on the number of branch-and-bound nodes. For *2-FullIns_4*,*2-FullIns_5* and *3-FullIns_5*, adding cutting planes at the root node increases the size of the branch-and-bound tree.

The instances *queen11_11* and *DSJR500.5* are a bit different from the other instances in the sense that for both $\chi_f(G) = \chi(G)$ holds. As the lower bound at the root node is already strong enough to prove optimality, and only a valid coloring to the Master Problem needs to be found. This is done in SCIP [26] by using many heuristics which round fractional solutions x^* obtained from the LP solutions to integer solutions. It seems that for these instances, adding cutting planes produces fractional solutions x^* which are 'closer' to integer solutions. However, we observed the exact opposite for the instance *queen10_10*, which was only solved by the branch-and-price algorithm (see Table 7.1). With such a small sample size, we cannot conclude that cuts help these heuristics in finding integral solutions, but this remains an interesting observation.

The larger *queen12_12*,*queen13_13* and *queen14_14* instances are also interesting, as they only instances where the branch-and-bound process seemed to be *faster* after adding cutting planes (see appendices). This can be explained by these cutting planes breaking some of the many symmetries of these graphs. This could perhaps help the running time of Algorithm 3 by

being able to cut off certain parts of the combinatorial branch-and-bound tree for the pricing problem.

From Table 6.6 we observe that configuration C_5, which separates cutting planes at each branch-and-bound node, produces the smallest branch-and-bound tree for all but a few instances. This indicates that adding cutting planes indeed increases the lower bound, resulting in a smaller branch-and-bound tree. Do note however, that for many instances the number of branch-and-bound nodes is close to that of configuration C_4. We also see this back in the computation time, especially for instances with more vertices, such as *DSJC125.9*. By separating cutting planes at each branch-and-bound node, the computation time is generally increased.

The specific strength of mod-k cuts and odd-cycle cuts for *insertion* and dense graphs respectively is also reflected in Table 6.6. We observe that for the denser instances *DSJC125.9* and *DSJR500.5* strategy C_3 using odd-cycle cuts is effective in reducing the number of branch-and-bound nodes. Mod-k cuts seem to specifically help for the sparser *myciel3*, *2-Insertions_3*,*3-Insertions_3* and *1-FullIns_5* instances, decreasing the number of necessary branch-and-bound nodes.

Table 6.7 contains an overview of where the extra time is spent when using configuration C_4. All percentages are relative to the total amount of time needed to solve the root node. Then, t_{ORIG} is the fraction of time needed to compute $\chi_f(G)$. The fraction of time spent in the odd-cycle and maximally violated mod-k separation procedures are denoted by t_{OC} and t_{MODK}, respectively. The fraction of time spent in Algorithm 3 after cutting planes where added is denoted by t_P. The remaining fraction of time, which is mostly dominated by solution of the LP for the RMP and heuristics executed by SCIP, is denoted by t_{LP}. The averaged ratio over all instances is also reported, along with a few examples.

Instance	t_{ORIG}	t_{OC}	t_{MODK}	t_P	t_{LP}
all (averaged)	59.8%	1.4%	20.5%	5.8%	12.5%
DSJC125.1	90.8%	0.1%	1.5%	7.2%	0.4%
DSJC125.5	41.9%	0.6%	21.2%	16.4%	19.9%
DSJC125.9	34.4%	1.6%	49.2%	8.4%	6.4%
DSJC500.9	26.8%	0.2%	52.6%	5.8%	14.6%
DSJC1000.9	30.0%	0.1%	34.3%	27.4%	8.2%
myciel5	37.8%	2.0%	5.6%	6.7%	47.9%
myciel6	21.3%	1.5%	25.9%	12.9%	38.4%
myciel7	37.5%	2.1%	18.7%	26.9%	14.8%

Table 6.7: Time division in the root node for a selection of instances

A few things stand out. Firstly, observe that t_{OC} is small so that the cycle separation procedure is relatively cheap compared to solving the LP. Secondly, t_{MODK} usually dominates the time spent after solving the original problem. This is primarily due to our implementation, which allows to check for all primes up to 2^{16} in the selection of k. Particularly for LP solutions with large denominators, the procedure to pick k as described in Section 3.1.3 will fail to find the correct rational representation. This is particularly well demonstrated by the large jump in increased computation time between the instances *myciel5* and *myciel6*, as the denominator of *myciel6* is the first denominator which this algorithm fails to compute. By Figure 6.1, it is also likely that one can reduce this time by picking a smaller limit for separating prime k. Lastly, we observe that t_P and t_{LP} are typically of the same order as t_{ORIG}, so that the extra time spent pricing and in the LP is not many orders of magnitude larger. This implies we can reasonably

expect to solve the root node within 10 times the amount of time it took to solve the original problem for $\chi_f(G)$.

In Table 6.8 we show the achieved lower bounds of all configurations $C_1 - C_5$ for instances that were not solved. Note that finding the upper bound associated to a strong coloring may be difficult. In the appendix Table 7.1, one can observe that although the lower bounds of the *DSJC125.5* and *DSJC125.1* instances are sufficient to prove optimality, a 17-coloring and a 5-coloring respectively are not found within the time limit. Instances such as these are left out of Table 4.2.2.

	C_1			C_2			C_3		
Instance	$\underline{\chi(G)}$	$\chi_f(G)$	nodes	$\underline{\chi(G)}$	$\chi_c(G)$	nodes	$\underline{\chi(G)}$	$\chi_c(G)$	nodes
2-Insertions_4	3.149	2.560	1563	3.111	3.000	1513	3.209*	2.560	1965
3-Insertions_4	3.000	2.438	31	3.000	3.000	131	3.000*	2.438	49
myciel6	5.300	3.834	83532	5.223	3.912	28661	5.400*	3.834	131995
DSJC250.5	25.368	25.165	897	25.364	25.171	874	25.360	25.168	808
DSJC500.9	122.611	122.307	10905	122.609	122.309	10490	122.599	122.334	5886
DSJC1000.9	214.972	214.855	766	214.958	214.858	688	215.019	214.942	60

	C_4			C_5		
Instance	$\underline{\chi(G)}$	$\chi_c(G)$	nodes	$\underline{\chi(G)}$	$\chi_c(G)$	nodes
2-Insertions_4	3.200	3.000	1964	3.172	3.000	1158
3-Insertions_4	3.000	3.000	184	3.000	3.000	140
myciel6	5.217	3.912	24148	4.885	3.912	4203
DSJC250.5	25.360	25.168	812	25.360	25.168	473
DSJC500.9	122.603	122.334	6296	122.548	122.334	727
DSJC1000.9	215.019	214.942	64	214.986	214.942	15

Table 6.8: Comparison of achieved lower bounds for a selection of unsolved instances. For instances marked with *, no cutting planes were found.

From Table 6.8 we can observe that for most instances, the final bound $\underline{\chi(G)}$ only differs by small amounts for each strategy. The only exception to this is strategy C_5, which for *myciel6* is much slower and obtains weaker bounds.
Instance *DSJC1000.9* stands out as interesting. If odd-cycle cuts are used as in configurations $C_3 - C_5$, the found bound is stronger. However, the number of branch-and-bound nodes explored is much smaller, indicating that these odd-cycle cuts slow down the solution speed of each branch-and-bound node. We even see a factor 10 difference, here. Still, the found bounds are stronger. This is a promising result which indicates that odd-cycle cutting planes can be effective in increasing the lower bound for dense random instances. DSJC500.9 shows similar but less extreme results for the branch-and-bound nodes, but the difference in found bound is very small, with C_1 winning out just barely.

Although for both *DSJC250.5* and *DSJC500.9* cutting planes are found which increase the lower bound for strategies C_2 and C_3, it seems that these have little to no effect. We also see for *DSJC1000.9* and *3-Insertions_4* that the C_1 configuration 'catches up' to the other configurations when comparing the obtained benefits from the root node. This pattern was also observed in Table 6.6 for many *FullIns* instances. Here, we see that although strong cuts were found (see Table 6.4), the branch-and-bound tree and time taken are not necessarily positively effected in Table 6.6.

Concluding the comparison between branch-and-price (C_1) and branch-price-and-cut ($C_2 - C_5$), we must conclude that their performance seems to be very similar in practice. Although differences between the two algorithms on certain problem instances exist, they generally perform highly similar on all tested instances. Only strategy C_5, which separates at every branch-and-bound node, performs worse than the branch-and-price algorithm C_1 in practice.

Chapter 7

Discussion and recommendations

Our research has shown that the produced branch-cut-and-price algorithm performs on par with the tested branch-and-price algorithm. However, a few limitations exist that should be discussed.

Firstly, we argue that the branching rule does not cooperate with the cutting planes very well. We observed in section 6.3.4 that the cutting planes at the root node can increase its lower bound, but that further on in the branch-and-bound tree, this advantage often vanishes. Our branching procedure does not take any cutting plane rows which are in the LP into account, and we observed no significant effect of cutting planes on the size of the branch-and-bound tree for a few instances.

Secondly, we do not focus on finding upper bounds in this research at all. Although this is an entirely different challenge, finding strong upper bounds can help the solution speed [44]. This is also highlighted by the many instances which were solved at the root node in Table 7.1, despite their size. Using an algorithm such as MMT [44] could improve the found results.

Regarding cutting planes, we were also somewhat limited by SCIP. SCIP does not easily allow to manually control the addition of cutting planes into the LP. Due to the significant observed increase in time spent in pricing and LP solving (Table 6.7), we would have liked to remove these rows from the LP tableau and only add them selectively for a few branch-and-bound nodes. This could possibly combine the benefits of a faster branch-and-bound search from configuration C_1 with the stronger lower bound cutting planes.

Lastly, we suspect that the performance of the combinatorial pricing algorithm in Algorithm 3 may start to degrade for large and sparse instances. Although these instances are difficult regardless, one could try to tackle the modified pricing problem using a MIP solver instead, and obtain better results.

7.1 Recommendations for further research

One clear recommendation from our research is that further investigation of the branching strategy with Zykov's branching rule is necessary. We have been able to prove many new results using the ISUN strategy (5.5) in section 6.3.1. From the experiments in Chapter 5, many of the newly tested branching rules improve on the known literature, particularly for sparser instances. However, performance of these branching rules seems to differ across the six tested instances, where even a random branching strategy outperforms some of the tested strategies for some instances. This is surprising, and indicates that there may be room for significant improvement

by using better branching strategies. Strong branching strategies could be interesting, considering the results we have obtained.

Results from other papers may also be improved with this new branching rule, in particular for [30, 36] and [44]. Our tests were only ran with a time limit of one hour, which is comparatively quite short, so it is possible that the results some instances could still be improved.

As the results we found do not match the results of Held et al. [36] on the dense instance *DSJC250.9*, we recommend further investigation is done to see where this might come from. The impact of early branching on the strength of the branching edge selection should be investigated here, in particular, but also pricing schemes and differences in LP solver software could be interesting to consider.

Additionally, as our results suggest that using information from the dual variables is also viable for edge selection, this could also be further investigated. Although we have not tested it one could also take the cutting plane dual weights into account when cutting planes are used. The odd-cycle cutting planes give better lower bounds on a (small) subset of vertices within the graph; perhaps branching on a pair of vertices from this subset could yield strong bounds.

Regarding cutting planes, we have a few recommendations. A clear path to follow from this research is to implement the arbitrary rank odd-cycle and mod-k cutting plane procedures from Section 3.3, which we unfortunately lacked the time for. We believe that Algorithm 3 for the pricing problem from Chapter 4 can easily be adjusted as well, by using the weight computation as outlined in section 3.3.3. The only difficulty one may have is obtaining strong upper bounds on the Modified pricing problem, as we have not investigated the structure of the dual problem (the clique-covering problem) in this case.

In this paper, we used only the violation for the weights in the odd-cycle separation algorithm. In our results, we observed that the norm $|a|$ is a decent indicator of the strength of a cutting plane. We suggest that the odd-cycle separation algorithm can be turned into a heuristic by modifying the edge weights. For example, one could add a constant weight to each edge, preferring short cycles, or impose extra penalties for rows with larger norms. This could make the algorithm prefer shorter length cycles, which could lead to cuts with smaller norms, although this is would also mean one loses the guarantee that all weakly violated cycles are found.

Another recommendation would be to use new methodologies to separate more cutting planes. In particular, using stable sets, it is easy to expand or contract stable sets on or from induced subgraphs by greedily adding or removing nodes. This way, one could perhaps separate on solutions for induced subgraphs $G[V']$, as this is equivalent to simply searching cuts with the multiplier $\lambda_v = 0$ if $v \notin V'$. This may lead to finding sparser and/or stronger cuts. For larger graphs, where solving the root node can be extremely challenging, one could also start by first optimizing over subgraphs and finding cuts in this manner before the root node is solved. Separation on solutions which are not yet optimal would also be a possibility and may yield better cuts [49]. Both of these ideas are also supported and motivated by the observed strong mod-k cutting planes for insertion graphs in Section 6.3.2, which highlight that the lower bound of the root node may increase based on chromatic numbers of easier subgraphs.

Another way to find more cutting planes would be to use more heuristic methods to find cutting planes. Particularly, heuristics could aim to find mod-k planes which are not necessarily maximally violated. As the mod-k separation problem can be formulated as an integer program, many heuristic approaches could be used to tackle it.

We observed no real issues with the pricing problem, but one may want to add stabilization procedures similar to the one as described by Held et al. [36]. This could potentially reduce the solution time of each branch-and-bound node. Lastly, we recommend to investigate parallelization of the pricing problem. In particular, running several types of algorithms for the (modified) pricing problem in parallel could be fruitful. One could let these algorithms communicate found solutions and bounds with each other, which may reduce the time needed to solve the pricing problem, in particular for the more difficult DIMACS instances.

Bibliography

[1] Diogo V. Andrade, Mauricio G.C. Resende, and Renato F. Werneck. "Fast local search for the maximum independent set problem". In: *Journal of Heuristics* 18.4 (2012), pp. 525–547.

[2] E. Balas and Jue Xue. "Weighted and Unweighted Maximum Clique Algorithms with Upper Bounds from Fractional Coloring". In: *Algorithmica* 15.5 (1996), p. 397.

[3] Egon Balas. "Cutting planes from conditional bounds: A new approach to set covering". In: 1980, pp. 19–36.

[4] Egon Balas and Andrew Ho. "Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study". In: February (1980), pp. 37–60.

[5] Una Benlic and Jin Kao Hao. "Breakout Local Search for maximum clique problems". In: *Computers and Operations Research* 40.1 (2013), pp. 192–206.

[6] Ivo Blöchliger and Nicolas Zufferey. "A graph coloring heuristic using partial solutions and a reactive tabu scheme". In: *Computers and Operations Research* 35.3 (2008), pp. 960–975.

[7] Noureddine Bouhmala and Ole-Christoffer Granmo. "Solving Graph Coloring Problems Using Learning Automata". In: 2008, pp. 277–288.

[8] Daniel Brélaz. "New methods to color the vertices of a graph". In: *Communications of the ACM* 22.4 (Apr. 1979), pp. 251–256.

[9] Edmund K. Burke et al. "A graph-based hyper-heuristic for educational timetabling problems". In: *European Journal of Operational Research* 176.1 (Jan. 2007), pp. 177–192.

[10] Manoel Campêlo, Victor A. Campos, and Ricardo C. Corrêa. "On the asymmetric representatives formulation for the vertex coloring problem". In: *Discrete Applied Mathematics* 156.7 (Apr. 2008), pp. 1097–1111.

[11] Alberto Caprara and Matteo Fischetti. "0, 1/2-Chvátal-Gomory cuts". In: *Mathematical Programming, Series B* 74.3 (Sept. 1996), pp. 221–235.

[12] Alberto Caprara, Matteo Fischetti, and Adam N. Letchford. "On the separation of maximally violated mod-k cuts". In: *Mathematical Programming* 87.1 (Jan. 2000), pp. 37–56.

[13] V. Chvátal. "Edmonds polytopes and a hierarchy of combinatorial problems". In: *Discrete Mathematics* 4.4 (Apr. 1973), pp. 305–337.

[14] V. Chvátal. *Colouring the queen graphs*. URL: `http://users.encs.concordia.ca/$%5Csim$chvatal/queengraphs.html` (visited on 08/13/2021).

[15] Denis Cornaz, Fabio Furini, and Enrico Malaguti. "Solving vertex coloring problems as maximum weight stable set problems". In: *Discrete Applied Mathematics* 217 (2017), pp. 151–162.

[16] D. De Werra et al. "On a graph-theoretical model for cyclic register allocation". In: *Discrete Applied Mathematics* 93.2-3 (1999), pp. 191–203.

[17] Thomas Dence and Joseph Dence. *Elements of the theory of numbers*. 1999, pp. 156–164.

[18] C. Desrosiers, P. Galinier, and A. Hertz. "Efficient algorithms for finding critical subgraphs". In: *Discrete Applied Mathematics* 156.2 (2008), pp. 244–266.

[19] Jacques Desrosiers and Marco E. Lübbecke. "Branch-Price-and-Cut Algorithms". In: *Wiley Encyclopedia of Operations Research and Management Science*. January. Hoboken, NJ, USA: John Wiley and Sons, Inc., Jan. 2011.

[20] Santanu S. Dey and Marco Molinaro. "Theoretical challenges towards cutting-plane selection". In: *Mathematical Programming* 170.1 (July 2018), pp. 237–266. arXiv: 1805.02782.

[21] Raphaël Dorne and Jin-Kao Hao. "Tabu Search for Graph Coloring, T-Colorings and Set T-Colorings". In: *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Vol. 6. Boston, MA: Springer US, 1999, pp. 77–92.

[22] Jean-Guillaume Dumas, Pascal Giorgi, and Clément Pernet. "Dense Linear Algebra over Word-Size Prime Fields: the FFLAS and FFPACK Packages". In: *ACM Trans. on Mathematical Software (TOMS)* 35.3 (2008), pp. 1–42.

[23] OEIS Foundation. *The On-Line Encyclopedia of Integer Sequences*. 2021. URL: https://oeis.org/A073834 (visited on 08/12/2021).

[24] Philippe Galinier and Alain Hertz. "A survey of local search methods for graph coloring". In: *Computers and Operations Research* 33.9 (Sept. 2006), pp. 2547–2562.

[25] Michel Gamache, Alain Hertz, and Jérôme Olivier Ouellet. "A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding". In: *Computers and Operations Research* 34.8 (2007), pp. 2384–2395.

[26] Gerald Gamrath et al. *The SCIP Optimization Suite 7.0*. Tech. rep. 05. 2020, pp. 1–46.

[27] A. M. H. Gerards and A. Schrijver. "Matrices with the edmonds—Johnson property". In: *Combinatorica* 6.4 (Dec. 1986), pp. 365–379.

[28] Ralph E. Gomory. "Outline of an Algorithm for Integer Solutions to Linear Programs and An Algorithm for the Mixed Integer Problem". In: *50 Years of Integer Programming 1958-2008*. January 2010. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 77–103.

[29] The LinBox group. *LinBox*. v1.6.3. 2019. URL: http://github.com/linbox-team/linbox.

[30] Stefano Gualandi and Federico Malucelli. "Exact solution of graph coloring problems via constraint programming and column generation". In: *INFORMS Journal on Computing* 24.1 (2012), pp. 81–100.

[31] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2021. URL: https://www.gurobi.com.

[32] P. Hansen, M. Labbé, and D. Schindl. "Set covering and packing formulations of graph coloring: Algorithms and first polyhedral results". In: *Discrete Optimization* 6.2 (May 2009), pp. 135–147.

[33] Jin Kao Hao and Qinghua Wu. "Improving the extraction and expansion method for large graph coloring". In: *Discrete Applied Mathematics* 160.16-17 (2012), pp. 2397–2407.

[34] Johan Håstad. "Clique is hard to approximate within n1ϵ". In: *Acta Mathematica* 182.1 (1999), pp. 105–142.

[35] Emmanuel Hebrard and George Katsirelos. "Clause Learning and New Bounds for Graph Coloring". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 11008 LNCS. 2018, pp. 179–194.

[36] Stephan Held, William Cook, and Edward C. Sewell. "Safe lower bounds for graph coloring". In: *International Conference on Integer Programming and Combinatorial Optimization*. Vol. 6655 LNCS. Berlin, Heidelberg: Springer, 2011, pp. 261–273.

[37] A. Hertz and D. de Werra. "Using tabu search techniques for graph coloring". In: *Computing* 39.4 (Dec. 1987), pp. 345–351.

[38] Alain Hertz, Matthieu Plumettaz, and Nicolas Zufferey. "Variable space search for graph coloring". In: *Discrete Applied Mathematics* 156.13 (July 2008), pp. 2551–2560.

[39] Richard M. Karp. "Reducibility Among Combinatorial Problems". In: *50 Years of Integer Programming 1958-2008*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 219–241.

[40] Michael Larsen, James Propp, and Daniel Ullman. "The fractional chromatic number of mycielski's graphs". In: *Journal of Graph Theory* 19.3 (May 1995), pp. 411–416.

[41] A.K. Lenstra and H.W. Lenstra Jr. "Algorithms in Number Theory". In: *Handbook of Theoretical Computer Science*. 1990, pp. 675–715.

[42] Chu-Min Li et al. "A new upper bound for the maximum weight clique problem". In: *European Journal of Operational Research* 270.1 (Oct. 2018), pp. 66–77.

[43] Marco E. Lübbecke and Jacques Desrosiers. "Selected Topics in Column Generation". In: *Operations Research* 53.6 (Dec. 2005), pp. 1007–1023.

[44] Enrico Malaguti, Michele Monaci, and Paolo Toth. "An exact approach for the Vertex Coloring Problem". In: *Discrete Optimization* 8.2 (2011), pp. 174–190.

[45] Anuj Mehrotra and Michael A. Trick. "A Column Generation Approach for Graph Coloring". In: *INFORMS Journal on Computing* 8.4 (Nov. 1996), pp. 344–354.

[46] Isabel Méndez-Díaz and Paula Zabala. "A Branch-and-Cut algorithm for graph coloring". In: *Discrete Applied Mathematics* 154.5 SPEC. ISS. (2006), pp. 826–847.

[47] Isabel Méndez-Díaz and Paula Zabala. "A cutting plane algorithm for graph coloring". In: *Discrete Applied Mathematics* 156.2 (Jan. 2008), pp. 159–179.

[48] J. W. Moon and L. Moser. "On cliques in graphs". In: *Israel Journal of Mathematics* 3.1 (Mar. 1965), pp. 23–28.

[49] Pedro Munari and Jacek Gondzio. "Using the primal-dual interior point algorithm within the branch-price-and-cut method". In: *Computers and Operations Research* 40.8 (2013), pp. 2026–2036.

[50] Sk Md Abu Nayeem and Madhumangal Pal. "Genetic algorithmic approach to find the maximum weight independent set of a graph". In: *Journal of Applied Mathematics and Computing* 25.1-2 (Sept. 2007), pp. 217–229.

[51] P. Nobili and A. Sassano. "Facets and Lifting Procedures for the Set Covering Polytope". In: *Math Prog.* 45 (1989), pp. 111–137.

[52] P. Nobili and A. Sassano. "A Separation Routine for the Set Covering Polytope". In: *Integer Programming and Combinatorial Optimiziation, Proc. of the 2nd Int IPCO Conf.* 1992, pp. 201–219.

[53] Patric R.J. Östergård. "A New Algorithm for the Maximum-Weight Clique Problem". In: *Electronic Notes in Discrete Mathematics* 3.April 1999 (May 1999), pp. 153–156.

[54] Daniel Cosmin Porumbel, Jin-Kao Hao, and Pascale Kuntz. "Position-Guided Tabu Search Algorithm for the Graph Coloring Problem". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 5851 LNCS. May 2014. 2009, pp. 148–162.

[55] Wayne Pullan. “Approximating the maximum vertex/edge weighted clique using local search”. In: *Journal of Heuristics* 14.2 (Apr. 2008), pp. 117–134.

[56] Wayne Pullan. “Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers”. In: *Discrete Optimization* 6.2 (2009), pp. 214–219.

[57] Steffen Rebennack et al. “A Branch and Cut solver for the maximum stable set problem”. In: *Journal of Combinatorial Optimization* 21.4 (May 2011), pp. 434–457.

[58] David Ryan and Brian Foster. “An Integer Programming Approach to Scheduling”. In: *Computer Scheduling of Public Transport* 1 (Jan. 1981), pp. 269–.

[59] David Schindl. “Some combinatorial optimization problems in graphs with applications in telecommunications and tomography”. PhD thesis. Ecole Polytechnique Federale de Lausanne, 2004.

[60] A. Schrijver. *Theory of Linear and Integer Programming*. 1998, pp. 347–351.

[61] Carla Silva et al. “Mapping graph coloring to quantum annealing”. In: *Quantum Machine Intelligence* 2.2 (Dec. 2020), p. 16.

[62] D. H. Smith, S. Hurley, and S. U. Thiel. “Improving heuristics for the frequency assignment problem”. In: *European Journal of Operational Research* 107.1 (1998), pp. 76–86.

[63] Wen Sun. “Heuristic Algorithms for Graph Coloring Problems”. PhD thesis. Université d’Angers, 2018. URL: `https://tel.archives-ouvertes.fr/tel-02136810/document`.

[64] Olawale Titiloye and Alan Crispin. “Quantum annealing of the graph coloring problem”. In: *Discrete Optimization* 8.2 (May 2011), pp. 376–384.

[65] Allen Van Gelder. “Another look at graph coloring via propositional satisfiability”. In: *Discrete Applied Mathematics* 156.2 (Jan. 2008), pp. 230–243.

[66] Michel Vasquez. “New results on the queens n^2 graph coloring problem”. In: *Journal of Heuristics* 10.4 (July 2004), pp. 407–413.

[67] JS Warren and IV Hicks. *Combinatorial branch-and-bound for the maximum weight independent set problem*. Tech. rep. 2006, pp. 1–23.

[68] Deepak Warrier. “A branch, price, and cut approach to solving the maximum weighted independent set problem”. In: 2007.

[69] D. de Werra. “An introduction to timetabling”. In: *European Journal of Operational Research* 19.2 (1985), pp. 151–162.

[70] D. Wiedemann. “Solving sparse linear equations over finite fields”. In: *IEEE Transactions on Information Theory* 32.1 (Jan. 1986), pp. 54–62.

[71] T.-K. Woo, S.Y.W. Su, and R. Newman-Wolfe. “Resource allocation in a dynamically partitionable bus network using a graph coloring algorithm”. In: *IEEE Transactions on Communications* 39.12 (1991), pp. 1794–1801.

[72] Qinghua Wu, Jin Kao Hao, and Fred Glover. “Multi-neighborhood tabu search for the maximum weight clique problem”. In: *Annals of Operations Research* 196.1 (2012), pp. 611–634.

[73] Zhaoyang Zhou et al. “An exact algorithm with learning for the graph coloring problem”. In: *Computers and Operations Research* 51.January 2020 (Nov. 2014), pp. 282–301.

[74] Nicolas Zufferey, Patrick Amstutz, and Philippe Giaccari. “Graph colouring approaches for a satellite range scheduling problem”. In: *Journal of Scheduling* 11.4 (2008), pp. 263–277.

[75] A.A. Zykov. “On some properties of linear complexes”. In: *Matematicheskij Sbornik* 24 (1949), pp. 163–188.

Appendix

This appendix contains the results of the experiments on the DIMACS instances with the configurations C_1-C_5 from Table 6.1 as described in Chapter 6. The 'Best UB' column in Table 7.1 contains the best known upper bounds from literature. The chromatic number $\chi(G)$ in Table 7.1 is highlighted in bold for instances which were proven optimal by a branch-and-price algorithm for the first time.

Table 7.1: Results on the DIMACS instances for strategy C_1

Instance	$\|V\|$	$\|E\|$	$\chi(G)$	time (s)	B&B nodes	$\underline{\chi}(G)$	Best UB	Found UB
1-FullIns_3	30	100	4	0.07	1	4	4	4
1-FullIns_4	93	593	**5**	0.07	21	5	5	5
1-FullIns_5	282	3247	**6**	3.44	2025	6	6	6
1-Insertions_4	67	232	5 [46]	tl	78729	4	5 [46]	5
1-Insertions_5	202	1227	?	tl	504	3.342	6 [46]	6
1-Insertions_6	607	6337	?	tl	1	2.535	7 [46]	7
2-FullIns_3	52	201	5	0.03	1	5	5	5
2-FullIns_4	212	1621	**6**	0.1	21	6	6	6
2-FullIns_5	852	12201	**7**	5.06	3879	7	7	7
2-Insertions_3	37	72	4	0.52	237	4	4	4
2-Insertions_4	149	541	?	tl	1563	3.149	5 [46]	5
2-Insertions_5	597	3936	?	tl	1	2.198	6 [46]	6
3-FullIns_3	80	346	6	0.06	1	6	6	6
3-FullIns_4	405	3524	**7**	0.06	19	7	7	7
3-FullIns_5	2030	33751	**8**	1.93	1475	8	8	8
3-Insertions_3	56	110	**4**	9.6	699	4	4	4
3-Insertions_4	281	1046	?	tl	31	3	5 [46]	5
3-Insertions_5	1406	9695	?	tl	1	2.023	6 [46]	6
4-FullIns_3	114	541	7	0.09	1	7	7	7
4-FullIns_4	690	6650	**8**	0.13	13	8	8	8
4-FullIns_5	4146	77305	**9**	5.73	3561	9	9	9
4-Insertions_3	79	156	**4**	437.77	2183	4	4	4
4-Insertions_4	475	1795	?	tl	1	2.039	5 [46]	5
5-FullIns_3	154	792	8	0.05	1	8	8	8
5-FullIns_4	1085	11395	**9**	0.11	15	9	9	9
C1000.9	1000	450079	?	tl	752	215.479	?	238
C2000.5	2000	999836	?	tl	1	21.187	145 [33]	175
C2000.9	2000	1799532	?	tl	2	389.6	400 [64]	446
C4000.5	4000	4000268	?	tl	1	15	259 [33]	326
DSJC1000.1	1000	49629	?	tl	1	6	20 [44]	23
DSJC1000.5	1000	249826	?	tl	1	18.854	83 [64]	97
DSJC1000.9	1000	449449	?	tl	766	214.972	222 [64]	238
DSJC125.1	125	736	5	tl	823	4.751	5 [44]	6
DSJC125.5	125	3891	17	tl	51541	16.179	17 [44]	18
DSJC125.9	125	6961	44	5.57	355	44	44	44
DSJC250.1	250	3218	?	tl	1	4	8 [44]	9
DSJC250.5	250	15668	?	tl	897	25.368	28 [44]	31

Table 7.1: Results on the DIMACS instances for strategy C_1

Instance	$\|V\|$	$\|E\|$	$\chi(G)$	time (s)	B&B nodes	$\underline{\chi(G)}$	Best UB	Found UB
DSJC250.9	250	27897	?	tl	122299	70.979	72 [44]	73
DSJC500.1	500	12458	?	tl	1	5	12 [44]	14
DSJC500.5	500	62624	?	tl	4	42.267	48 [64]	54
DSJC500.9	500	112437	?	tl	10905	122.611	126 [64]	132
DSJR500.1	500	3555	12	0.04	1	12	12	12
DSJR500.1c	500	121275	85	6.77	10	85	85	85
DSJR500.5	500	58862	122	880.12	467	122	122	122
abb313GPIA	1557	65390	?	tl	1	8	9 [44]	9
anna	138	986	11	0.08	1	11	11	11
ash331GPIA	662	4185	4	5.95	1	4	4	4
ash608GPIA	1216	7844	4	tl	1	3	4 [44]	4
ash958GPIA	1916	12506	4	tl	1	3	4 [44]	4
david	87	812	11	0.02	1	11	11 [45]	11
flat1000_50_0	1000	245000	50	tl	1	18.436	50 [46]	50
flat1000_60_0	1000	245830	60	tl	1	18.207	60 [46]	60
flat1000_76_0	1000	246708	76	tl	1	18.748	81 [33]	96
flat300_20_0	300	21375	20	71.19	1	20	20	20
flat300_26_0	300	21633	26	224.75	1	26	26	26
flat300_28_0	300	21695	28	tl	150	27.612	28 [6]	34
fpsol2.i.1	496	11654	65	0.07	1	65	65	65
fpsol2.i.2	451	8691	30	0.13	1	30	30	30
fpsol2.i.3	425	8688	30	0.14	1	30	30	30
games120	120	1276	9	0.02	1	9	9	9
homer	561	3258	13	0.05	1	13	13	13
huck	74	602	11	0.04	1	11	11	11
inithx.i.1	864	18707	54	0.1	1	54	54	54
inithx.i.2	645	13979	31	0.1	1	31	31	31
inithx.i.3	621	13969	31	0.11	1	31	31	31
jean	80	508	10	0.05	1	10	10	10
latin_square_10	900	307350	?	tl	225	90	98 [64]	109
le450_15a	450	8168	15	tl	10	15	15 [44]	16
le450_15b	450	8169	15	tl	28	15	15 [44]	16
le450_15c	450	16680	15	tl	1	15	15 [44]	16
le450_15d	450	16750	15	tl	1	15	15 [44]	16
le450_25a	450	8260	25	0.26	1	25	25	25
le450_25b	450	8263	25	0.17	1	25	25	25
le450_25c	450	17343	25	tl	13	25	25 [44]	28
le450_25d	450	17425	25	tl	15	25	25[44]	28
le450_5a	450	5714	5	tl	1	5	5 [44]	6
le450_5b	450	5734	5	tl	1	5	5 [44]	6
le450_5c	450	9803	5	1604.08	1	5	5	5
le450_5d	450	9757	5	tl	1	5	5 [44]	6
miles1000	128	6432	42	0.05	1	42	42	42
miles1500	128	10396	73	0.06	1	73	73	73
miles250	128	774	8	0.03	1	8	8	8
miles500	128	2340	20	0.03	1	20	20	20
miles750	128	4226	31	0.04	1	31	31	31
mug100_1	100	166	4	1.53	1	4	4	4
mug100_25	100	166	4	1.45	1	4	4	4
mug88_1	88	146	4	0.79	1	4	4	4
mug88_25	88	146	4	0.99	1	4	4	4
mulsol.i.1	197	3925	49	0.05	1	49	49	49
mulsol.i.2	188	3885	31	0.07	1	31	31	31
mulsol.i.3	184	3916	31	0.04	1	31	31	31
mulsol.i.4	185	3946	31	0.05	1	31	31	31

Table 7.1: Results on the DIMACS instances for strategy C_1

Instance	$\|V\|$	$\|E\|$	$\chi(G)$	time (s)	B&B nodes	$\underline{\chi(G)}$	Best UB	Found UB
mulsol.i.5	186	3973	31	0.05	1	31	31	31
myciel3	11	20	4	0.13	5	4	4	4
myciel4	23	71	5	0.28	77	5	5	5
myciel5	47	236	6	6.27	3619	6	6	6
myciel6	95	755	7	tl	83532	5.3	7	7
myciel7	191	2360	8	tl	7762	4.898	8	8
qg.order100	10000	990000	100	512.93	1	100	100	100
qg.order30	900	26100	30	1.15	1	30	30	30
qg.order40	1600	62400	40	tl	1	40	40 [44]	41
qg.order60	3600	212400	60	13.86	1	60	60	60
queen10_10	100	2940	11	3113.88	20057	11	11	11
queen11_11	121	3960	11	666.94	856	11	11	11
queen12_12	144	5192	12	tl	1133	12	12 [66]	14
queen13_13	169	6656	13	tl	127	13	13 [66]	15
queen14_14	196	8372	14	tl	17	14	14 [66]	16
queen15_15	225	10360	15	tl	5	15	16 [44]	17
queen16_16	256	12640	16	tl	2	16	17 [44]	18
queen5_5	25	320	5	0.06	1	5	5	5
queen6_6	36	580	7	0.09	1	7	7	7
queen7_7	49	952	7	0.14	1	7	7	7
queen8_12	96	2736	12	0.11	1	12	12	12
queen8_8	64	1456	9	0.25	1	9	9	9
queen9_9	81	2112	10	4.35	23	10	10	10
school1	385	19095	14	1.07	1	14	14	14
school1_nsh	352	14612	14	0.95	1	14	14	14
wap01a	2368	110871	?	tl	1	41	43 [44]	47
wap02a	2464	111742	?	tl	1	40	42 [44]	44
wap03a	4730	286722	?	tl	1	40	47 [44]	48
wap04a	5231	294902	?	tl	1	40	42 [33]	46
wap05a	905	43081	50	0.35	1	50	50	50
wap06a	947	43571	40	tl	35	40	40 [44]	42
wap07a	1809	103368	?	tl	1	40	41 [33]	45
wap08a	1870	104176	?	tl	1	40	42 [44]	45
will199GPIA	701	7065	7	5.4	1	7	7	7
zeroin.i.1	211	4100	49	0.05	1	49	49	49
zeroin.i.2	211	3541	30	0.04	1	30	30	30
zeroin.i.3	206	3540	30	0.06	1	30	30	30

Table 7.2: Results on the DIMACS instances for strategy C_2

Instance	$\|V\|$	$\|E\|$	$\chi(G)$	time (s)	B&B nodes	$\underline{\chi(G)}$	$\chi_f(G)$	$\chi_c(G)$	$\|\mathcal{H}\|_{\text{modk}}$	Found UB
1-FullIns_4	93	593	5	0.08	15	5	3.633	4.0	18	5
1-FullIns_5	282	3247	6	3.38	1625	6	3.909	4.066	70	6
1-Insertions_4	67	232	5	tl	79290	4	2.774	3.0	17	5
1-Insertions_5	202	1227	?	tl	1	2.943	2.943	-	7	6
1-Insertions_6	607	6337	?	tl	1	2.497	-	-	0	7
2-FullIns_4	212	1621	6	0.06	25	6	4.485	5.0	7	6
2-FullIns_5	852	12201	7	6.98	4567	7	4.708	4.8	27	7
2-Insertions_3	37	72	4	0.31	147	4	2.423	3.0	19	4
2-Insertions_4	149	541	?	tl	1513	3.111	2.56	3.0	18	5
2-Insertions_5	597	3936	?	tl	1	2.198	-	-	0	6
3-FullIns_4	405	3524	7	0.06	19	7	5.392	5.5	7	7
3-FullIns_5	2030	33751	8	3.46	2245	8	5.578	6.0	34	8
3-Insertions_3	56	110	4	5.7	505	4	2.334	3.0	4	4
3-Insertions_4	281	1046	?	tl	131	3	2.438	3.0	4	5

Table 7.2: Results on the DIMACS instances for strategy C_2

Instance	$\|V\|$	$\|E\|$	$\chi(G)$	time (s)	B&B nodes	$\underline{\chi(G)}$	$\chi_f(G)$	$\chi_c(G)$	$\|\mathcal{H}\|_{\text{modk}}$	Found UB
3-Insertions_5	1406	9695	?	tl	1	2.026	-	-	0	6
4-FullIns_4	690	6650	8	0.1	13	8	6.329	7.0	16	8
4-FullIns_5	4146	77305	9	7.82	3851	9	6.487	6.531	113	9
4-Insertions_3	79	156	4	175.5	1365	4	2.276	3.0	4	4
4-Insertions_4	475	1795	?	tl	1	2.039	-	-	0	5
5-FullIns_4	1085	11395	9	0.13	15	9	7.283	8.0	14	9
C1000.9	1000	450079	?	tl	637	215.481	215.354	215.357	2	240
C2000.5	2000	999836	?	tl	1	21.86	-	-	0	175
C2000.9	2000	1799532	?	tl	1	389.6	389.6	389.6	31	446
C4000.5	4000	4000268	?	tl	1	25.077	-	-	0	326
DSJC1000.1	1000	49629	?	tl	1	6	-	-	0	23
DSJC1000.5	1000	249826	?	tl	1	19.152	-	-	0	97
DSJC1000.9	1000	449449	?	tl	688	214.958	214.855	214.858	26	238
DSJC125.1	125	736	5	tl	813	4.731	4.454	4.456	1	6
DSJC125.5	125	3891	17	tl	50887	16.149	15.727	15.737	8	18
DSJC125.9	125	6961	44	5.9	395	44	42.742	42.729	3	44
DSJC250.1	250	3218	?	tl	1	4	-	-	0	9
DSJC250.5	250	15668	?	tl	874	25.364	25.165	25.171	2	31
DSJC250.9	250	27897	?	tl	122784	70.991	70.392	70.398	7	73
DSJC500.1	500	12458	?	tl	1	5	-	-	0	14
DSJC500.5	500	62624	?	tl	4	42.267	42.264	42.264	0	54
DSJC500.9	500	112437	?	tl	10490	122.609	122.307	122.309	18	132
DSJR500.5	500	58862	122	395.18	259	122	122.0	122.0	2	122
abb313GPIA	1557	65390	4	tl	1	8	-	-	0	9
ash608GPIA	1216	7844	4	tl	1	3	-	-	0	4
ash958GPIA	1916	12506	4	tl	1	3	-	-	0	4
flat1000_50_0	1000	245000	50	489.71	1	50	50.0	50.0	0	50
flat1000_60_0	1000	245830	60	tl	1	19.055	-	-	0	60
flat1000_76_0	1000	246708	76	tl	1	18.804	-	-	0	96
flat300_28_0	300	21695	28	tl	152	27.609	27.52	27.52	0	34
latin_square_10	900	307350	?	tl	207	90	90.0	90.0	3	109
le450_15a	450	8168	15	tl	14	15	15.0	15.0	2	16
le450_15b	450	8169	15	tl	2	15	15.0	15.0	3	16
le450_15c	450	16680	15	tl	1	15	-	-	0	16
le450_15d	450	16750	15	tl	1	15	-	-	0	16
le450_25c	450	17343	25	tl	4	25	25.0	25.0	8	28
le450_25d	450	17425	25	tl	10	25	25.0	25.0	0	28
le450_5a	450	5714	5	tl	1	5	-	-	0	6
le450_5b	450	5734	5	tl	1	5	-	-	0	6
le450_5d	450	9757	5	tl	1	5	-	-	0	6
myciel3	11	20	4	0.16	1	4	2.9	3.028	8	4
myciel4	23	71	5	0.33	77	5	3.245	3.362	10	5
myciel5	47	236	6	5.44	3581	6	3.553	3.622	8	6
myciel6	95	755	7	tl	28661	5.223	3.834	3.912	10	7
myciel7	191	2360	8	tl	7745	4.898	4.095	4.147	16	8
qg.order40	1600	62400	40	tl	1	40	40.0	40.0	7	41
queen10_10	100	2940	11	tl	25888	10.161	10.0	10.0	0	12
queen11_11	121	3960	11	6.32	5	11	11.0	11.0	11	11
queen12_12	144	5192	12	tl	954	12	12.0	12.0	1	14
queen13_13	169	6656	13	tl	158	13	13.0	13.0	0	15
queen14_14	196	8372	14	tl	25	14	14.0	14.0	4	16
queen15_15	225	10360	15	tl	17	15	15.0	15.0	0	17
queen16_16	256	12640	16	tl	6	16	16.0	16.0	5	18
queen9_9	81	2112	10	4.46	21	10	9.0	9.0	23	10
wap01a	2368	110871	?	tl	1	41	-	-	0	47

Table 7.2: Results on the DIMACS instances for strategy C_2

Instance	$\|V\|$	$\|E\|$	$\chi(G)$	time (s)	B&B nodes	$\underline{\chi(G)}$	$\chi_f(G)$	$\chi_c(G)$	$\|\mathcal{H}\|_{\text{modk}}$	Found UB
wap02a	2464	111742	?	tl	1	40	-	-	0	44
wap03a	4730	286722	?	tl	1	40	-	-	0	48
wap04a	5231	294902	?	tl	1	40	-	-	0	46
wap06a	947	43571	40	tl	38	40	40.0	40.0	2	42
wap07a	1809	103368	?	tl	1	40	-	-	0	45
wap08a	1870	104176	?	tl	1	40	-	-	0	45

Table 7.3: Results on the DIMACS instances for strategy C_3

Instance	$\|V\|$	$\|E\|$	$\chi(G)$	time (s)	B&B nodes	$\underline{\chi(G)}$	$\chi_f(G)$	$\chi_c(G)$	$\|\mathcal{H}\|_{\text{oc}}$	Found UB
1-FullIns_4	93	593	5	0.06	25	5	3.633	3.633	7	5
1-FullIns_5	282	3247	6	3.41	1909	6	3.909	3.909	8	6
1-Insertions_4	67	232	5	tl	83439	4	2.774	2.774	0	5
1-Insertions_5	202	1227	?	tl	471	3.333	2.943	2.943	0	6
1-Insertions_6	607	6337	?	tl	1	2.497	-	-	0	7
2-FullIns_4	212	1621	6	0.06	23	6	4.485	4.485	3	6
2-FullIns_5	852	12201	7	5.28	4103	7	4.708	4.708	13	7
2-Insertions_3	37	72	4	0.52	219	4	2.423	2.423	0	4
2-Insertions_4	149	541	?	tl	1965	3.209	2.56	2.56	0	5
2-Insertions_5	597	3936	?	tl	1	2.198	-	-	0	6
3-FullIns_4	405	3524	7	0.06	19	7	5.392	5.392	15	7
3-FullIns_5	2030	33751	8	1.9	1437	8	5.578	5.578	0	8
3-Insertions_3	56	110	4	13.59	871	4	2.334	2.334	0	4
3-Insertions_4	281	1046	?	tl	49	3	2.438	2.438	0	5
3-Insertions_5	1406	9695	?	tl	1	2.026	-	-	0	6
4-FullIns_4	690	6650	8	0.08	13	8	6.329	6.329	17	8
4-FullIns_5	4146	77305	9	6.18	3843	9	6.487	6.487	32	9
4-Insertions_3	79	156	4	235.18	1739	4	2.276	2.276	0	4
4-Insertions_4	475	1795	?	tl	1	2.039	-	-	0	5
5-FullIns_4	1085	11395	9	0.11	15	9	7.283	7.283	24	9
C1000.9	1000	450079	?	tl	61	215.483	215.354	215.421	538	241
C2000.5	2000	999836	?	tl	1	21.86	-	-	0	175
C2000.9	2000	1799532	?	tl	4	389.6	389.6	389.6	412	446
C4000.5	4000	4000268	?	tl	1	25.077	-	-	0	326
DSJC1000.1	1000	49629	?	tl	1	6	-	-	0	23
DSJC1000.5	1000	249826	?	tl	1	19.152	-	-	0	97
DSJC1000.9	1000	449449	?	tl	60	215.019	214.855	214.942	706	241
DSJC125.1	125	736	5	tl	802	4.744	4.454	4.454	16	6
DSJC125.5	125	3891	17	tl	43333	16.188	15.727	15.744	65	18
DSJC125.9	125	6961	44	5.73	209	44	42.742	42.796	146	44
DSJC250.1	250	3218	?	tl	1	4	-	-	0	9
DSJC250.5	250	15668	?	tl	808	25.36	25.165	25.168	31	31
DSJC250.9	250	27897	?	tl	58292	70.936	70.392	70.522	358	73
DSJC500.1	500	12458	?	tl	1	5	-	-	0	14
DSJC500.5	500	62624	?	tl	4	42.267	42.264	42.265	8	54
DSJC500.9	500	112437	?	tl	5886	122.599	122.307	122.334	370	132
DSJR500.5	500	58862	122	290.19	123	122	122.0	122.0	803	122
abb313GPIA	1557	65390	4	tl	1	8	-	-	0	9
ash608GPIA	1216	7844	4	tl	1	3	-	-	0	4
ash958GPIA	1916	12506	4	tl	1	3	-	-	0	4
flat1000_50_0	1000	245000	50	488.16	1	50	50.0	50.0	0	50
flat1000_60_0	1000	245830	60	tl	1	19.055	-	-	0	60
flat1000_76_0	1000	246708	76	tl	1	18.804	-	-	0	96
flat300_28_0	300	21695	28	tl	140	27.608	27.52	27.522	22	34
latin_square_10	900	307350	?	tl	261	90	90.0	90.0	95	109

Table 7.3: Results on the DIMACS instances for strategy C_3

Instance	$\|V\|$	$\|E\|$	$\chi(G)$	time (s)	B&B nodes	$\underline{\chi(G)}$	$\chi_f(G)$	$\chi_c(G)$	$\|\mathcal{H}\|_{\text{oc}}$	Found UB
le450_15a	450	8168	15	tl	14	15	15.0	15.0	4	16
le450_15b	450	8169	15	tl	7	15	15.0	15.0	7	16
le450_15c	450	16680	15	tl	1	15	-	-	0	16
le450_15d	450	16750	15	tl	1	15	-	-	0	16
le450_25c	450	17343	25	tl	5	25	25.0	25.0	28	28
le450_25d	450	17425	25	tl	12	25	25.0	25.0	38	28
le450_5a	450	5714	5	tl	1	5	-	-	0	6
le450_5b	450	5734	5	tl	1	5	-	-	0	6
le450_5d	450	9757	5	tl	1	5	-	-	0	6
myciel3	11	20	4	0.07	5	4	2.9	2.9	0	4
myciel4	23	71	5	0.22	77	5	3.245	3.245	0	5
myciel5	47	236	6	4.81	3511	6	3.553	3.553	0	6
myciel6	95	755	7	tl	131995	5.4	3.834	3.834	0	7
myciel7	191	2360	8	tl	7964	4.898	4.095	4.095	0	8
qg.order40	1600	62400	40	tl	1	40	40.0	40.0	418	41
queen10_10	100	2940	11	tl	24543	10.157	10.0	10.0	33	12
queen11_11	121	3960	11	7.82	5	11	11.0	11.0	35	11
queen12_12	144	5192	12	tl	769	12	12.0	12.0	3	14
queen13_13	169	6656	13	tl	124	13	13.0	13.0	6	15
queen14_14	196	8372	14	tl	29	14	14.0	14.0	1	16
queen15_15	225	10360	15	tl	17	15	15.0	15.0	0	17
queen16_16	256	12640	16	tl	3	16	16.0	16.0	0	18
queen9_9	81	2112	10	4.0	21	10	9.0	9.0	62	10
wap01a	2368	110871	?	tl	1	41	-	-	0	47
wap02a	2464	111742	?	tl	1	40	-	-	0	44
wap03a	4730	286722	?	tl	1	40	-	-	0	48
wap04a	5231	294902	?	tl	1	40	-	-	0	46
wap06a	947	43571	40	tl	28	40	40.0	40.0	54	42
wap07a	1809	103368	?	tl	1	40	-	-	0	45
wap08a	1870	104176	?	tl	1	40	-	-	0	45

Table 7.4: Results on the DIMACS instances for strategy C_4

Instance	$\|V\|$	$\|E\|$	$\chi(G)$	time (s)	B&B nodes	$\underline{\chi(G)}$	$\chi_f(G)$	$\chi_c(G)$	$\|\mathcal{H}\|_{\text{oc}}$	$\|\mathcal{H}\|_{\text{modk}}$	Found UB
1-FullIns_4	93	593	5	0.07	15	5	3.633	4.0	15	18	5
1-FullIns_5	282	3247	6	3.63	1713	6	3.909	4.063	19	70	6
1-Insertions_4	67	232	5	tl	79191	4	2.774	3.0	8	17	5
1-Insertions_5	202	1227	?	tl	1	2.943	2.943	-	0	7	6
1-Insertions_6	607	6337	?	tl	1	2.497	-	-	0	0	7
2-FullIns_4	212	1621	6	0.06	23	6	4.485	4.485	3	10	6
2-FullIns_5	852	12201	7	5.67	4003	7	4.708	4.783	26	55	7
2-Insertions_3	37	72	4	0.18	91	4	2.423	3.0	4	18	4
2-Insertions_4	149	541	?	tl	1964	3.2	2.56	3.0	30	18	5
2-Insertions_5	597	3936	?	tl	1	2.198	-	-	0	0	6
3-FullIns_4	405	3524	7	0.06	19	7	5.392	5.392	15	14	7
3-FullIns_5	2030	33751	8	3.21	2069	8	5.578	6.0	20	34	8
3-Insertions_3	56	110	4	5.1	495	4	2.334	3.0	6	4	4
3-Insertions_4	281	1046	?	tl	184	3	2.438	3.0	3	4	5
3-Insertions_5	1406	9695	?	tl	1	2.026	-	-	0	0	6
4-FullIns_4	690	6650	8	0.08	13	8	6.329	7.0	38	15	8
4-FullIns_5	4146	77305	9	7.69	3737	9	6.487	6.531	39	113	9
4-Insertions_3	79	156	4	91.64	1109	4	2.276	3.0	6	4	4
4-Insertions_4	475	1795	?	tl	1	2.039	-	-	0	0	5
5-FullIns_4	1085	11395	9	0.12	15	9	7.283	7.283	24	22	9
C1000.9	1000	450079	?	tl	68	215.483	215.354	215.421	538	1	241

Table 7.4: Results on the DIMACS instances for strategy C_4

Instance	$\|V\|$	$\|E\|$	$\chi(G)$	time (s)	B&B nodes	$\underline{\chi(G)}$	$\chi_f(G)$	$\chi_c(G)$	$\|\mathcal{H}\|_{\text{oc}}$	$\|\mathcal{H}\|_{\text{modk}}$	Found UB
C2000.5	2000	999836	?	tl	1	21.86	-	-	0	0	175
C2000.9	2000	1799532	?	tl	1	389.6	389.6	389.6	297	31	446
C4000.5	4000	4000268	?	tl	1	25.077	-	-	0	0	326
DSJC1000.1	1000	49629	?	tl	1	6	-	-	0	0	23
DSJC1000.5	1000	249826	?	tl	1	19.152	-	-	0	0	97
DSJC1000.9	1000	449449	?	tl	64	215.019	214.855	214.942	706	23	241
DSJC125.1	125	736	5	tl	795	4.729	4.454	4.454	16	1	6
DSJC125.5	125	3891	17	tl	47726	16.191	15.727	15.744	65	7	18
DSJC125.9	125	6961	44	5.95	209	44	42.742	42.796	146	3	44
DSJC250.1	250	3218	?	tl	1	4	-	-	0	0	9
DSJC250.5	250	15668	?	tl	812	25.36	25.165	25.168	31	1	31
DSJC250.9	250	27897	?	tl	61616	70.956	70.392	70.522	358	7	73
DSJC500.1	500	12458	?	tl	1	5	-	-	0	0	14
DSJC500.5	500	62624	?	tl	4	42.267	42.264	42.265	8	0	54
DSJC500.9	500	112437	?	tl	6296	122.603	122.307	122.334	378	17	132
DSJR500.5	500	58862	122	431.92	287	122	122.0	122.0	803	1	122
abb313GPIA	1557	65390	4	tl	1	8	-	-	0	0	9
ash608GPIA	1216	7844	4	tl	1	3	-	-	0	0	4
ash958GPIA	1916	12506	4	tl	1	3	-	-	0	0	4
flat1000_50_0	1000	245000	50	478.94	1	50	50.0	50.0	0	0	50
flat1000_60_0	1000	245830	60	tl	1	19.055	-	-	0	0	60
flat1000_76_0	1000	246708	76	tl	1	18.804	-	-	0	0	96
flat300_28_0	300	21695	28	tl	146	27.611	27.52	27.522	22	0	34
latin_square_10	900	307350	?	tl	244	90	90.0	90.0	95	3	109
le450_15a	450	8168	15	tl	15	15	15.0	15.0	4	2	16
le450_15b	450	8169	15	tl	7	15	15.0	15.0	7	4	16
le450_15c	450	16680	15	tl	1	15	-	-	0	0	16
le450_15d	450	16750	15	tl	1	15	-	-	0	0	16
le450_25c	450	17343	25	tl	10	25	25.0	25.0	28	7	28
le450_25d	450	17425	25	tl	12	25	25.0	25.0	38	0	28
le450_5a	450	5714	5	tl	1	5	-	-	0	0	6
le450_5b	450	5734	5	tl	1	5	-	-	0	0	6
le450_5d	450	9757	5	tl	1	5	-	-	0	0	6
myciel3	11	20	4	0.09	1	4	2.9	3.026	19	7	4
myciel4	23	71	5	0.17	77	5	3.245	3.362	10	10	5
myciel5	47	236	6	7.73	3555	6	3.553	3.622	19	8	6
myciel6	95	755	7	tl	24148	5.217	3.834	3.912	0	10	7
myciel7	191	2360	8	tl	6823	4.898	4.095	4.147	6	16	8
qg.order40	1600	62400	40	tl	1	40	40.0	40.0	418	7	41
queen10_10	100	2940	11	tl	23338	10.158	10.0	10.0	33	1	12
queen11_11	121	3960	11	103.98	78	11	11.0	11.0	35	13	11
queen12_12	144	5192	12	tl	925	12	12.0	12.0	3	1	14
queen13_13	169	6656	13	tl	124	13	13.0	13.0	6	0	15
queen14_14	196	8372	14	tl	12	14	14.0	14.0	17	12	16
queen15_15	225	10360	15	tl	17	15	15.0	15.0	0	0	17
queen16_16	256	12640	16	tl	6	16	16.0	16.0	0	5	18
queen9_9	81	2112	10	3.87	19	10	9.0	9.0	42	26	10
wap01a	2368	110871	?	tl	1	41	-	-	0	0	47
wap02a	2464	111742	?	tl	1	40	-	-	0	0	44
wap03a	4730	286722	?	tl	1	40	-	-	0	0	48
wap04a	5231	294902	?	tl	1	40	-	-	0	0	46
wap06a	947	43571	40	tl	28	40	40.0	40.0	54	2	42
wap07a	1809	103368	?	tl	1	40	-	-	0	0	45
wap08a	1870	104176	?	tl	1	40	-	-	0	0	45

Table 7.5: Results on the DIMACS instances for strategy C_5

Instance	$\|V\|$	$\|E\|$	$\chi(G)$	time (s)	B&B nodes	$\underline{\chi(G)}$	$\|\mathcal{H}\|_{\text{oc}}$	$\|\mathcal{H}\|_{\text{modk}}$	Found UB
1-FullIns_4	93	593	5	0.07	7	5	29	20	5
1-FullIns_5	282	3247	6	4.28	1249	6	852	224	6
1-Insertions_4	67	232	5	tl	9023	3.642	12741	3113	5
1-Insertions_5	202	1227	?	tl	1	2.943	0	7	6
1-Insertions_6	607	6337	?	tl	1	2.497	0	0	7
2-FullIns_4	212	1621	6	0.06	13	6	47	23	6
2-FullIns_5	852	12201	7	37.31	3799	7	2705	498	7
2-Insertions_3	37	72	4	0.37	127	4	144	49	4
2-Insertions_4	149	541	?	tl	1158	3.172	3778	720	5
2-Insertions_5	597	3936	?	tl	1	2.198	0	0	6
3-FullIns_4	405	3524	7	0.07	13	7	70	24	7
3-FullIns_5	2030	33751	8	7.63	1655	8	1219	245	8
3-Insertions_3	56	110	4	4.76	445	4	315	86	4
3-Insertions_4	281	1046	?	tl	140	3	1300	222	5
3-Insertions_5	1406	9695	?	tl	1	2.026	0	0	6
4-FullIns_4	690	6650	8	0.09	13	8	102	16	8
4-FullIns_5	4146	77305	9	17.55	2635	9	1363	386	9
4-Insertions_3	79	156	4	164.93	1129	4	912	206	4
4-Insertions_4	475	1795	?	tl	1	2.039	0	0	5
5-FullIns_4	1085	11395	9	0.13	15	9	81	28	9
C1000.9	1000	450079	?	tl	20	215.467	1535	1	243
C2000.5	2000	999836	?	tl	1	21.86	0	0	175
C2000.9	2000	1799532	?	tl	1	389.6	297	31	446
C4000.5	4000	4000268	?	tl	1	25.077	0	0	326
DSJC1000.1	1000	49629	?	tl	1	6	0	0	23
DSJC1000.5	1000	249826	?	tl	1	19.152	0	0	97
DSJC1000.9	1000	449449	?	tl	15	214.986	1866	23	243
DSJC125.1	125	736	5	tl	672	4.736	2096	298	6
DSJC125.5	125	3891	17	tl	4669	16.062	20063	137	18
DSJC125.9	125	6961	44	8.84	133	44	1251	3	44
DSJC250.1	250	3218	?	tl	1	4	0	0	9
DSJC250.5	250	15668	?	tl	473	25.36	1235	8	31
DSJC250.9	250	27897	?	tl	1904	70.91	42147	8	74
DSJC500.1	500	12458	?	tl	1	5	0	0	14
DSJC500.5	500	62624	?	tl	3	42.265	24	0	54
DSJC500.9	500	112437	?	tl	727	122.548	12221	17	132
DSJR500.5	500	58862	?	tl	59	122	1990	1	133
abb313GPIA	1557	65390	4	tl	1	8	0	0	9
ash608GPIA	1216	7844	4	tl	1	3	0	0	4
ash958GPIA	1916	12506	4	tl	1	3	0	0	4
flat1000_50_0	1000	245000	50	475.78	1	50	0	0	50
flat1000_60_0	1000	245830	60	tl	1	19.055	0	0	60
flat1000_76_0	1000	246708	76	tl	1	18.804	0	0	96
flat300_28_0	300	21695	28	tl	106	27.604	165	6	34
latin_square_10	900	307350	?	tl	111	90	692	8	109
le450_15a	450	8168	15	tl	8	15	91	17	16
le450_15b	450	8169	15	tl	29	15	63	36	16
le450_15c	450	16680	15	tl	1	15	0	0	16
le450_15d	450	16750	15	tl	1	15	0	0	16
le450_25c	450	17343	25	tl	17	25	53	151	28
le450_25d	450	17425	25	tl	14	25	62	17	28
le450_5a	450	5714	5	tl	1	5	0	0	6
le450_5b	450	5734	5	tl	1	5	0	0	6
le450_5d	450	9757	5	tl	1	5	0	0	6
myciel3	11	20	4	0.1	1	4	19	7	4

Table 7.5: Results on the DIMACS instances for strategy C_5

Instance	$\|V\|$	$\|E\|$	$\chi(G)$	time (s)	B&B nodes	$\underline{\chi(G)}$	$\|\mathcal{H}\|_{\text{oc}}$	$\|\mathcal{H}\|_{\text{modk}}$	Found UB
myciel4	23	71	5	0.37	63	5	162	57	5
myciel5	47	236	6	25.98	3657	6	300	136	6
myciel6	95	755	7	tl	4203	4.885	3414	1536	7
myciel7	191	2360	8	tl	3089	4.862	4808	2764	8
qg.order40	1600	62400	40	tl	1	40	418	7	41
queen10_10	100	2940	11	tl	619	10.035	2123	292	12
queen11_11	121	3960	11	38.79	14	11	228	24	11
queen12_12	144	5192	12	tl	657	12	968	181	14
queen13_13	169	6656	13	tl	84	13	271	209	15
queen14_14	196	8372	14	tl	13	14	66	18	16
queen15_15	225	10360	15	tl	5	15	51	37	17
queen16_16	256	12640	16	tl	8	16	25	15	18
queen9_9	81	2112	10	4.35	17	10	157	39	10
wap01a	2368	110871	?	tl	1	41	0	0	47
wap02a	2464	111742	?	tl	1	40	0	0	44
wap03a	4730	286722	?	tl	1	40	0	0	48
wap04a	5231	294902	?	tl	1	40	0	0	46
wap06a	947	43571	40	tl	55	40	191	5	42
wap07a	1809	103368	?	tl	1	40	0	0	45
wap08a	1870	104176	?	tl	1	40	0	0	45