Isabel Pretterhofer, BSc

An Exploration of Column Generation and Branch-and-Price With Special Regard to the Fair Matching Over Time Problem

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieurin

Master's degree programme: Mathematics

submitted to

Graz University of Technology

Supervisor

Eranda Dragoti-Çela, Ao.Univ.-Prof. Dipl.-Ing. Dr.techn.

Institute of Discrete Mathematics

Graz, August 2024

Abstract

Column generation is an algorithm for handling linear programs with many variables by starting with a small subset of the variables and iteratively adding further variables if needed as the algorithm progresses. Branch-and-price algorithms apply column generation at every node of a branch-and-bound tree to deal with suitably structured mixed integer linear programs. In this thesis we investigate these two algorithms and apply branch-and-price to the fair matching over time problem. This problem is a generalization of the balanced assignment problem, where we look not only at bipartite graphs but at general graphs and add a time horizon. The time horizon is the number of perfect matchings we choose. The branch-and-price algorithm for this problem is implemented using the branch-and-price framework SCIP with the interface PySCIPOpt and applied to solve randomly generated instances of the fair matching over time problem on graphs of different types. The empirical analysis shows that our branch-and-price algorithm successfully reduces the symmetry of the fair matching over time problem. In the empirical study the running time of the algorithm is strongly correlated with the order of the graph and its density and only slightly with the time horizon. Furthermore, we observe that instances on graphs with a high number of vertices and a low maximal degree or a low average degree can often be solved quickly.

Kurzfassung

Spaltengenerierung ist ein Algorithmus zur Lösung von linearen Programmen mit einer sehr großen Anzahl an Variablen, bei welchem ausgehend von einer kleinen Teilmenge der Variablen iterativ bei Bedarf mehr Variablen hinzugefügt werden. Branch-and-Price (engl. für verzweigen und bepreisen) ist ein Algorithmus zur Lösung von gemischt ganzzahligen linearen Programmen mit spezieller Struktur. Dabei wird in jedem Knoten eines branch-and-bound (engl. für verzweigen und beschränken) Baumes Spaltengenerierung angewendet. In dieser Arbeit werden diese beiden Methoden genauer betrachtet und es wird ein Branch-and-Price Algorithmus für das faire Matching Problem über Zeit aufgestellt und implementiert. Das faire Matching Problem über Zeit ist eine Generalisierung des ausgeglichenen Zuordnungsproblems (engl. balanced assignment problem), wobei nicht nur bipartite, sondern allgemeine Graphen betrachtet werden und ein Zeithorizont hinzugefügt wird. Hierbei ist der Zeithorizont die Anzahl der gesuchten Matchings. Der Branch-and-Price Algorithmus für dieses Problem wird mit dem Branch-and-Price Framework SCIP mit der Schnittstelle PySCIPOpt implementiert und wird auf zufällig generierte Instanzen des fairen Matching Problems für verschiedene Arten von Graphen angewendet. Eine empirische Analyse hat einerseits gezeigt, dass unser Branch-and-Price Algorithmus die Symmetrie im fairen Matching Problem über Zeit erfolgreich reduziert. Andererseits wurde ersichtlich, dass die Laufzeit dieses Algorithmus von der Ordnung und Dichte des Graphen stark abhängig ist, während der Zeithorizont einen geringen Einfluss hat. Es wurde festgestellt, dass Instanzen mit Graphen mit vielen Knoten und einem niedrigen maximalen Knotengrad oder einem niedrigen durchschnittlichen Knotengrad häufig schnell mit diesem Algorithmus gelöst werden können.

Contents

List of Algorithms

1 Introduction

Mixed integer linear programs are a way to model various problems in combinatorial optimization and it is therefore highly desirable to be able to solve them efficiently. However, in general this is a burdensome undertaking. Many of these problems can be reformulated and subsequently solved more efficiently using a so-called branch-and-price approach. When the reformulation is such that there is a large number of variables, we can solve the linear relaxation of the problem by starting with a small subset of the variables and then iteratively adding more promising variables until we have found the optimal solution. This is called column generation. The method was introduced by Gilmore and Gomory for the cutting stock problem, see [GG63], and by Dantzig and Wolfe, see [DW60]. A major part of its effectiveness is revealed when solving mixed integer linear problems by performing branch-and-bound, where we use column generation at every node of the branch-and-bound tree. This is called branch-and-price. Branch-and-price was introduced by Vance, Barnhart, Johnson, and Nemhauser, see [Van+94], and has been successfully used to solve many problems, including vehicle routing problems [Des+95], crew scheduling problems [Des+95] and coloring problems [LNS24]. The research on branch-and-price is ongoing, very recently a book by Desrosiers, Lübbecke, Desaulniers and Gauthier on branch-and-price was published, see [Des+24]. The branch-and-price procedure can be combined with the branch-and-cut method by occasionally adding cutting planes instead of branching. This yields the branch-price-and-cut method.

During the last years there has been an increasing interest in combinatorial optimization to not only find solutions which optimize some cost function, but to find solutions which are as fair as possible to a certain number of stakeholders, see [BFT11]. To this end, one needs to define some notion of fairness. This is done using an unfairness function which rates how unfair a certain distribution of utilities is. Minimizing this unfairness can however lead to a low overall utility, resulting in low global welfare. Thus, one has to make a trade-off between fairness and welfare. When the same optimization problem occurs multiple times it is often possible to achieve more fairness and better welfare by not always making the same decision. Thus, we try to find a sequence of solutions such that the overall utilities the stakeholders get are as fair as possible, while securing a certain level of welfare. This concept is called fairness over time. In [Lod+22] a specific fairness over time problem has been solved using branch-and-price. Here, we want to solve a generalization of the balanced assignment problem, which tries to find a fair assignment of workers to tasks, see [NBN22a] and [NBN22b]. We generalize the balanced assignment problem by considering general simple graphs and adding a time horizon. This yields a fairness over time problem, which we call the fair matching over time problem.

In this thesis we give an overview on column generation and branch-and-price. In particular, in the context of column generation we address questions such as suitable problem formulations, usage of Lagrangian relaxation and the tailing-off effect and stabilization in Chapter 2.

In Chapter 3 we discuss the branch-and-price method, several possibilities to branch and heuristics to get good upper bounds for the branch-and-bound tree are considered. Finally, we briefly introduce the concept of branch-price-and-cut in terms of two particular methods to add cuts, in the compact formulation and in the extended formulation.

In Chapter 4 we concisely introduce the notion of fairness over time, establish the fair matching over time problem, and describe how it can be reformulated and solved using branch-and-price. This branch-and-price algorithm for the fair matching over time problem was implemented using the branch-and-price framework SCIP [Bol+24] with the interface PySCIPOpt [Mah+16] from Python to SCIP. At the end of Chapter 4 computational experiments on this implementation are documented. We generate random instances using different types of random graphs, random weights and set different time horizons. We set time limits for the solving process of 5 or 10 seconds and observe the ratio of instances which are solved to optimality to instances which contain a perfect matching. We observe that this ratio is strongly dependent on the order of the graph and its density and only weakly dependent on the size of the time horizon. Furthermore, the algorithm can solve large instances quickly when the graph has a limited maximal degree or a limited average degree.

Finally, we summarize the findings of this thesis and formulate some open questions for further research.

2 Column Generation

2.1 Introduction to Column Generation

We now want to introduce the method of column generation as in [DL05] and [GGBM13]. Consider the following linear program, which we call the master problem.

$$\begin{aligned} \min_{(x_1,...,x_n)} \quad & \sum_{j\in J} c_j x_j \\ \text{s.t.} \quad & \sum_{j\in J} \boldsymbol{a}_j x_j \geq \boldsymbol{b}, \\ & x_j \geq 0, \quad \forall j \in J, \end{aligned} \tag{2.1}$$

where $|J| = n$, $(c_1, ..., c_n) \in \mathbb{R}^n$, $\boldsymbol{a}_j \in \mathbb{R}^m$ for all $j \in J$ and $\boldsymbol{b} \in \mathbb{R}^m$. When $|J|$ is exceptionally large, solving this linear program directly will be difficult. Thus, we only consider a subset $J' \subseteq J$ of the variables and solve the so-called restricted master problem.

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \sum_{j\in J'} c_j x_j \\ \text{s.t.} \quad & \sum_{j\in J'} \boldsymbol{a}_j x_j \geq \boldsymbol{b}, \\ & x_j \geq 0, \quad \forall j \in J'. \end{aligned} \tag{2.2}$$

We initially choose J' such that the restricted master problem is feasible. Then we iteratively solve the restricted master problem and add $j \in J \setminus J'$ to J', until the optimal solution of the restricted master problem is already optimal for the master problem. Adding j to J' adds the column $\boldsymbol{a}_j$, the corresponding cost c_j and variable x_j to the restricted master problem, hence the name column generation. By adding only promising j's, the hope is that the number of columns in the restricted master problem stays significantly below the number of columns in the master problem.

If choosing J' such that the initial restricted master problem is feasible is not possible, the master problem is not feasible itself, as we can choose $J' = J$. From now on, we assume that the master problem is feasible and that we choose the initial J' such that the corresponding restricted master problem is feasible.

Regarding the choice of which j to add to J', let $\boldsymbol{x}^* = (\boldsymbol{x}_B, \boldsymbol{0})$ be an optimal primal basic solution of the restricted master problem where we add columns $\boldsymbol{a}_{n+1}, ..., \boldsymbol{a}_{n+m}$ as slack variables. We assume that the submatrix $(\boldsymbol{a}_1, ..., \boldsymbol{a}_m)$ of $(\boldsymbol{a}_1, ..., \boldsymbol{a}_n, \boldsymbol{a}_{n+1}, ..., \boldsymbol{a}_{n+m})$ has full

rank and corresponds to the basis B. The reduced cost coefficients of the variables $\boldsymbol{x_j}$ for $j \in J \setminus J'$ are then defined as $c_j - (c_1, ..., c_m)(\boldsymbol{a}_1, ..., \boldsymbol{a}_m)^{-1}\boldsymbol{a}_j$, see [LY84]. Then we can see that $(c_1, ..., c_m)(\boldsymbol{a}_1, ..., \boldsymbol{a}_m)^{-1}$ is feasible for the dual of the restricted master problem, as follows. We have

$$(c_1, ..., c_{|J'|}) - (c_1, ..., c_m)(\boldsymbol{a}_1, ..., \boldsymbol{a}_m)^{-1}(\boldsymbol{a}_1, ..., \boldsymbol{a}_{|J'|}) \geq \boldsymbol{0},$$

due to the fact that B is a basis of an optimal basic solution. And hence,

$$(\boldsymbol{a}_1, ..., \boldsymbol{a}_{|J'|})^\intercal((c_1, ..., c_m)(\boldsymbol{a}_1, ..., \boldsymbol{a}_m)^{-1})^\intercal \leq (c_1, ..., c_{|J'|})^\intercal.$$

Furthermore, $\boldsymbol{u}^* \coloneqq (c_1, ..., c_m)(\boldsymbol{a}_1, ..., \boldsymbol{a}_m)^{-1}$ is an optimal solution of the dual restricted master problem, as $(c_1, ..., c_m)(\boldsymbol{a}_1, ..., \boldsymbol{a}_m)^{-1}\boldsymbol{b} = (c_1, ..., c_m)\boldsymbol{x}_B = \sum_{i=1}^{m} c_i x_i^*$ and we have weak duality. Thus we can rewrite the reduced cost as $c_j - \boldsymbol{u}^{*\intercal}\boldsymbol{a}_j$.

Observe that if the reduced cost $c_j - \boldsymbol{u}^{*\intercal}\boldsymbol{a}_j$ of all variables x_j, $j \in J \setminus J'$ is non-negative, we already have that the reduced cost of all $x_j, j \in J$ is non-negative and hence the solution is already optimal for the master problem. Otherwise, there exists a variable $x_j, j \in J \setminus J'$ with negative reduced cost, which we can add to J' and get a potentially better optimal primal solution for the updated restricted master problem. Indeed, when the restricted master problem is feasible for J', it is also feasible for $J' \cup \{j\}$ and cannot have a larger objective value, as we can always choose $x_j = 0$ and take the values of all other variables as in a feasible solution for J'. If we iterate this process, we will eventually get an optimal solution of the master problem. We call this step, where we want to find a variable with negative reduced cost, the pricing problem or subproblem.

This yields the column generation procedure as in Algorithm 1.

Algorithm 1 Column Generation

Input feasible master problem
Output optimal primal and dual solution of the master problem

1: Find small J' such that the restricted master problem is feasible.
2: Solve the restricted master problem, $\boldsymbol{x}^*$ primal solution, $\boldsymbol{u}^*$ dual solution.
3: **if** there exists a variable with negative reduced cost **then**
4: add one or more variables to J', at least one having negative reduced cost.
5: **go to** 2
6: **else return** $(\boldsymbol{x}^*, \boldsymbol{u}^*)$
7: **end if**

Theorem 2.1.1. *The column generation algorithm yields optimal primal and dual solutions for a feasible master problem in finite time whenever solving the restricted master problem is done in finite time.*

Proof. The termination of this procedure is guaranteed whenever finding an optimal primal and dual solution of the restricted master problem in every iteration terminates within a finite amount of time, as we add at least one variable in each iteration, and we can have at most $|J| = n$ variables. The termination of the optimization of the restricted master problem can for instance be taken for granted when using the simplex algorithm with suitable variable selection to prevent cycling.

The optimality of the solution is clear, as we stop prior to having $J' = J$ only in the case of non-negative reduced cost for all variables in J. □

As any basic feasible solution of the master problem contains no more than m non-zero variables, it is likely that we can consider a rather small set J' and still reach a solution which is already optimal to the master problem.

In order to find a variable with negative reduced cost with index in $J \setminus J'$ to enter J', one possibility is to solve $\min_{j \in J \setminus J'} \{c_j - \boldsymbol{u}^{*\intercal} \boldsymbol{a}_j\}$. If $\min_{j \in J \setminus J'} \{c_j - \boldsymbol{u}^{*\intercal} \boldsymbol{a}_j\} \geq 0$ we know that there is no variable with negative reduced cost and if $\min_{j \in J \setminus J'} \{c_j - \boldsymbol{u}^{*\intercal} \boldsymbol{a}_j\} < 0$ we can add the index corresponding to a variable which attains this minimum to J'. Computing $\min_{j \in J \setminus J'} \{c_j - \boldsymbol{u}^{*\intercal} \boldsymbol{a}_j\}$ can often be done without explicitly computing all reduced costs, but rather as some kind of a combinatorial optimization problem, which might be solvable more efficiently. This is especially true, when the columns $\boldsymbol{a}_j$ are only given implicitly as elements of a non-empty set $\mathcal{A}$, whose elements all follow some known rule and are only made explicit when needed.

Notice that it is also possible to add multiple variables at once in each iteration of the column generation procedure. Indeed, this might speed up the process, as it can lead to less iterations.

Finding an initial J' such that the restricted master problem is feasible can often be done using an ad-hoc method depending on the specific problem, but it can also be done more systematically, see [Van05]. For instance, it can be done by introducing artificial columns with a cost which is high enough so that the artificial columns do not appear in the final solution. However, when we choose the cost of the artificial variables too high, this results in weak bounds on the dual variables and can lead to adding unnecessary columns to the restricted master problem, see [LD05]. Hence, we want to keep the cost as high as necessary but as low as possible. If we have chosen the costs for the artificial variables too low so that they stay relevant for too long, we may restart the procedure with higher cost.

Note that column generation is classically done by using the simplex algorithm

- to solve the restricted master problem in every iteration, which yields a primal and dual solution
- and to find a new column by finding the variable with the most negative reduced cost.

However, many different variations of column generation have been considered and come

with different advantages, such as better convergence, see for instance Section 2.3 and 2.4.

2.1.1 A Dual Point of View

The dual of the master problem (2.1) looks as follows.

$$\begin{aligned} \max_{\boldsymbol{u}} \quad & \boldsymbol{b}^\intercal \boldsymbol{u} \\ \text{s.t.} \quad & \boldsymbol{a}_j^\intercal \boldsymbol{u} \leq c_j, \quad \forall j \in J, \\ & \boldsymbol{u} \geq 0 \end{aligned} \tag{2.3}$$

Let us now look at the column generation procedure from a Lagrangian dual view, as in [Bri+08] and [Bri+05]. Recall that the Lagrangian subproblem of the master problem (2.1) with all constraints dualized is

$$L(\boldsymbol{u}) \coloneqq \min_{\boldsymbol{x} \geq 0} \{\boldsymbol{c}^\intercal \boldsymbol{x} - \boldsymbol{u}^\intercal (A\boldsymbol{x} - \boldsymbol{b})\} \tag{2.4}$$

where $A = (\boldsymbol{a}_1, ..., \boldsymbol{a}_n)$ and $\boldsymbol{c} = (c_1, ..., c_n)$. For every $\boldsymbol{u} \geq 0$ this yields a lower bound on the optimal objective value z_{MP} of the master problem (2.1), as for an optimal primal solution of the master problem $\boldsymbol{x}_{MP}$ we have that $\boldsymbol{u}^\intercal (A\boldsymbol{x}_{MP} - \boldsymbol{b}) \geq 0$. The Lagrangian dual of Problem (2.1) consists of maximizing this lower bound.

$$z_{LD} = \max_{\boldsymbol{u} \geq 0} L(\boldsymbol{u}) = \max_{\boldsymbol{u} \geq 0} \min_{\boldsymbol{x} \geq 0} \{\boldsymbol{c}^\intercal \boldsymbol{x} - \boldsymbol{u}^\intercal (A\boldsymbol{x} - \boldsymbol{b})\} \tag{2.5}$$

For the restricted master problem (2.2) we can do the same and get the Lagrangian subproblem of the restricted master problem

$$L^{RMP}(\boldsymbol{u}) \coloneqq \min_{\boldsymbol{x} \geq 0} \{\tilde{\boldsymbol{c}}^\intercal \boldsymbol{x} - \boldsymbol{u}^\intercal (\tilde{A}\boldsymbol{x} - \boldsymbol{b})\}, \tag{2.6}$$

where $\tilde{\boldsymbol{c}}$ and $\tilde{A}$ consist of the entries and columns of $\boldsymbol{c}$ and A corresponding to J', respectively. The Lagrangian dual of the restricted master problem (2.2) looks as follows.

$$z_{RLD} = \max_{\boldsymbol{u} \geq 0} L^{RMP}(\boldsymbol{u}) = \max_{\boldsymbol{u} \geq 0} \min_{\boldsymbol{x} \geq 0} \{\tilde{\boldsymbol{c}}^\intercal \boldsymbol{x} - \boldsymbol{u}^\intercal (\tilde{A}\boldsymbol{x} - \boldsymbol{b})\}. \tag{2.7}$$

We can rewrite the Lagrangian dual of the master problem

$$z_{LD} = \max_{\boldsymbol{u} \geq 0} \min_{\boldsymbol{x} \geq 0} \{\boldsymbol{c}^\intercal \boldsymbol{x} - \boldsymbol{u}^\intercal (A\boldsymbol{x} - \boldsymbol{b})\} = \max_{\boldsymbol{u} \geq 0} \{s : s \leq \boldsymbol{u}^\intercal \boldsymbol{b} + (\boldsymbol{c}^\intercal - \boldsymbol{u}^\intercal A)\, \boldsymbol{x}, \, \forall \boldsymbol{x} \geq 0\} \tag{2.8}$$

and observe that when we set $s = \boldsymbol{u}^\intercal \boldsymbol{b}$, we recognize the linear programming dual of the master problem (2.3). This works analogously for the restricted master problem.

Using a Lagrangian subproblem we can achieve the following dual lower bound for the optimal solution of the master problem. This gives us an idea of the quality of the current solution when we stop the column generation procedure before reaching optimality in the master problem, see [DL05].

Theorem 2.1.2. *Let z_{MP} be the optimal objective value of the master problem and z_{RMP} and $\boldsymbol{u}_{RMP}^\intercal$ the optimal objective value and the optimal dual solution of the restricted master problem for a fixed J', respectively. Furthermore, let κ be an upper bound on the sum of the entries of an optimal solution $(x_1^*, ..., x_n^*)$ of the master problem, i.e. $\kappa \geq \sum_{j=1}^n x_j^*$. Then the following bounds hold*

$$z_{RMP} + \kappa \min_{j \in J}\{c_j - \boldsymbol{u}_{RMP}^\intercal \boldsymbol{a}_j\} \leq z_{MP} \leq z_{RMP}.$$

Proof. As any feasible solution of the restricted master problem can be transformed into a feasible solution of the master problem with the same objective value by keeping the values for x_j if $j \in J'$ and setting $x_j = 0$ for $j \in J \setminus J'$, it holds that $z_{MP} \leq z_{RMP}$.

For the lower bound, we consider the Lagrangian subproblem

$$z_{MP} \geq L(\boldsymbol{u}_{RMP}) = \min_{\boldsymbol{x} \geq 0}\{\boldsymbol{c}^\intercal \boldsymbol{x} - \boldsymbol{u}_{RMP}^\intercal (A\boldsymbol{x} - \boldsymbol{b})\} = \min_{\boldsymbol{x} \geq 0}\{\boldsymbol{u}_{RMP}^\intercal \boldsymbol{b} \ + \boldsymbol{c}^\intercal \boldsymbol{x} - \boldsymbol{u}_{RMP}^\intercal A\boldsymbol{x}\}$$

$$= z_{RMP} + \min_{\boldsymbol{x} \geq 0}\{\boldsymbol{c}^\intercal \boldsymbol{x} - \boldsymbol{u}_{RMP}^\intercal A\boldsymbol{x}\} = z_{RMP} + \min_{\boldsymbol{x} \geq 0}\{\sum_{j=1}^n (c_j x_j - \boldsymbol{u}_{RMP}^\intercal \boldsymbol{a}_j x_j)\}$$

$$= z_{RMP} + \min_{\boldsymbol{x} \geq 0}\{\sum_{j=1}^n x_j (c_j - \boldsymbol{u}_{RMP}^\intercal \boldsymbol{a}_j)\} \geq z_{RMP} + \min_{\boldsymbol{x} \geq 0}\{\sum_{j=1}^n x_j \min_{j \in J}\{c_j - \boldsymbol{u}_{RMP}^\intercal \boldsymbol{a}_j\}\}$$

$$\geq z_{RMP} + \max_{\boldsymbol{x} \geq 0}\{\sum_{j=1}^n x_j\} \min_{j \in J}\{c_j - \boldsymbol{u}_{RMP}^\intercal \boldsymbol{a}_j\} \geq z_{RMP} + \kappa \min_{j \in J}(c_j - \boldsymbol{u}_{RMP}^\intercal \boldsymbol{a}_j).$$

Here the key insights are that the Lagrangian subproblem is a lower bound, $\boldsymbol{u}_{RMP}^\intercal \boldsymbol{b} = z_{RMP}$ by strong duality, $\min_{j \in J}(c_j - \boldsymbol{u}_{RMP}^\intercal \boldsymbol{a}_j) \leq 0$ and thus we minimize a negative number. Hence, maximizing the absolute value of the individual factors makes the expression smaller. Thus, it holds that $z_{RMP} + \kappa \min_{j \in J}\{c_j - \boldsymbol{u}_{RMP}^\intercal \boldsymbol{a}_j\} \leq z_{MP}$. □

When we add columns in the primal, this corresponds to adding rows in the dual, as $A^\intercal$ is used in the dual instead of A. This means, that when adding columns $\boldsymbol{a}_j$ with the negative reduced cost $c_j - \boldsymbol{u}^{*\intercal} \boldsymbol{a}_j \leq 0$ in the primal, this corresponds to adding violated constraints $\boldsymbol{u}^\intercal \boldsymbol{a}_j \geq c_j$ to the dual problem, where $\boldsymbol{u}^*$ is an optimal solution to the previous dual restricted master problem. Hence, when performing column generation for the primal problem, we actually perform a cutting-plane method for the dual problem.

The cutting-plane method in the dual corresponding to column generation in the primal would be as in Algorithm 2.

From this cutting-plane algorithm we can obtain an optimal primal solution $\boldsymbol{x}^*$ of the master problem by solving the primal restricted master problem with J' at the end of the cutting plane algorithm once.

Algorithm 2 Cutting-Plane Algorithm

Input feasible dual master problem
Output optimal solution of the dual master problem

1: Choose small J'.
2: Solve the dual restricted master problem, to get solution $\boldsymbol{u}^*$.
3: **if** there exists a constraint in $J \setminus J'$ which is violated **then**
4: add one or more constraints to J', at least one being violated.
5: **go to** 2
6: **else return** $\boldsymbol{u}^*$
7: **end if**

Observe that contrary to the primal algorithm, it is possible to start with $J' = \emptyset$.

Also notice, that

$$z_{LD} = \max_{\boldsymbol{u} \geq 0} L(\boldsymbol{u}) \geq L(\operatorname{argmax}_{\boldsymbol{u} \geq 0} L^{RMP}(\boldsymbol{u})), \tag{2.9}$$

but $L(\operatorname{argmax}_{\boldsymbol{u} \geq 0} L^{RMP}(\boldsymbol{u}))$ is not monotonically increasing over the iterations, see [Van05] and [BADF04]. Hence it can be meaningful to track the best lower bound for z_{LD} of this form.

Definition 2.1.3 (Stability Center)**.** *Let $\boldsymbol{u}^k := \operatorname{argmax}_{\boldsymbol{u} \geq 0} L^{RMP}(\boldsymbol{u})$ for the restricted master problem in iteration k, i.e. $\boldsymbol{u}^k$ is the optimal solution of the Lagrangian dual of the restricted master problem in iteration k. Then the best lower bound for z_{LD} of this form found until round l is $\hat{\boldsymbol{u}} := \operatorname{argmax}_{k \in \{1,\ldots,l\}} L(\boldsymbol{u}^k)$, which we call the stability center after round l.*

2.2 Dantzig-Wolfe Decomposition

We now want to look at how to reformulate integer linear problems with a special structure such that the solution of the linear relaxation of the reformulation

- provides a good approximation of the original integer linear program,
- and can be obtained in a decently efficient way using column generation.

Whenever possible we try to reformulate the original problem such that the pricing problem can be solved well. For instance, it is sometimes possible to solve the subproblem by using a fairly efficient combinatorial algorithm.

One such possibility of using the structure of the problem to get a good reformulation is the Dantzig-Wolfe decomposition. We are going to introduce the concept of the Dantzig-Wolfe decomposition as in [DL05] and [VW10].

Consider the following integer linear program, which we call the compact formulation of the program.

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{c}^\intercal \boldsymbol{x} \\ \text{s.t.} \quad & A_1 \boldsymbol{x} \geq \boldsymbol{b_1}, \\ & A_2 \boldsymbol{x} \geq \boldsymbol{b_2}, \\ & \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0}, \end{aligned} \tag{2.10}$$

where $\boldsymbol{c} \in \mathbb{R}^k$, $A_1 \in \mathbb{R}^{m \times k}$, $A_2 \in \mathbb{R}^{l \times k}$, $\boldsymbol{b_1} \in \mathbb{R}^m$, $\boldsymbol{b_2} \in \mathbb{R}^l$. For mixed-integer programs the following works analogously, see [Van05].

2.2.1 Convexification

The convex hull $conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2 \boldsymbol{x} \geq \boldsymbol{b_2}\}$ is a polyhedron and can thus be represented by a finite set of extreme points and rays using the theorem of Minkowski and Weyl.

Theorem 2.2.1 (Minkowski, Weyl)**.** *Let $X = \{\boldsymbol{x} \in \mathbb{R}^k : A\boldsymbol{x} \leq \boldsymbol{b}\}$ be a polyhedron. Then there exists a finite set of extreme points $\{\boldsymbol{x}_p : p \in P\}$ of X and a finite set of extreme rays $\{\boldsymbol{v}_r : r \in R\}$ of X such that X can be represented as a combination of the extreme rays plus a convex combination of the extreme points, i.e.*

$$X = \{\boldsymbol{x} \in \mathbb{R}^k : \boldsymbol{x} = \sum_{p \in P} \boldsymbol{x}_p \lambda_p + \sum_{r \in R} \boldsymbol{v}_r \lambda_r, \sum_{p \in P} \lambda_p = 1, \lambda_p \in \mathbb{R}_{\geq 0} \, \forall p \in P, \lambda_r \in \mathbb{R}_{\geq 0} \, \forall r \in R\}.$$

Thus, we can write $conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2 \boldsymbol{x} \geq \boldsymbol{b_2}\} = \{\boldsymbol{x} \in \mathbb{R}^k : \boldsymbol{x} = \sum_{p \in P} \boldsymbol{x}_p \lambda_p + \sum_{r \in R} \boldsymbol{v}_r \lambda_r, \sum_{p \in P} \lambda_p = 1, \lambda_p \in \mathbb{R}_{\geq 0} \, \forall p \in P, \lambda_r \in \mathbb{R}_{\geq 0} \, \forall r \in R\}$ for some finite set of extreme points $\{\boldsymbol{x}_p : p \in P\}$ and a finite set of extreme rays $\{\boldsymbol{v}_r : r \in R\}$. If we plug this into the compact formulation (2.10), we get the following extensive formulation.

$$\begin{aligned} \min_{\boldsymbol{\lambda}} \quad & \sum_{p \in P} c_p \lambda_p + \sum_{r \in R} c_r \lambda_r \\ \text{s.t.} \quad & \sum_{p \in P} \boldsymbol{a}_p \lambda_p + \sum_{r \in R} \boldsymbol{a}_r \lambda_r \geq \boldsymbol{b_1}, \\ & \sum_{p \in P} \lambda_p = 1, \\ & \boldsymbol{\lambda} \geq \boldsymbol{0}, \\ & \sum_{p \in P} \boldsymbol{x}_p \lambda_p + \sum_{r \in R} \boldsymbol{v}_r \lambda_r = \boldsymbol{x}, \\ & \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0}, \end{aligned} \tag{2.11}$$

where $c_p = \boldsymbol{c}^\intercal \boldsymbol{x}_p, c_r = \boldsymbol{c}^\intercal \boldsymbol{v}_r, \boldsymbol{a}_p = A_1 \boldsymbol{x}_p, \boldsymbol{a}_r = A_1 \boldsymbol{v}_r$ for all $p \in P, r \in R$ and $\boldsymbol{\lambda} \in \mathbb{R}^{|P|+|R|}$ in the concrete notation of $(\lambda_p)_{p \in P}, (\lambda_r)_{r \in R}$. The extensive formulation is equivalent to the compact formulation. This kind of extensive formulation is also called the convexification of the compact formulation, see [VW10].

The extensive formulation has $m + 1 + k$ constraints additionally to the non-negativity and integrality constraints, which can be less than the $m + l$ constraints of the compact formulation. However, the number of variables is only k in the compact formulation and can be exponential in l in the extensive formulation.

In order to deal with this potentially large amount of variables, we can use column generation to solve the linear relaxation of the extensive formulation (2.11). Notice, that if we relax the integrality of $\boldsymbol{x}$, there is no reason to link $\boldsymbol{x}$ and $\boldsymbol{\lambda}$, thus we have the following linear relaxation of (2.11).

$$\begin{aligned} \min_{\boldsymbol{\lambda}} \quad & \sum_{p \in P} c_p \lambda_p + \sum_{r \in R} c_r \lambda_r \\ \text{s.t.} \quad & \sum_{p \in P} \boldsymbol{a}_p \lambda_p + \sum_{r \in R} \boldsymbol{a}_r \lambda_r \geq \boldsymbol{b_1}, \\ & \sum_{p \in P} \lambda_p = 1, \\ & \boldsymbol{\lambda} \geq \boldsymbol{0} \end{aligned} \tag{2.12}$$

We solve Problem (2.12) by column generation, i.e. Problem (2.12) is our master problem. Then we take an initial subset of the columns $P' \subseteq P$, $R' \subseteq R$ to get the following restricted master problem. Here we make sure to choose P' and R' such that the restricted master problem is feasible.

$$\begin{aligned} \min_{\boldsymbol{\lambda}} \quad & \sum_{p \in P'} c_p \lambda_p + \sum_{r \in R'} c_r \lambda_r \\ \text{s.t.} \quad & \sum_{p \in P'} \boldsymbol{a}_p \lambda_p + \sum_{r \in R'} \boldsymbol{a}_r \lambda_r \geq \boldsymbol{b_1}, \\ & \sum_{p \in P'} \lambda_p = 1, \\ & \boldsymbol{\lambda} \geq \boldsymbol{0} \end{aligned} \tag{2.13}$$

Let $(\boldsymbol{u}^{*\intercal}, u_0^*)$ be an optimal dual solution of the restricted master problem at a certain point of time, where $\boldsymbol{u}^*$ corresponds to the constraints $\sum_{p \in P} \boldsymbol{a}_p \lambda_p + \sum_{r \in R} \boldsymbol{a}_r \lambda_r \geq \boldsymbol{b_1}$ and u_0^* corresponds to the convexity constraint $\sum_{p \in P} \lambda_p = 1$. Then the pricing problem is $\min\{\min_{p \in P}\{c_p - \boldsymbol{u}^{*\intercal} \boldsymbol{a}_p - u_0^*\}, \min_{r \in R}\{c_r - \boldsymbol{u}^{*\intercal} \boldsymbol{a}_r\}\}$. We can also find the extreme point or extreme ray with the least reduced cost by using the following Lemma.

Lemma 2.2.2. *In the notation from above, it holds that*

$$\inf\{\boldsymbol{c}^\intercal \boldsymbol{x} - \boldsymbol{u}^{*\intercal} A_1 \boldsymbol{x} - u_0^* : \boldsymbol{x} \in conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2 \boldsymbol{x} \geq \boldsymbol{b_2}\}\} \in$$

$$\begin{cases} (\infty, 0], & \textit{if there are no extreme points and extreme rays with negative reduced cost}, \\ (0, -\infty), & \textit{if there is an extreme point, but no extreme ray with negative reduced cost}, \\ \{-\infty\}, & \textit{if there exists an extreme ray with negative reduced cost}. \end{cases}$$

Proof. First of all, notice that $\boldsymbol{x} \in conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}$ are precisely the points which can be written as $\boldsymbol{x} = \sum_{p\in P} \boldsymbol{x}_p\lambda_p + \sum_{r\in R} \boldsymbol{v}_r\lambda_r$ for some $\boldsymbol{\lambda} \geq \boldsymbol{0}$ with $\sum_{p\in P} \lambda_p = 1$. Thus, we have that $\boldsymbol{c}^\intercal\boldsymbol{x} - \boldsymbol{u}^{*\intercal}A_1\boldsymbol{x} - u_0^* = \sum_{p\in P} \lambda_p(c_p - \boldsymbol{u}^{*\intercal}\boldsymbol{a}_p - u_0^*) + \sum_{r\in R} \lambda_r(c_r - \boldsymbol{u}^{*\intercal}\boldsymbol{a}_r)$, as $\sum_{p\in P} \lambda_p = 1$, $\boldsymbol{c}^\intercal\boldsymbol{x}_p = c_p$, $\boldsymbol{c}^\intercal\boldsymbol{v}_r = c_r$, $A_1\boldsymbol{x}_p = \boldsymbol{a}_p$, $A_1\boldsymbol{v}_r = \boldsymbol{a}_r$ and linearity. Furthermore, observe that when $\min_{r\in R}\{c_r - \boldsymbol{u}^{*\intercal}\boldsymbol{a}_r\}$ is negative, this means that there exists a ray $\boldsymbol{v}_{r_0}$ with $r_0 \in R$ such that $\boldsymbol{c}^\intercal\boldsymbol{v}_{r_0} - \boldsymbol{u}^{*\intercal}A_1\boldsymbol{v}_{r_0} < 0$. If we have some fixed $\boldsymbol{\lambda} \geq \boldsymbol{0}$ with $\sum_{p\in P} \lambda_p = 1$, then $\boldsymbol{x^{(0)}} := \sum_{p\in P} \boldsymbol{x}_p\lambda_p + \sum_{r\in R} \boldsymbol{v}_r\lambda_r$ is in $conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}$. Then, also $\boldsymbol{x^{(1)}} := \sum_{p\in P} \boldsymbol{x}_p\lambda_p + \sum_{r\in R} \boldsymbol{v}_r\lambda_r + \boldsymbol{v}_{r_0}d$ is in $conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}$ for any fixed $d \in \mathbb{R}_{\geq 0}$. However, $\boldsymbol{c}^\intercal\boldsymbol{x^{(1)}} - \boldsymbol{u}^{*\intercal}A_1\boldsymbol{x^{(1)}} - u_0^* = \boldsymbol{c}^\intercal\boldsymbol{x^{(0)}} - \boldsymbol{u}^{*\intercal}A_1\boldsymbol{x^{(0)}} - u_0^* + d(\boldsymbol{c}^\intercal\boldsymbol{v}_{r_0} - \boldsymbol{u}^{*\intercal}A_1\boldsymbol{v}_{r_0})$. Therefore, we have $\inf\{\boldsymbol{c}^\intercal\boldsymbol{x} - \boldsymbol{u}^{*\intercal}A_1\boldsymbol{x} - u_0^* : \boldsymbol{x} \in conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}\} = -\infty$ in this case.

Let us now consider the case where $\min_{r\in R}\{c_r - \boldsymbol{u}^{*\intercal}\boldsymbol{a}_r\} \geq 0$. Consider again $\boldsymbol{x^{(0)}}$ as above. Then also $\boldsymbol{x^{(2)}} := \sum_{p\in P} \boldsymbol{x}_p\lambda_p$ is in $conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}$. But, $\boldsymbol{c}^\intercal\boldsymbol{x^{(0)}} - \boldsymbol{u}^{*\intercal}A_1\boldsymbol{x^{(0)}} - u_0^* = \boldsymbol{c}^\intercal\boldsymbol{x^{(2)}} - \boldsymbol{u}^{*\intercal}A_1\boldsymbol{x^{(2)}} - u_0^* + \sum_{r\in R} \lambda_r(c_r - \boldsymbol{u}^{*\intercal}\boldsymbol{a}_r) \geq \boldsymbol{c}^\intercal\boldsymbol{x^{(2)}} - \boldsymbol{u}^{*\intercal}A_1\boldsymbol{x^{(2)}} - u_0^* \geq \min_{p\in P}\{c_p - \boldsymbol{u}^{*\intercal}\boldsymbol{a}_p - u_0^*\} > -\infty$. Accordingly, we have $\inf\{\boldsymbol{c}^\intercal\boldsymbol{x} - \boldsymbol{u}^{*\intercal}A_1\boldsymbol{x} - u_0^* : \boldsymbol{x} \in conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}\} > -\infty$ in this case. Furthermore, we know that the minimum is attained at an extreme point, as for $p_0 = \operatorname{argmin}_{p\in P}\{c_p - \boldsymbol{u}^{*\intercal}\boldsymbol{a}_p - u_0^*\}$ we have that $\boldsymbol{x}_{p_0} \in conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}$. Hence, we know that $\inf\{\boldsymbol{c}^\intercal\boldsymbol{x} - \boldsymbol{u}^{*\intercal}A_1\boldsymbol{x} - u_0^* : \boldsymbol{x} \in conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}\} = \min_{p\in P}\{c_p - \boldsymbol{u}^{*\intercal}\boldsymbol{a}_p - u_0^*\}$, whenever $\min_{r\in R}\{c_r - \boldsymbol{u}^{*\intercal}\boldsymbol{a}_r\} \geq 0$.

□

Therefore, instead of solving the pricing problem, we can solve the linear program $\min\{\boldsymbol{c}^\intercal\boldsymbol{x} - \boldsymbol{u}^{*\intercal}A_1\boldsymbol{x} - u_0^* : \boldsymbol{x} \in conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}\} = \min\{\boldsymbol{c}^\intercal\boldsymbol{x} - \boldsymbol{u}^{*\intercal}A_1\boldsymbol{x} - u_0^* : \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}$. Then, if the optimal objective value is positive, we conclude that we are already optimal for the master problem (2.12) and stop the column generation procedure. In the case where it is negative and finite, the minimum is attained at an extreme point. In this case we can add this extreme point to the restricted master problem. Otherwise, we find an extreme ray with negative reduced cost and add it to the restricted master problem, see [LD05] and [DL05].

Observe, that Lemma 2.2.2 also holds for $\inf\{\boldsymbol{c}^\intercal\boldsymbol{x} - \boldsymbol{u}^{*\intercal}A_1\boldsymbol{x} - u_0^* : A_2\boldsymbol{x} \geq \boldsymbol{b_2}, \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0}\}$. This is due to the fact, that we minimize a linear function. When we assume that all constraints are rational we can argue more constructively, that $\boldsymbol{x}_p$ are integral points for all $p \in P$, and the extreme rays $r \in R$ start at an integral point and can be scaled such that they end in another integral point. Therefore, if $\boldsymbol{x^{(0)}} := \sum_{p\in P} \boldsymbol{x}_p\lambda_p + \sum_{r\in R} \boldsymbol{v}_r\lambda_r$ is in $\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}$, then for all $r_0 \in R$ there exist arbitrarily large $d \in \mathbb{R}_{\geq 0}$ such that $\boldsymbol{x^{(1)}} := \sum_{p\in P} \boldsymbol{x}_p\lambda_p + \sum_{r\in R} \boldsymbol{v}_r\lambda_r + \boldsymbol{v}_{r_0}d$ is also in $\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}$.

Example (Convexification)**.** *Consider the following compact formulation of an integer linear program.*

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & x_1 + x_2 \\ \text{s.t.} \quad & 6x_1 + x_2 \geq 6, \\ & x_1 + 2x_2 \geq 6, \\ & -x_1 + x_2 \geq -3, \\ & \boldsymbol{x} \in \mathbb{Z}^2_{\geq 0}. \end{aligned} \tag{2.14}$$

Suppose we want to convexify the constraints $x_1 + 2x_2 \geq 6$ *and* $-x_1 + x_2 \geq -3$*. Hence, we need the extreme points and rays of the polyhedron* $conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : x_1 + 2x_2 \geq 6, -x_1 + x_2 \geq -3\}$*. The set of extreme points is* $\{(4,1)^\intercal, (0,3)^\intercal\}$ *and the set extreme rays is* $\{(1,1)^\intercal, (0,1)^\intercal\}$*. Then, we have* $\boldsymbol{c} = (1,1)$*,* $A_1 = (6,1)$ *and* $\boldsymbol{b_1} = 6$*. It follows, that* $c_{(4,1)} = 5$*,* $c_{(0,3)} = 3$*,* $c_{(1,1)} = 2$*,* $c_{(0,1)} = 1$ *and* $\boldsymbol{a}_{(4,1)} = 25$*,* $\boldsymbol{a}_{(0,3)} = 3$*,* $\boldsymbol{a}_{(1,1)} = 7$*,* $\boldsymbol{a}_{(0,1)} = 1$*. Hence, the extended formulation after the Dantzig-Wolfe convexification looks as follows.*

$$\begin{aligned} \min_{\boldsymbol{\lambda}} \quad & 5\lambda_{(4,1)} + 3\lambda_{(0,3)} + 2\lambda_{(1,1)} + \lambda_{(0,1)} \\ \text{s.t.} \quad & 25\lambda_{(4,1)} + 3\lambda_{(0,3)} + 7\lambda_{(1,1)} + \lambda_{(0,1)} \geq 6, \\ & \lambda_{(4,1)} + \lambda_{(0,3)} = 1, \\ & \boldsymbol{\lambda} \geq \boldsymbol{0}, \\ & (4,1)^\intercal\lambda_{(4,1)} + (0,3)^\intercal\lambda_{(0,3)} + (1,1)^\intercal\lambda_{(1,1)} + (0,1)^\intercal\lambda_{(0,1)} = \boldsymbol{x}, \\ & \boldsymbol{x} \in \mathbb{Z}^2_{\geq 0} \end{aligned} \tag{2.15}$$

The master problem is the linear relaxation of the extended formulation.

$$\begin{aligned} \min_{\boldsymbol{\lambda}} \quad & 5\lambda_{(4,1)} + 3\lambda_{(0,3)} + 2\lambda_{(1,1)} + \lambda_{(0,1)} \\ \text{s.t.} \quad & 25\lambda_{(4,1)} + 3\lambda_{(0,3)} + 7\lambda_{(1,1)} + \lambda_{(0,1)} \geq 6, \\ & \lambda_{(4,1)} + \lambda_{(0,3)} = 1, \\ & \boldsymbol{\lambda} \geq \boldsymbol{0} \end{aligned} \tag{2.16}$$

The restricted master problem is the master problem with a subset of the columns. We want to initially choose the columns of the restricted master problem such that it is feasible. For instance, we could take the column corresponding to the extreme point $(0,3)$ *and the extreme ray* $(0,1)$ *to initialize the restricted master problem. This looks as follows.*

$$\begin{aligned} \min_{\boldsymbol{\lambda}} \quad & 3\lambda_{(0,3)} + \lambda_{(0,1)} \\ \text{s.t.} \quad & 3\lambda_{(0,3)} + \lambda_{(0,1)} \geq 6, \\ & \lambda_{(0,3)} = 1, \\ & \boldsymbol{\lambda} \geq \boldsymbol{0} \end{aligned} \tag{2.17}$$

After the column generation procedure finishes, we have an optimal solution to the linear programming relaxation of the Dantzig-Wolfe formulation and are left to deal with the integrality constraint. Notice, that if we require $\boldsymbol{\lambda}$ to be integer in (2.12), we are not

guaranteed to receive the optimal solution for Problem (2.11), as there are integer $\boldsymbol{x}$ which do not correspond to an integer $\boldsymbol{\lambda}$.

Example. *Consider the following integer linear program.*

$$\begin{aligned} \min \quad & x \\ \text{s.t.} \quad & x \geq 0.5, \\ & -x \geq -2, \\ & x \in \mathbb{Z}_{\geq 0}, \end{aligned}$$

where $A_1 = (1)$, $b_1 = 0.5$, $A_2 = (-1)$, $b_2 = -2$. *Then the polyhedron* $conv\{x \in \mathbb{Z}_{\geq 0} : A_2 x \geq b_2, x \geq 0\}$ *has the two extreme points* $x_1 = 0$ *and* $x_2 = 2$. *Thus,* $conv\{x \in \mathbb{Z}_{\geq 0} : A_2 x \geq b_2, x \geq 0\} = \{x \in \mathbb{R}_{\geq 0} : x = 0 * \lambda_1 + 2 * \lambda_2, \lambda_1 + \lambda_2 = 1, \lambda_1, \lambda_2 \geq 0\}$. *Hence, for integer* λ_1, λ_2 *we can only reach* $x = 0$ *or* $x = 2$. *As* $x = 0$ *is not feasible for* $x \geq 0.5$, *the best optimal solution with integer* λ_1, λ_2 *would be* $x = 2$. *However, the optimal solution to the original integer linear program is* $x = 1$.

Therefore, we will look at another version of the Dantzig-Wolfe decomposition.

2.2.2 Discretization

To directly represent all integer points $\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}$, one can use the following theorem in [NW88].

Theorem 2.2.3. *Let* $X = \{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A\boldsymbol{x} \geq \boldsymbol{b}\}$ *be the set of all integer points in the polyhedron* $\{\boldsymbol{x} \in \mathbb{R}^k_{\geq 0} : A\boldsymbol{x} \geq \boldsymbol{b}\}$. *Then there are finitely many integer points* $\{\boldsymbol{x}_p : p \in P\}$ *and finitely many integer rays* $\{\boldsymbol{v}_r : r \in R\}$ *such that every* $\boldsymbol{x} \in X$ *can be represented as* $\boldsymbol{x} = \sum_{p \in P} \boldsymbol{x}_p \lambda_p + \sum_{r \in R} \boldsymbol{v}_r \lambda_r$, *where* $\sum_{p \in P} \lambda_p = 1$ *and* $\boldsymbol{\lambda}_s \in \mathbb{Z}_{\geq 0}$ *for all* $s \in P \cup R$.

In contrast to Theorem 2.2.1, we describe precisely the integer points $\boldsymbol{x}$ with $A\boldsymbol{x} \geq \boldsymbol{b}$ and not all points in the convex hull $conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A\boldsymbol{x} \geq \boldsymbol{b}\}$. When $conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A\boldsymbol{x} \geq \boldsymbol{b}\}$ is a bounded polytope, Theorem 2.2.1 only takes the extreme points into its set of points, while the set of points in Theorem 2.2.3 is $\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A\boldsymbol{x} \geq \boldsymbol{b}\}$. Normally, we will therefore have a larger set of points when using Theorem 2.2.3. In Theorem 2.2.1 it can thus happen that $\boldsymbol{\lambda}$ is not integral, but $\boldsymbol{x}$ is. Whereas in Theorem 2.2.3 the convexity constraint $\sum_{p \in P} \lambda_p = 1$ together with $\boldsymbol{\lambda} \in \mathbb{Z}_{\geq 0}$ implies that we have exactly one $p_0 \in P$ with $\lambda_{p_0} = 1$ and $\lambda_p = 0$ for all $p \in P \setminus \{p_0\}$.

We use the representation of Theorem 2.2.3 for $\boldsymbol{x}$ in Problem (2.10), which yields the

method of discretization, see [Van00].

$$\begin{aligned} \min_{\boldsymbol{\lambda}} \quad & \sum_{p\in P} c_p\lambda_p + \sum_{r\in R} c_r\lambda_r \\ \text{s.t.} \quad & \sum_{p\in P} \boldsymbol{a}_p\lambda_p + \sum_{r\in R} \boldsymbol{a}_r\lambda_r \geq \boldsymbol{b_1}, \\ & \sum_{p\in P} \lambda_p = 1, \\ & \lambda_s \in \mathbb{Z}_{\geq 0} \quad \forall s \in P \cup R, \end{aligned} \tag{2.18}$$

where $c_p = \boldsymbol{c}^\intercal \boldsymbol{x}_p, c_r = \boldsymbol{c}^\intercal \boldsymbol{v}_r, \boldsymbol{a}_p = A_1\boldsymbol{x}_p, \boldsymbol{a}_r = A_1\boldsymbol{v}_r$ for all $p \in P, r \in R$. The linear relaxation of Problem (2.18) can again be solved using column generation and one only needs to think about the integrality of $\boldsymbol{\lambda}$. Notice, that the linear relaxation of the discretization (2.18) has the same structure as the linear relaxation of Problem (2.12) attained by convexification.

Example (Discretization)**.** *We now want to use Dantzig-Wolfe discretization on the compact formulation of the convexification example from above. Suppose that we want to use discretization for the constraints* $x_1 + 2x_2 \geq 6$ *and* $-x_1 + x_2 \geq -3$*. Hence, we need integer points and integer rays which describe* $\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : x_1 + 2x_2 \geq 6, -x_1 + x_2 \geq -3\}$*. The set of points is* $\{(4,1)^\intercal, (3,2)^\intercal, (2,2)^\intercal, (1,3)^\intercal, (0,3)^\intercal\}$ *and the set rays is* $\{(1,1)^\intercal, (0,1)^\intercal\}$*. Given these points, we can compute* c_s *and* $\boldsymbol{a}_s$ *for* $s \in P \cup R$ *and construct the master problem similarly to the convexification approach.*

2.2.3 Generators

More generally, whenever we have a finite set of so called generators such that every point in $\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}$ can be represented as a weighted sum of generators, where the weights follow certain rules, we get a reformulation of our Problem (2.10) in a similar way as above, see [Van05], [VS06].

Definition 2.2.4 (Generating Set, Generator)**.** *Let* G^{A_2} *be a finite set of points in* $\mathbb{R}^k$*. Then* G^{A_2} *is called a generating set for* $\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}$*, if there exists a set of weights* $W^{A_2} \subseteq \mathbb{R}^{|G^{A_2}|}$*, which can be described using only linear and integer constraints, such that* $\{\sum_{\boldsymbol{g}\in G^{A_2}} \boldsymbol{g}\lambda_{\boldsymbol{g}} : \boldsymbol{\lambda} \in W^{A_2}\} = \{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}$*. The elements of the generating set are called generators.*

A generating set G^{A_2} together with a suitable set of weights W^{A_2} yields the following reformulation of Problem (2.10).

$$\begin{aligned} \min_{\boldsymbol{\lambda}} \quad & \sum_{\boldsymbol{g}\in G^{A_2}} \boldsymbol{c}^\intercal \boldsymbol{g}\lambda_{\boldsymbol{g}} \\ \text{s.t.} \quad & \sum_{\boldsymbol{g}\in G^{A_2}} A_1\boldsymbol{g}\lambda_{\boldsymbol{g}} \geq \boldsymbol{b_1}, \\ & \boldsymbol{\lambda} \in W^{A_2}. \end{aligned} \tag{2.19}$$

For the pricing step in the column generation procedure, one has to solve $\min\{\boldsymbol{c}^\intercal \boldsymbol{g} - \boldsymbol{u}^{*\intercal} A_1 \boldsymbol{g} : \boldsymbol{g} \in G^{A_2}\}$, where u^* is the optimal dual solution of the current restricted master problem.

The convexification and the discretization of the compact formulation are special cases of the reformulation using generating sets.

Example (Convexification)**.** *For the convexification of Problem (2.10), let $\{\boldsymbol{x}_p : p \in P\}$ be a set of extreme points and $\{\boldsymbol{v}_r : r \in R\}$ a set of extreme rays of the polyhedron $conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}$, from the Theorem of Minkowski and Weyl. When we plug in $G^{A_2} = \{\boldsymbol{x}_p : p \in P\} \cup \{\boldsymbol{v}_r : r \in R\}$ and $W^{A_2} = \{\boldsymbol{\lambda} \in \mathbb{R}^{|G^{A_2}|} : \sum_{p\in P} \lambda_p = 1, \sum_{p\in P} \boldsymbol{x}_p\lambda_p + \sum_{r\in R} \boldsymbol{v}_r\lambda_r \in \mathbb{Z}^k_{\geq 0}\}$ into (2.19), we get Problem (2.11).*

Example (Discretization)**.** *For the discretization of Problem (2.10), let $\{\boldsymbol{x}_p : p \in P\}$ be a set of integer points and $\{\boldsymbol{v}_r : r \in R\}$ a set of integer rays describing the set $\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}$ as in Theorem 2.2.3. When we plug in $G^{A_2} = \{\boldsymbol{x}_p : p \in P\} \cup \{\boldsymbol{v}_r : r \in R\}$ and $W^{A_2} = \{\boldsymbol{\lambda} \in \mathbb{R}^{|G^{A_2}|} : \sum_{p\in P} \lambda_p = 1, \lambda_s \in \mathbb{Z}_{\geq 0}\ \forall s \in P \cup R\}$ into (2.19), we get Problem (2.18).*

2.2.4 Choosing A_1 and A_2

Choosing which constraints to put into A_1 and which into A_2 is essential to be able to really use the structure of the problem and to get a good linear programming relaxation. Partitioning in the following three ways depending on how the constraints relate to each other often works well, see [VS06] and [Van05].

- **Difficult vs. Easy Constraints:** Choosing $A_1\boldsymbol{x} \geq \boldsymbol{b_1}$ as the difficult constraints and the subsystem $A_2\boldsymbol{x} \geq \boldsymbol{b_2}$ as the more efficiently solvable constraints can be a good choice, especially when optimizing over $A_2\boldsymbol{x} \geq \boldsymbol{b_2}$ can be done by using a combinatorial algorithm which is more efficient then solving the whole problem. When choosing this partitioning of the constraints, the pricing problem will be easier to solve.

- **Block Diagonal and Linking Constraints:** When A_2 has a block-diagonal structure and $A_1\boldsymbol{x} \geq \boldsymbol{b_1}$ are the constraints linking these blocks, we can describe the polyhedron of each block of A_2 individually. Let

$$A_2 = \begin{pmatrix} A_2^1 & & & \\ & A_2^2 & & \\ & & \ddots & \\ & & & A_2^d \end{pmatrix}, \boldsymbol{b_2} = \begin{pmatrix} \boldsymbol{b_2}^1 \\ \boldsymbol{b_2}^2 \\ \vdots \\ \boldsymbol{b_2}^d \end{pmatrix}.$$

  Let $G^{A_2^j}$ be a generating set for $\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2^j\boldsymbol{x} \geq \boldsymbol{b_2}^j\}$ with set of weights $W^{A_2^j}$, for each j in $\{1, ..., d\}$. Then the master problem looks as follows after the Dantzig-Wolfe decomposition.

$$\begin{aligned} \min_{\boldsymbol{\lambda}} \quad & \sum_{j=1}^{d} \sum_{\boldsymbol{g} \in G^{A_2^j}} \boldsymbol{c}^{j\intercal} \boldsymbol{g} \lambda_{\boldsymbol{g}}^j \\ \text{s.t.} \quad & \sum_{j=1}^{d} \sum_{\boldsymbol{g} \in G^{A_2^j}} A_1^j \boldsymbol{g} \lambda_{\boldsymbol{g}}^j \geq \boldsymbol{b_1}, \\ & \boldsymbol{\lambda}^j \in W^{A_2^j}, \forall j \in \{1, ..., d\}, \end{aligned} \tag{2.20}$$

where A_1^j are the columns of A_1 that correspond to the columns of A_2^j in A_2. E.g., if A_2^1 is a block of size l, then A_1^1 are the first l columns of A_1^1. And $\boldsymbol{c}^j$ are the entries of $\boldsymbol{c}$ corresponding to the columns of A_2^j.

The pricing problem is then decomposed of d separate pricing problems, $\min\{\boldsymbol{c}^\intercal \boldsymbol{g} - \boldsymbol{u}^{*\intercal} A_2^j \boldsymbol{g} : \boldsymbol{g} \in G^{A_2^j}\}$, for each j in $\{1, ..., d\}$, where $\boldsymbol{u}^*$ is the optimal dual solution of the current restricted master problem. Solving these subproblems can be stopped as soon as one column with negative reduced cost is found in one of the pricing problems, or it can be solved completely until the column with the overall most negative reduced cost is found.

It can happen that some of the blocks are identical, i.e. $A_2^j = A_2^S$ and $\boldsymbol{b_2}^j = \boldsymbol{b_2}^S$ for all j in a subset $S \subseteq \{1, ..., d\}$. Then, also the pricing problems are identical. Hence, we only need one pricing problem for each class of identical blocks. If further the $A_1^j = A_1^S$ and $\boldsymbol{c}^j = \boldsymbol{c}^S$ for all $j \in S$, then we can we can aggregate the values of the variables to one variable $\lambda_{\boldsymbol{g}}^S := \sum_{j \in S} \lambda_{\boldsymbol{g}}^j$ for all $j \in S$ in the master problem. The restrictions of $W^{A_2^S}$ are changed accordingly. Suppose that there are L different types of blocks which can be aggregated. Then the master problem looks as follows.

$$\begin{aligned} \min_{\boldsymbol{\lambda}} \quad & \sum_{l=1}^{L} \sum_{\boldsymbol{g} \in G^{A_2^{S_l}}} \boldsymbol{c}^{j\intercal} \boldsymbol{g} \lambda_{\boldsymbol{g}}^{S_l} \\ \text{s.t.} \quad & \sum_{l=1}^{L} \sum_{\boldsymbol{g} \in G^{A_2^{S_l}}} A_1^{S_l} \boldsymbol{g} \lambda_{\boldsymbol{g}}^{S_l} \geq \boldsymbol{b_1}, \\ & \boldsymbol{\lambda}^{S_l} \in W^{A_2^{S_l}}, \forall l \in \{1, ..., L\}, \end{aligned} \tag{2.21}$$

The symmetry of the problem is thereby reduced, see [DL11] and [Gam10].

- **Multiple Subsystems:** When $A_1 \boldsymbol{x} \geq \boldsymbol{b_1}$ and $A_2 \boldsymbol{x} \geq \boldsymbol{b_2}$ are both easier to solve on their own then together at once, choosing A_1 and A_2 such that we can optimize over the polyhedron induced by A_2 better can be beneficial as this facilitates the computation in the pricing step of column generation.

Example (Cutting Stock Problem)**.** *The cutting stock problem is a very classical application of column generation. We want to see how it can be formulated in order to efficiently use*

column generation, as in [BAC05].

The task in the one-dimensional cutting stock problem is to find the minimal number of rolls of length $L \in \mathbb{Z}_{\geq 0}$, such that we can cut from these rolls at least $d_i \in \mathbb{Z}_{\geq 0}$ pieces of length $l_i \in \mathbb{Z}_{\geq 0}$ for $i \in \{1, ..., m\}$.

A first integer linear programming model for this problem was introduced in [Kan60]. For this model, we have an upper bound on the minimal number of rolls U. Then we make U dummy-rolls $\boldsymbol{x}^j$ where we have a variable x_0^j which indicates whether the i-th dummy-roll is used and x_i^j which indicates how often we cut a piece of length l_i from it for $i \in \{1, ..., m\}$ and $j \in \{1, ..., U\}$.

$$\min_{\boldsymbol{x}_0} \quad \sum_{j=1}^{U} x_0^j \tag{2.22a}$$

$$\text{s.t.} \quad \sum_{j=1}^{U} x_i^j \geq d_i, \quad \forall i \in \{1, ..., m\}, \tag{2.22b}$$

$$\sum_{i=1}^{m} x_i^j l_i \leq L x_0^j, \quad \forall j \in \{1, ..., U\}, \tag{2.22c}$$

$$\boldsymbol{x}_i \in \mathbb{Z}_{\geq 0}^U, \quad \forall i \in \{1, ..., m\}, \tag{2.22d}$$

$$\boldsymbol{x}_0 \in \{0, 1\}^U. \tag{2.22e}$$

The constraints (2.22b) imply that at least d_i pieces of length l_i need to be cut for all $i \in \{1, ..., m\}$. The constraints (2.22c) mean that whenever pieces are cut from the j-th roll, we need to use it and that the total length of all pieces cut from one roll cannot exceed the length of a roll.

Notice that this problem has a lot of symmetry, as we can permute the rolls and still get the same result. Regarding a Dantzig-Wolfe decomposition of the problem, observe that the second set of constraints (2.22c) has a block-diagonal structure when represented with a matrix and the first set of constraints (2.22b) links the blocks together. Hence, we want to choose A_1 to model the first set of constraints (2.22b) and A_2 to model the second set of constraints (2.22c).

We want to generate the set $\{\boldsymbol{x}^j \in \mathbb{Z}_{\geq 0}^k : A_2^j \boldsymbol{x}^j \geq \boldsymbol{b_2}^j\} = \{\boldsymbol{x}^j \in \mathbb{Z}_{\geq 0}^k : \sum_{i=1}^m l_i x_i^j \leq L x_0^j\} = \{\boldsymbol{0}\} \cup \{(1, x_1^j, ..., x_m^j) : \sum_{i=1}^m l_i x_i^j \leq L\}$, i.e. this set is just the zero vector and the set of vectors starting with 1 and then having x_i^j the number of times a piece of length l_i is cut in a possible cutting of a roll. Those are finitely many, hence we can use all these points for a Dantzig-Wolfe discretization. As the zero vector is irrelevant to the problem, we can also say that the set of generators is the set of feasible cutting patterns $G^{A_2} = \{\boldsymbol{g} : (g_1, ..., g_m) : \sum_{i=1}^m l_i g_i \leq L\}$, where every element has a cost of 1. As all

blocks in the block-diagonal matrix have precisely the same structure, we can aggregate the variables and the problem simplifies.

This leads to the formulation of Gilmore and Gomory, as in [GG63].

$$\begin{aligned} \min_{\boldsymbol{\lambda}} \quad & \sum_{\boldsymbol{g} \in G^{A_2}} \lambda_{\boldsymbol{g}} \\ \text{s.t.} \quad & \sum_{\boldsymbol{g} \in G^{A_2}} g_i \lambda_{\boldsymbol{g}} \geq d_i, \quad \forall i \in \{1, ..., m\}, \\ & \lambda_{\boldsymbol{g}} \in \mathbb{Z}_{\geq 0}, \end{aligned} \tag{2.23}$$

Here, $\lambda_{\boldsymbol{g}}$ can be interpreted as the number of times the feasible cutting pattern $\boldsymbol{g}$ is chosen and g_i corresponds to the number of pieces of length l_i in the cutting pattern $\boldsymbol{g}$.

This can then be solved via column generation. Also, the subproblem for every block is the same, hence we can reduce it to solving just one subproblem.

$$\begin{aligned} \max_{\boldsymbol{g}} \quad & \boldsymbol{u}^{*\mathsf{T}} \boldsymbol{g} \\ \text{s.t.} \quad & \sum_{i=1}^{m} g_i l_i \leq L, \\ & \boldsymbol{g} \in \mathbb{Z}^m_{\geq 0}, \end{aligned} \tag{2.24}$$

where $\boldsymbol{u}^*$ is the optimal dual solution of the restricted master problem. As the cost for every feasible pattern is 1, we can just maximize over $\boldsymbol{u}^{*\mathsf{T}}\boldsymbol{g}$ in order to minimize the reduced cost. Notice that the subproblem is thus actually a knapsack problem.

2.2.5 Relation to Lagrangian Dual

The Dantzig-Wolfe Decomposition has a strong relation with the Lagrangian dual. We are now going to study this connection as in [VW10] and [LD05].

Recall that we form the Lagrangian subproblem of (2.10) by relaxing the "more complicated" constraints $A_1\boldsymbol{x} \geq \boldsymbol{b_1}$ and incorporate the violation of those constraints in the objective value.

$$L_{A_1}(\boldsymbol{u}) \coloneqq \min\{\boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} - \boldsymbol{u}^{\mathsf{T}}\left(A_1\boldsymbol{x} - \boldsymbol{b_1}\right) : A_2\boldsymbol{x} \geq \boldsymbol{b_2}, \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0}\} \tag{2.25}$$

For every $\boldsymbol{u} \geq 0$ this yields a lower bound on the optimal objective value $\boldsymbol{x}^*$ of Problem (2.10), as $\boldsymbol{u}^{\mathsf{T}}\left(A_1\boldsymbol{x}^* - \boldsymbol{b_1}\right) \geq 0$. The Lagrangian dual of Problem (2.10) consists of maximizing this lower bound.

$$z_{LD} = \max_{\boldsymbol{u} \geq 0} L_{A_1}(\boldsymbol{u}) = \max_{\boldsymbol{u} \geq 0} \min_{\boldsymbol{x}} \{\boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} - \boldsymbol{u}^{\mathsf{T}}\left(A_1\boldsymbol{x} - \boldsymbol{b_1}\right) : A_2\boldsymbol{x} \geq \boldsymbol{b_2}, \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0}\} \tag{2.26}$$

We now want to rewrite the Lagrangian dual as a linear program. Let $\{\boldsymbol{x}_p : p \in P\}$ be a set of extreme points and $\{\boldsymbol{v}_r : r \in R\}$ a set of extreme rays of the polyhedron

$conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}$. Notice that for some fixed $\boldsymbol{u}$, if there exists an extreme ray $\boldsymbol{v}_r$ such that $c_p - \boldsymbol{u}^\intercal \boldsymbol{a}_r < 0$, we already have $L_{A_1}(\boldsymbol{u}) = -\infty$. This is, because $\boldsymbol{c}^\intercal\boldsymbol{x} - \boldsymbol{u}^\intercal A_1\boldsymbol{x} - u_0$ and $\boldsymbol{c}^\intercal\boldsymbol{x} - \boldsymbol{u}^\intercal(A_1\boldsymbol{x} - \boldsymbol{b_1})$ only differ by a constant and Lemma 2.2.2 also holds for $\inf\{\boldsymbol{c}^\intercal\boldsymbol{x} - \boldsymbol{u}^{*\intercal}A_1\boldsymbol{x} - u_0^* : \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}$, as we observed in Section 2.2.1. The proof of Lemma 2.2.2 works analogously for any fixed $\boldsymbol{u}$ instead of $\boldsymbol{u}^*$, when we look at $c_p - \boldsymbol{u}^\intercal\boldsymbol{a}_r$ and $c_p - \boldsymbol{u}^\intercal\boldsymbol{x}_p - \boldsymbol{u}^\intercal\boldsymbol{b_1}$ instead of the reduced costs. Hence, either there exists a $\boldsymbol{u} \geq 0$ such that no such ray exists, or $z_{LD} = -\infty$. If no such ray exists, then for every given $\boldsymbol{u}$ there is some extreme point $p \in P$ such that $L_{A_1}(\boldsymbol{u}) = c_p - \boldsymbol{u}^\intercal\boldsymbol{x}_p - \boldsymbol{u}^\intercal\boldsymbol{b_1}$, again because Lemma 2.2.2 can be extended as above, see [LD05]. Assume that we know a set of extreme points $\{\boldsymbol{x}_p : p \in P\}$ and a set of extreme rays $\{\boldsymbol{v}_r : r \in R\}$ as in the Theorem of Minkowski and Weyl 2.2.1 and there exists no extreme ray with negative reduced cost. Then we can restate the Lagrangian dual as the following linear program.

$$\begin{aligned} \max_{\boldsymbol{u},\, v} \quad & \boldsymbol{u}^\intercal\boldsymbol{b_1} + v \\ \text{s.t.} \quad & \boldsymbol{u}^\intercal A_1\boldsymbol{x}_p + v \leq \boldsymbol{c}^\intercal\boldsymbol{x}_p, \quad \forall p \in P, \\ & \boldsymbol{u}^\intercal A_1\boldsymbol{v}_r \leq \boldsymbol{c}^\intercal\boldsymbol{v}_r, \quad \forall r \in R., \\ & \boldsymbol{u} \geq 0 \end{aligned} \tag{2.27}$$

The linear programming dual of Problem (2.27) looks as follows.

$$\begin{aligned} \min_{\boldsymbol{\lambda}} \quad & \sum_{p\in P} \boldsymbol{c}^\intercal\boldsymbol{x}_p\lambda_p + \sum_{r\in R} \boldsymbol{c}^\intercal\boldsymbol{v}_r\lambda_r \\ \text{s.t.} \quad & \sum_{p\in P} A_1\boldsymbol{x}_p\lambda_p + \sum_{r\in R} A_1\boldsymbol{v}_r\lambda_r \geq \boldsymbol{b_1}, \\ & \sum_{p\in P} \lambda_p = 1, \\ & \boldsymbol{\lambda} \geq \boldsymbol{0} \end{aligned} \tag{2.28}$$

Observe that this is the same linear program as the linear relaxation of the Dantzig-Wolfe convexification (2.12). Hence, the optimal solution of the Lagrangian dual is the same as the optimal solution of the linear relaxation of the Dantzig-Wolfe convexification when we choose the same partitioning of the constraints.

2.2.6 Strength of Dual Bounds of the Dantzig-Wolfe Reformulation

When looking at the linear relaxation of the Dantzig-Wolfe convexification of a problem, one achieves a lower bound on the optimal value of the original problem. How good this lower bound is depends on which constraints are convexified. Suppose we want to reformulate the following problem.

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{c}^\intercal\boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{a}_i^\intercal\boldsymbol{x} \geq b_i \quad \forall i \in I, \\ & \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0}, \end{aligned} \tag{2.29}$$

where $\boldsymbol{c} \in \mathbb{R}^k, \boldsymbol{a}_i \in \mathbb{R}^k, b_i \in \mathbb{R}$ for all $i \in I$. Thus, we have $|I|$ constraints. Notice that contrary to the notation before, in this chapter we are looking at the rows $\boldsymbol{a}_i^\intercal, i \in I$ instead of the columns $\boldsymbol{a}_j, j \in J$ as in Problem (2.1).

When we say that we convexify the constraints $I' \subseteq I$, this means that we use the Dantzig-Wolfe convexification with A_2 having precisely $\boldsymbol{a}_i$ with $i \in I'$ as rows, $\boldsymbol{b_2}$ has b_i for $i \in I'$ as the corresponding entries and A_1 having precisely $\boldsymbol{a}_i$ with $i \in I \setminus I'$ as rows and $\boldsymbol{b_1}$ has b_i for $i \in I \setminus I'$ as the corresponding entries.

When we choose to convexify all constraints in the Dantzig-Wolfe reformulation, i.e. $I' = I$, the linear relaxation of the Dantzig-Wolfe reformulation is the same as optimizing over the integer hull $P_{IP} \coloneqq conv\{\boldsymbol{x} \in \mathbb{Z}^k : \boldsymbol{a}_i^\intercal \boldsymbol{x} \geq b_i \, \forall i \in I\}$, yielding the strongest possible reformulation. When we choose to convexify no constraints in the Dantzig-Wolfe reformulation, i.e. $I' = \emptyset$, the linear relaxation of the Dantzig-Wolfe relaxation coincides with the linear relaxation of the original problem, where we optimize over the polyhedron $P_{LP} \coloneqq \{\boldsymbol{x} \in \mathbb{R}^k : \boldsymbol{a}_i^\intercal \boldsymbol{x} \geq b_i \, \forall i \in I\}$ hence leading to the weakest possible reformulation.

When we convexify a subset of the constraints I', we could end up with any bound in between for the linear relaxation of the Dantzig-Wolfe decomposition, as we are then optimizing over $P_{I'} \coloneqq \{\boldsymbol{x} \in \mathbb{R}^k : \boldsymbol{a}_i^\intercal \boldsymbol{x} \geq b_i \, \forall i \in I \setminus I', \boldsymbol{x} \in conv\{\boldsymbol{x'} \in \mathbb{Z}^k : \boldsymbol{a}_i^\intercal \boldsymbol{x'} \geq b_i \, \forall i \in I'\}\}$. This is studied more precisely in [LW18] and [BLW18]. We are now going to discuss some of their findings.

A first insight is that the lower bound gained from the linear relaxation of the Dantzig-Wolfe decomposition can be better than the bound from just the linear relaxation only if

$$conv\{\boldsymbol{x} \in \mathbb{Z}^k : \boldsymbol{a}_i^\intercal \boldsymbol{x} \geq b_i \, \forall i \in I'\} \subsetneq \{\boldsymbol{x} \in \mathbb{R}^k : \boldsymbol{a}_i^\intercal \boldsymbol{x} \geq b_i \, \forall i \in I'\}, \tag{2.30}$$

i.e. $\{\boldsymbol{x} \in \mathbb{R}^k : \boldsymbol{a}_i^\intercal \boldsymbol{x} \geq b_i \forall i \in I'\}$ is not an integral polyhedron. This was proven by Geoffrion [Geo74] for Lagrangian relaxations, and holds thus also for the Dantzig-Wolfe convexification or discretization described in Section 2.2.5.

In [LW18] it is characterized when the Dantzig-Wolfe formulation is as weak or as strong as possible for specific problems, starting with the weighted stable set problem.

Definition 2.2.5 (Weighted Stable Set Problem)**.** *Let $G = (V, E)$ be an undirected graph with vertex set V and edge set $E \subseteq \{\{u, v\} : u, v \in V, u \neq v\}$ and let $w : V \to \mathbb{Z}_{\geq 0}$ be a weight function. Then a stable set in G is a set of vertices $S \subseteq V$ such that no two vertices in S are connected by an edge in G, i.e. for all $u, v \in S$ we have that $\{u, v\}$ is not in E. The task of the weighted stable set problem is to find a stable set with maximal sum of the weights, i.e. find $\operatorname{argmax}\{\sum_{v \in S} w(v) : S$ is a stable set$\}$.*

The weighted stable set problem can also be formulated as a linear program with binary variables using one constraint per edge in the following way. This is called the edge formulation of this problem.

$$\begin{aligned} \max_{\boldsymbol{x}} \quad & \sum_{v \in V} w(v) x_v \\ \text{s.t.} \quad & x_v + x_u \leq 1 \quad \forall \{u, v\} \in E, \\ & x_v \in \{0, 1\}, \quad \forall v \in V. \end{aligned} \tag{2.31}$$

For this formulation it can be shown, that the two polyhedra P_{IP} and P_{LP} coincide if and only if G is a bipartite graph, see [LW18]. This also means, that if G is a bipartite graph, every Dantzig-Wolfe reformulation gives the same lower bound.

Regarding the weakest Dantzig-Wolfe reformulation for Problem (2.31), Geoffrion's necessary condition (2.30) is already sufficient in the case of the edge formulation of the weighted stable set problem.

Theorem 2.2.6. *Let $G = (V, E)$ be a graph, $E' \subseteq E$ a subset of the edges of G and consider the edge formulation (2.31) of its stable set problem. Then the polyhedron $P_{LP}(G) = \{\boldsymbol{x} \in [0,1]^{|V|} : x_u + x_v \leq 1 \, \forall \{u, v\} \in E\}$ of the linear relaxation of Problem (2.31) is the same as the polyhedron $P_{E'}(G) = \{\boldsymbol{x} \in [0,1]^{|V|} : x_u + x_v \leq 1 \, \forall \{u, v\} \in E \setminus E', \boldsymbol{x} \in conv\{\boldsymbol{x'} \in \{0,1\}^{|V|} : x'_u + x'_v \leq 1 \, \forall \{u, v\} \in E'\}\}$ coming from the Dantzig-Wolfe reformulation when convexifying the constraints corresponding to the edges in E' if and only if the graph (V, E') is bipartite.*

Proof. If (V, E') is bipartite, then $P_{IP}((V, E'))$ and $P_{LP}((V, E'))$ are the same for the edge formulation of the problem. This means that $conv\{\boldsymbol{x'} \in \{0,1\}^{|V|} : \boldsymbol{x'}_u + \boldsymbol{x'}_v \leq 1 \, \forall \{u, v\} \in E'\} = \{\boldsymbol{x'} \in [0,1]^{|V|} : x'_u + x'_v \leq 1 \, \forall \{u, v\} \in E'\}$ and hence $P_{LP}(G) = P_{E'}(G)$.

On the other hand, if (V, E') is not bipartite, there must exist an odd cycle C in (V, E'). Consider the vector $\boldsymbol{x}^C$ for which $x^C_v = 1/2$ if v is a vertex in the cycle C, and $x^C_v = 0$ if v is not in C. Then clearly $x^C_u + x^C_v \leq 1$ for all vertices $u, v \in V$ and thus also for all edges $\{u, v\} \in E$. Hence, $\boldsymbol{x}^C \in P_{LP}(G)$. However, for all $\boldsymbol{x} \in conv\{\boldsymbol{x'} \in \{0,1\}^{|V|} : x'_u + x'_v \leq 1 \, \forall \{u, v\} \in E'\}$ it must hold that $\sum_{v \in C} x_v \leq (|C| - 1)/2$, i.e. the odd cycle constraint for C must hold, but $\sum_{v \in C} \boldsymbol{x}^C_v = |C|/2 > (|C| - 1)/2$. Hence $\boldsymbol{x}^C \notin conv\{\boldsymbol{x'} \in \{0,1\}^{|V|} : x'_u + x'_v \leq 1 \, \forall \{u, v\} \in E'\}$, $\boldsymbol{x}^C \notin P_{E'}(G)$ and thus, $P_{E'}(G) \neq P_{LP}(G)$. □

Notice, that for different problems, Geoffrion's necessary condition (2.30) is not always sufficient.

Example. *Consider minimizing $x \in \mathbb{R}$ under the constraints $x \geq 0.5$ and $x \geq 1.5$, i.e. $a_1 = a_2 = 1, b_1 = 0.5$, and $b_2 = 1.5$, and choosing $I' = \{1\}$, then $conv\{\boldsymbol{x} \in \mathbb{Z} : x \geq 0.5\} = \{\boldsymbol{x} \in \mathbb{R} : x \geq 1\} \subsetneq \{\boldsymbol{x} \in \mathbb{R} : x \geq 0.5\}$, but the lower bound gained from the linear relaxation of the Dantzig-Wolfe approximation is $\min\{x : x \geq 1.5, x \in conv\{x' \in \mathbb{Z} : x' \geq 0.5\}\} = min\{x : x \geq 1.5\}$ which is just the bound from the linear relaxation, as the second constraint is more powerful than the first one.*

Regarding the strongest possible Dantzig-Wolfe reformulation for the weighted stable set problem, the following holds.

Theorem 2.2.7. *Let $G = (V, E)$ be a graph, $E' \subseteq E$ a subset of the edges of G and consider the edge formulation (2.31) of its stable set problem. Then the polyhedron $P_{IP}(G) = conv\{\boldsymbol{x} \in \{0,1\}^{|V|} : x_u + x_v \leq 1 \; \forall \{u,v\} \in E\}$ of the integer hull of Problem (2.31) is the same as the polyhedron $P_{E'}(G) = \{\boldsymbol{x} \in [0,1]^{|V|} : x_u + x_v \leq 1 \; \forall \{u,v\} \in E \setminus E', \boldsymbol{x} \in conv\{\boldsymbol{x'} \in \{0,1\}^{|V|} : x'_u + x'_v \leq 1 \; \forall \{u,v\} \in E'\}$ coming from the Dantzig-Wolfe reformulation when convexifying the constraints corresponding to the edges in E' if and only if the graph (V, E') contains all odd induced cycles of G.*

Here, an odd induced cycle $C = (V_C, E_C)$ of G is a cycle with an odd number of vertices $|V_C| \geq 3$, such that there are no edges in $E \setminus E_C$ connecting two different vertices in V_C.

These findings were also expanded to similar problems in [LW18], namely the weighted clique problem and the weighted node covering problem.

In [LW18] the strongest and weakest possible Dantzig-Wolfe reformulation is also characterized for the maximum weight matching problem, as an example on how this can be done for problems where one has a full description of the integer hull.

Definition 2.2.8 (Maximum Weight Matching Problem)**.** *Let $G = (V, E)$ be an undirected graph with vertex set V and edge set $E \subseteq \{\{u,v\} : u, v \in V, u \neq v\}$ and $c : E \to \mathbb{Z}_{\geq 0}$ a weight function for the edges. Then a matching in G is a set of edges $M \subseteq E$ such that no two edges in M are adjacent in G. The task of the maximum weight matching problem is to find a matching with maximum sum of the weights, i.e. find $\operatorname{argmax}\{\sum_{e \in M} c(e) :$ M is a matching$\}$.*

The maximum weight matching problem can also be formulated as the following a linear program with binary variables.

$$\begin{aligned} \max_{\boldsymbol{x}} \quad & \sum_{e \in E} c(e) x_e \\ \text{s.t.} \quad & \sum_{e \,:\, v \in e} x_e \leq 1 \quad \forall v \in V, \\ & x_e \in \{0,1\} \quad \forall e \in E \end{aligned} \tag{2.32}$$

Thus, we have $P_{LP} = \{\boldsymbol{x} \in [0,1]^{|E|} : \sum_{e \,:\, v \in e} x_e \leq 1 \; \forall v \in V\}$ and $P_{IP} = conv\{\boldsymbol{x} \in \{0,1\}^{|E|} : \sum_{e \,:\, v \in e} x_e \leq 1 \; \forall v \in V\}$ for the maximum weight matching problem.

For further study of these polytopes let us first define the following two notions as in [KV12].

Definition 2.2.9 (2-connected)**.** *Let $G = (V, E)$ be an undirected graph. Then G is called 2-connected if $|V| > 2$ and for all $v \in V$ the graph $(V \setminus \{v\}, E \cap \{\{u,w\} : u, w \in V \setminus \{v\}\})$, which arises from G by deleting the vertex v, is connected.*

Definition 2.2.10 (factor-critical)**.** *Let $G = (V, E)$ be an undirected graph. Then G is called factor-critical if for every $v \in V$ the graph $(V \setminus \{v\}, E \cap \{\{u,w\} : u, w \in V \setminus \{v\}\})$, which arises from G by deleting the vertex v, contains a perfect matching.*

From theory about matchings we know, that $P_{LP} = P_{IP}$ if and only if G is a bipartite graph and we have the following complete description of the integer hull

$$P_{IP} = P_{IP}(V) = \{\boldsymbol{x} \in [0,1]^{|E|} : \sum_{e\,:\,v \in e} x_e \leq 1 \; \forall v \in V,$$

$$\sum_{e \in E(S)} x_e \leq \frac{|S|-1}{2} \; \forall S \subseteq V \text{ with } |S| \text{ odd and } G|_S \text{ factor-critical and 2-connected}\},$$

where $G|_S = (S, E(S))$ is the subgraph of G induced by S.

Then, the weakest Dantzig-Wolfe formulation appears precisely in the following case, as proven in [BADF04]

Theorem 2.2.11. *Let $G = (V, E)$ be an undirected graph and $V' \subseteq V$. Then the polyhedron $P_{LP} = \{\boldsymbol{x} \in [0,1]^{|E|} : \sum_{e\,:\,v \in e} x_e \leq 1 \; \forall v \in V\}$ of the linear relaxation of the maximum weight matching problem of G is the same as the polyhedron $P_{V'} = \{\boldsymbol{x} \in [0,1]^{|E|} : \boldsymbol{x} \in P_{IP}(V'), \sum_{e\,:\,v \in e} x_e \leq 1 \; \forall v \in V \setminus V'\}$ coming from the Dantzig-Wolfe reformulation when convexifying the constraints corresponding to the vertices in V' if and only if the subgraph of G induced by V' is bipartite.*

The strongest Dantzig-Wolfe formulation can be characterized as follows.

Theorem 2.2.12. *Let $G = (V, E)$ be an undirected graph and $V' \subseteq V$. Then the polyhedron P_{IP} of the of the maximum weight matching problem of G is the same as the polyhedron $P_{V'} = \{\boldsymbol{x} \in [0,1]^{|E|} : \boldsymbol{x} \in P_{IP}(V'), \sum_{e\,:\,v \in e} x_e \leq 1 \; \forall v \in V \setminus V'\}$ coming from the Dantzig-Wolfe reformulation when convexifying the constraints corresponding to the vertices in V' if and only if the subgraph of G induced by V' contains every 2-connected, factor-critical subgraph of G.*

In [BLW18] the focus is not only on when the reformulation is weakest or strongest, but also what happens in between. This is investigated computationally for some specific integer problems using a brute-force approach. The authors set up very small instances of integer linear programs coming from classical combinatorial optimization problems, such as bin packing or vehicle routing and integer linear programming instances without such an underlying structure. Then, the authors consider all possible choices of the restrictions to be convexified in the Dantzig-Wolfe reformulation and compute the corresponding bounds. These experiments are repeated for multiple randomly chosen objective functions, as the size of the gap between the dual bounds also depends on the objective function. The results show that the number of different dual bounds that occur is much smaller than the number of possibilities of choosing rows to be used for the Dantzig-Wolfe reformulation. Further, the authors notice that the number of different dual bounds is often very close to a power of 2. The authors also check the frequencies with which certain bounds occur and how often good dual bounds are achieved. Their findings imply that strong dual bounds are not achieved often, but also not many reformulations result in the weakest possible bounds, so most reformulations were somewhere in between. Also, the authors find that the number of constraints which are convexified is not a good indicator of the quality of the reformulation.

These observations together support the conjecture that some rows have a lot of impact on the dual bound for most Dantzig-Wolfe reformulations, while most rows normally only have a minor impact. Thus, the authors also investigate what differences can be seen in the dual bounds when we first convexify a set of rows I' with $i \notin I'$ and then convexify $I' \cup i$ for different rows i and different sets I'. Most often it can be seen, that for a particular row i the average gain among all sets $I' \subseteq I \setminus \{i\}$ when adding i is a good indicator for how much impact constraint i has. Indeed, it is often the case that if the highest gain of row i was higher than the highest gain of row i', the gain of i is almost always higher than the one of i'. Focusing on the average gains for a constraint, the authors find that this indeed highly depends on the kind of constraint. Especially, for the bin packing problem and the knapsack problem the columns with a lot of impact coincide with the Dantzig-Wolfe reformulations found in literature.

However, it is unclear whether a similar behavior occurs when dealing with larger instances and the question on how to efficiently find a good Dantzig-Wolfe reformulation remains wide open.

2.3 Using Lagrangian Relaxation in Column Generation

At this point, we wish to explore how to incorporate Lagrangian relaxation in column generation in a beneficial way, instead of using only the simplex method to solve the restricted master problem and the pricing problem. We will consider a method shown in [Hui+05].

2.3.1 Lagrangian Relaxation for the Pricing Problem

One method to potentially speed up the column generation procedure after a Dantzig-Wolfe decomposition is to not only use the simplex method but also use Lagrangian relaxation on the original problem to obtain a lower bound on the objective value and to add columns resulting from Lagrangian subproblems to the restricted master problem additionally to the column obtained by simplex method in each iteration. We will describe this method as in [Hui+05], [BJ98] and [DJ07].

Suppose that we want to solve the linear relaxation of the Dantzig-Wolfe convexification (2.12) of the compact formulation (2.10). Hence, we choose Problem (2.12) as our master problem. Then the pricing problem is

$$\min_{s \in P \cup R} \{c_s - \boldsymbol{u}^{*\mathsf{T}} \boldsymbol{a}_s - u_0^*\} = \min\{\boldsymbol{c}^{\mathsf{T}} \boldsymbol{x} - \boldsymbol{u}^{*\mathsf{T}} A_1 \boldsymbol{x} - u_0^* : A_2 \boldsymbol{x} \geq \boldsymbol{b_2}, \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0}\}, \qquad (2.33)$$

where the notation is as in Section 2.2.1.

If we consider the Lagrangian relaxation of the compact formulation of the Problem (2.10), where we relax the constraints $A_1 \boldsymbol{x} \geq \boldsymbol{b_1}$ and use $\boldsymbol{u}^*$ as a multiplier, we get the Lagrangian

subproblem

$$\begin{aligned} &\min\{\boldsymbol{c}^\intercal \boldsymbol{x} - \boldsymbol{u}^{*\intercal}(A_1\boldsymbol{x} - \boldsymbol{b_1}) : A_2\boldsymbol{x} \geq \boldsymbol{b_2}, \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0}\} = \\ &\min\{\boldsymbol{c}^\intercal \boldsymbol{x} - \boldsymbol{u}^{*\intercal} A_1\boldsymbol{x} + \boldsymbol{u}^{*\intercal}\boldsymbol{b_1} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}, \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0}\}. \end{aligned} \tag{2.34}$$

We can see in Equation 2.34 that the pricing problem and the Lagrangian subproblem are minimized by the same $\boldsymbol{x}$, as the pricing problem is $\min\{\boldsymbol{c}^\intercal \boldsymbol{x} - \boldsymbol{u}^{*\intercal} A_1\boldsymbol{x} - u_0^* : A_2\boldsymbol{x} \geq \boldsymbol{b_2}, \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0}\}$ and $\boldsymbol{u}^{*\intercal}\boldsymbol{b_1}$ and u_0^* are constant. Hence, we can also create new columns by solving the Lagrangian subproblem. Also, the objective value of the Lagrangian subproblem gives us a lower bound on the objective value of the linear relaxation of the master problem. To get even better lower bounds, we can perform a couple of subgradient iterations to update the Lagrangian multipliers, where we can use the primal solution of the restricted master problem as an upper bound for the master problem to compute a step size. The minimizers of these new Lagrangian subproblems can also be used as additional columns to be added to the restricted master problem.

All in all, we have a hybrid column generation and Lagrangian relaxation algorithm to solve the linear relaxation of an integer linear program, summarized in Algorithm 3.

Algorithm 3 Column Generation with Lagrangian Relaxation in the Pricing Problem

Input feasible master problem
Output optimal primal and dual solution of the master problem

1: Find small J' such that the restricted master problem is feasible.
2: Solve the restricted master problem, $\boldsymbol{x}^*$ primal solution, $(\boldsymbol{u}^{*\intercal}, u_0^*)^\intercal$ dual solution using a simplex algorithm, where u_0^* corresponds to the convexity constraint.
3: **if** $\min\{\boldsymbol{c}^\intercal \boldsymbol{x} - \boldsymbol{u}^{*\intercal} A_1\boldsymbol{x} - u_0^* : A_2\boldsymbol{x} \geq \boldsymbol{b_2}, \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0}\} < 0$ **then**
4: $\quad J' = J' \cup \{\operatorname{argmin}\{\boldsymbol{c}^\intercal \boldsymbol{x} - \boldsymbol{u}^{*\intercal} A_1\boldsymbol{x} - u_0^* : A_2\boldsymbol{x} \geq \boldsymbol{b_2}, \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0}\}\}$
5: $\quad$ stopping criterion = false
6: $\quad \boldsymbol{\lambda} = \boldsymbol{u}^*$
7: $\quad$ **while** stopping criterion == false **do**
8: $\quad\quad J' = J' \cup \{\operatorname{argmin}\{\boldsymbol{c}^\intercal \boldsymbol{x} - \boldsymbol{\lambda}^\intercal(A_1\boldsymbol{x} - \boldsymbol{b_1}) : A_2\boldsymbol{x} \geq \boldsymbol{b_2}, \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0}\}\}$
9: $\quad\quad$ Update multiplier $\boldsymbol{\lambda}$ by using a subgradient method
10: $\quad\quad$ Update stopping criterion
11: $\quad$ **end while**
12: $\quad$ **go to** 2
13: **else return** $(\boldsymbol{x}^*, \boldsymbol{u}^*)$
14: **end if**

In this algorithm we make sure that we solve $\operatorname{argmin}\{\boldsymbol{c}^\intercal \boldsymbol{x} - \boldsymbol{u}^{*\intercal} A_1\boldsymbol{x} - u_0^* : A_2\boldsymbol{x} \geq \boldsymbol{b_2}, \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0}\}$ in a way that we attain an extreme point or ray as a solution in order to be able to generate a column from it. Thus, in the pricing step of the column generation algorithm, we first add the column with smallest reduced cost. Then we solve a few Lagrangian relaxation

subproblems, where we start with the optimal dual solution of the restricted master problem as a Lagrangian multiplier and improve it by some subgradient method. Thereby each Lagrangian subproblem yields a new column which is also added to the restricted master problem and a lower bound on the optimal objective value of the master problem.

The stopping criterion for the subgradient iterations could be any combination of a maximal number of iterations, the (relative) size of the gap between the lower bound $\min\{\boldsymbol{c}^\intercal \boldsymbol{x} - \boldsymbol{u}^{*\intercal}(A_1\boldsymbol{x} - \boldsymbol{b_1}) : A_2\boldsymbol{x} \geq \boldsymbol{b_2}, \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0}\}$ and the upper bound $\boldsymbol{x}^*$, computation time, etc..

The correctness of this algorithm follows directly from the fact that we only stop the algorithm when there are no more columns with negative reduced cost. The termination of the algorithm is clear, whenever the stopping criterion for the subgradient iterations is chosen such that stopping after a finite number of iterations is guaranteed.

The advantages of this hybrid algorithm come from the fact that producing a new column using the simplex method to resolve the restricted master problem is computationally more expensive than performing a subgradient iteration and solving a Lagrangian subproblem. By adding multiple columns in each pricing step, we potentially reduce the number of iterations. Also, we receive a good lower bound on the optimal objective value of the master problem through solving the Lagrangian subproblem. Compared to using only Lagrangian relaxation and a subgradient method, we receive primal feasible solutions by solving the restricted master problems and have an exact solution at termination.

In [BJ98] this method was used for the plant location problem with minimum inventory and the computations showed a significant improvement of the computing time and the lower bound, compared to the results obtained by using a classical column generation approach without Lagrangian relaxation. In [DJ07] it was reported that for the capacitated lot sizing problem the number of iterations for the hybrid column generation algorithm were lower and more columns were added, while the running time was only about half as long as for the classical column generation algorithm.

2.4 Tailing-Off Effect and Stabilization

Using column generation has proven itself to be more efficient than solving problems directly for many applications where a linear program with many variables occurs. However, when using the classical column generation approach, one can often notice that a good approximate solution is attained quite fast, but it takes much longer to reach the optimal solution, as the convergence is very slow towards the end of the procedure. This is known as the tailing-off effect. A possible cause for this is that the dual bounds in each iteration are not monotonically decreasing, but rather oscillating. Being able to control the dual bounds in the column generation procedure seems to be one of the most promising things to do towards increasing efficiency, see for instance [LD05]. As stated in [BADF04], the tailing-off effect also appears to be influenced by degenerate solutions in the primal problem. Therefore it

is suggested in [Van05] that one should try not to use redundant constraints and equality constraints whenever possible.

In order to avoid the oscillation of the dual bounds, many stabilization techniques have been developed. This typically leads to a lower number of total iterations but might result in more time expensive iterations as the pricing problems often get more difficult. Nevertheless, from many computational studies it is clear that stabilization can lead to significantly lower overall running time, see [Lü11].

In [Van05] stabilization methods are put into four categories: methods which define bounds on the dual prices, penalize the deviation from some stability center, smooth dual prices, or work with interior dual solutions. These methods can also be combined. We now want to look at some stabilization techniques. Note that this is not an exhaustive list of stabilization techniques.

2.4.1 Penalizing Deviation from a Stability Center

As the dual solutions of the restricted master problems are not monotonically increasing during the column generation procedure, the current dual solution might not yield the best bound for the primal optimal solution. Taking the maximum over all previous dual solutions of the restricted master problem would give the best lower bound. After the l-th iteration of the column generation procedure this bound is given by $\hat{\boldsymbol{u}} = \operatorname{argmax}_{k\in\{1,\dots,l\}} L(\boldsymbol{u}^k)$ and is called the stability center, where $\boldsymbol{u}^k$ is the optimal solution of the dual restricted master problem in iteration k. As the oscillation of the lower bounds is considered a major issue regarding the speed of the convergence, one idea is to favor solutions where the new dual solution is closer to the best dual solution found so far, thus penalizing the deviation from the stability center.

We want to discuss such penalization methods as in [Bri+08], [Bri+05] and [BADF04]. Here the stabilization is done via a stabilization function $S : \mathbb{R}^m \to \mathbb{R}_{\geq 0}$ with $\mathbf{0}$ being a minimizer of S. Then, instead of solving the Lagrangian dual of the restricted master problem (2.7) in every iteration to find a new column, we solve the stabilized Lagrangian dual of the restricted master problem

$$\max_{\boldsymbol{u}\geq 0} L^{RMP}(\boldsymbol{u}) - S(\boldsymbol{u} - \hat{\boldsymbol{u}}), \tag{2.35}$$

where $\hat{\boldsymbol{u}}$ is the current stability center. This is supposed to favor solutions close to the stability center. Further, strategies to update the stabilization function during the algorithm can be included, for instance by changing the parameters of the stabilization function.

A generic stabilization method of this type works as in Algorithm 4.

For $S(\boldsymbol{u}) = 0$ for all $\boldsymbol{u}$, and $\min_{j\in J\setminus J'}\{c_j - \boldsymbol{u}^{k\intercal}\boldsymbol{a}_j\} \geq 0$ as a stopping criterion, this coincides with the cutting plane algorithm for the dual of the master problem.

Algorithm 4 Stabilized Column Generation by Penalizing Deviation from a Stability Center

Input feasible master problem
Output approximations of the primal and dual solution of the master problem

1: Start with some first columns J'. And initialize $\hat{\boldsymbol{u}}$ by a heuristic or by $\hat{\boldsymbol{u}} = \mathbf{0}$. Set $k = 1$.
2: Solve the stabilized Lagrangian dual of the restricted master problem (2.35), yielding a minimizer $\boldsymbol{u}^k$.
3: Compute new column $j^{k+1} = \operatorname{argmin}_{j \in J \setminus J'} \{c_j - \boldsymbol{u}^{k\intercal} \boldsymbol{a}_j\}$.
4: **if** stopping criterion not fulfilled **then**
5: $J' = J' \cup \{j^{k+1}\}$
6: **if** $L(\boldsymbol{u}^k) > L(\hat{\boldsymbol{u}})$ **then**
7: Set $\hat{\boldsymbol{u}} = \boldsymbol{u}^k$.
8: **end if**
9: Update S.
10: $k = k + 1$
11: **go to** 2
12: **else** Compute approximate primal solution for the master problem $\hat{\boldsymbol{x}}$, **return** $(\hat{\boldsymbol{x}}, \boldsymbol{u}^k)$
13: **end if**

Note that one could also choose to only update the center of stability, if $L(\boldsymbol{u}) > L(\hat{\boldsymbol{u}}) + \epsilon$ for a suitably chosen ϵ in order to avoid changing the stabilized dual restricted master problem too often.

As for the computation of an approximation $\hat{\boldsymbol{x}}$ for the primal optimal solution of the master problem, we want to consider some kind of dual of Problem (2.35) which is similar to the primal master problem. For this we reformulate the stabilized Lagrangian dual of the restricted master problem (2.35) as follows.

$$\begin{aligned} \max_{\boldsymbol{u} \geq 0} \quad & L^{RMP}(\boldsymbol{u}) - S(\boldsymbol{v}) \\ \text{s.t.} \quad & \boldsymbol{v} = \boldsymbol{u} - \hat{\boldsymbol{u}}, \end{aligned} \tag{2.36}$$

The corresponding Lagrangian dual is then

$$\min_{\boldsymbol{g}} \max_{\boldsymbol{u} \geq 0} \{L^{RMP}(\boldsymbol{u}) - S(\boldsymbol{v}) + \boldsymbol{g}^\intercal(\boldsymbol{v} - \boldsymbol{u} + \hat{\boldsymbol{u}})\}. \tag{2.37}$$

After some reformulation this turns out to be the same as the so called Fenchel dual.

$$\begin{aligned} \min_{(\boldsymbol{x}, \boldsymbol{g})} \quad & \tilde{\boldsymbol{c}}^\intercal \boldsymbol{x} + \hat{\boldsymbol{u}}^\intercal \boldsymbol{g} + S^*(\boldsymbol{g}) \\ \text{s.t.} \quad & \tilde{A}\boldsymbol{x} \geq \boldsymbol{b} - \boldsymbol{g}, \end{aligned} \tag{2.38}$$

where $S^*(\boldsymbol{g}) \coloneqq \max_{\boldsymbol{v}}(\boldsymbol{g}^\intercal \boldsymbol{v} - S(\boldsymbol{v}))$ and $\tilde{\boldsymbol{c}}$ and $\tilde{A}$ consist of the entries and columns of $\boldsymbol{c}$ and A corresponding to J'.

Then the following theorem is proven in [Bri+08] and [Bri+05].

Theorem 2.4.1. *Let S be a convex function and $\boldsymbol{u}^k$ an optimal solution of the stabilized dual restricted master problem (2.35) such that S only takes finite values in a neighborhood of $\boldsymbol{u}-\hat{\boldsymbol{u}}$. Then there exist a minimizer $\hat{\boldsymbol{x}}$ of $L^{RMP}(\boldsymbol{u}^k)$, and a subgradient $\hat{\boldsymbol{g}}$ of S at $\boldsymbol{u}-\hat{\boldsymbol{u}}$, such that complementary slackness w.r.t. the Fenchel dual is satisfied, i.e. $\tilde{A}\hat{\boldsymbol{x}}-\boldsymbol{b}+\hat{\boldsymbol{g}}\geq\boldsymbol{0}$, $\boldsymbol{u}^k\geq\boldsymbol{0}$ and $(\tilde{A}\hat{\boldsymbol{x}}-\boldsymbol{b}+\hat{\boldsymbol{g}})^\intercal\boldsymbol{u}^k=\boldsymbol{0}$. Then $(\hat{\boldsymbol{x}},\hat{\boldsymbol{g}})$ are the optimal solutions for the Fenchel dual (2.38). Furthermore, $\hat{\boldsymbol{x}}$ is an optimal solution of the restricted master problem if $\hat{\boldsymbol{g}}=\boldsymbol{0}$ and an optimal solution of the master problem if $\boldsymbol{c}^\intercal\hat{\boldsymbol{x}}=L(\hat{\boldsymbol{u}})$*

To compute $\hat{\boldsymbol{x}}$ one can for instance solve the corresponding Fenchel dual.

As for the stopping criterion, this theorem tells us that we want $\hat{\boldsymbol{g}}\leq\boldsymbol{0}$ in order to have $\hat{\boldsymbol{x}}$ feasible for the restricted master problem and we want $\boldsymbol{c}^\intercal\hat{\boldsymbol{x}}=L(\hat{\boldsymbol{u}})$ for $\hat{\boldsymbol{x}}$ to be an optimal primal solution of the master problem. When applying these stopping criteria in practice, we allow some small deviation.

The question under which conditions this procedure converges remains open. There are some convergence results for certain specific stabilization functions. For example, in [Fra02] the convergence is proven if certain conditions on the function to be minimized and the stabilizing function are fulfilled. In [BADF04] some less general conditions are given, which suffice for the convergence of the procedure to optimal primal and dual solutions of the master problem. Among other possibilities, one of the following conditions suffices.

- S is differentiable at 0
- $S(0)=0$, S is convex, non-negative and its level sets are convex and full dimensional
- S is strictly convex at 0
- S is "steep enough" to ensure that the optimal solution of Problem (2.35) is always finite

Algorithm 4 can be used for various stability functions. We now want to look at some common stabilization functions.

Example (Boxstep method)**.** *For instance we can force the next dual solution to have a distance of at most ϵ from the stability center $\hat{\boldsymbol{u}}$ by choosing S to be the indicator function $S(\boldsymbol{v})=0$, if $|\boldsymbol{v}_i|\leq\epsilon$ for every component i of $\boldsymbol{v}$ and $S(\boldsymbol{v})=\infty$ otherwise. Thus this is actually a method where we put bounds on dual prices. Of course one needs to carefully choose ϵ, as a too small epsilon keeps us from finding a good next solution, while a too large ϵ does not stabilize much.*

The stabilized restricted master problem is then

$$\max\{L^{RMP}(\boldsymbol{u}):||\boldsymbol{u}-\hat{\boldsymbol{u}}||_\infty\leq\epsilon,\boldsymbol{u}\geq 0\}.$$

And we have $S^*(\boldsymbol{g})=\epsilon\,||\boldsymbol{g}||_1$, which gives the Fenchel dual.

$$\begin{array}{ll}\min\limits_{(\boldsymbol{x},\boldsymbol{g})} & \tilde{\boldsymbol{c}}^\intercal\boldsymbol{x}+\hat{\boldsymbol{u}}^\intercal\boldsymbol{g}+\epsilon\,||\boldsymbol{g}||_1\\ \text{s.t.} & \tilde{A}\boldsymbol{x}\geq\boldsymbol{b}-\boldsymbol{g}\end{array}\tag{2.39}$$

Other possible choices for S include piecewise linear functions which are close to a quadratic stabilization function S, or a quadratic stabilization function.

Example (Bundle method)**.** *When taking the Euclidean distance to the stability center as penalization, this stabilization method is called bundle method. We take* $S(\boldsymbol{v}) = \frac{1}{2t}\,||\boldsymbol{v}||^2$*, where* t *is a constant which can be chosen such that the trade-off between staying close to the stability center and finding an actual maximum of the dual restricted master problem is dealt with well and can be changed in each iteration.*

The stabilized restricted master problem is then

$$\max_{\boldsymbol{u}\geq 0}\{L^{RMP}(\boldsymbol{u}) + \frac{1}{2t}\,||\boldsymbol{u}-\hat{\boldsymbol{u}}||^2\}.$$

And we have $S^*(\boldsymbol{g}) = \frac{t}{2}\,||\boldsymbol{g}||^2$*, which gives the Fenchel dual*

$$\begin{aligned} \min_{(\boldsymbol{x},\boldsymbol{g})} \quad & \tilde{\boldsymbol{c}}^\intercal\boldsymbol{x} + \hat{\boldsymbol{u}}^\intercal\boldsymbol{g} + \frac{t}{2}\,||\boldsymbol{g}||^2 \\ \text{s.t.} \quad & \tilde{A}\boldsymbol{x} \geq \boldsymbol{b} - \boldsymbol{g}, \end{aligned} \tag{2.40}$$

In [BADF04] computational experiments on the multiple depot vehicle routing problem and the simultaneous vehicle and crew scheduling problem show that the bundle method performs better than classical column generation. In [Bri+08] and [Bri+05] the bundle method is computationally compared to classical column generation, in this case performed as a cutting plane method on the dual, on five different problems, namely the cutting stock problem, the vertex coloring problem, the capacitated vehicle routing problem, the multi-item lot sizing problem and the traveling salesman problem. The authors report that the number of iterations is often similar, but when the number of rows also gets large the classical column generation procedure sometimes takes far more iterations than the bundle method. Furthermore, the authors also report that the time needed to solve the pricing problem is indeed sometimes longer for the bundle method but suggest that a warm start might help. The authors state that a main downside of the bundle method is the need to exactly solve the pricing problem in every step. All in all, it is not always clear which method is better. However, if there are developments in quadratic optimization, this could improve the running time of the bundle method significantly.

2.4.2 Smoothing Dual Prices

A disadvantage of penalizing the deviation from the stability center in the way described above is that the pricing problem can get more difficult. Further, we need to choose several parameters, e.g. parameters for the stabilization function and for the update of the stability center. Also we need to decide when we want to change these parameters during the procedure. To overcome these downsides we now want to discover a stabilization method described in [Pes+08] and [Pes+10] which still solves the pricing problem with dual variables

close to a stability center.

Instead of penalizing dual solutions far from the stability center by adding a stabilization function to the dual restricted master problem, the ordinary restricted master problem is solved, but the pricing problem in iteration k is performed with a convex combination $\boldsymbol{u}_S^k$ of the optimal dual solution of the restricted master problem $\boldsymbol{u}^k$ and a center of stability $\hat{\boldsymbol{u}}$, i.e. $\boldsymbol{u}_S^k = \alpha \boldsymbol{u}^k + (1-\alpha)\hat{\boldsymbol{u}}$ for some $\alpha \in (0,1)$. Here we use $\hat{\boldsymbol{u}} = \operatorname{argmax}_{k\in\{1,\dots,l\}} L(\boldsymbol{u}_S^k)$, i.e. our center of stability is the best Lagrangian lower bound obtained from the convex combinations so far.

Algorithm 5 Stabilized Column Generation by Smoothing Dual Prices

Input feasible master problem, $\epsilon \in \mathbb{R}_+$
Output primal and dual solution of the master problem

1: Start with some first columns J' such that the restricted master problem is feasible. And initialize $\hat{\boldsymbol{u}}$ by a heuristic or by $\hat{\boldsymbol{u}} = \boldsymbol{0}$. Choose ϵ small enough. Set $k = 1$.
2: Solve the restricted master problem (2.2), yielding a primal optimal solution $\boldsymbol{x}^k$ with optimal objective value Z^k and a dual optimal solution $\boldsymbol{u}^k$.
3: **if** $Z^k - L(\hat{\boldsymbol{u}}) > \epsilon$ **then**
4: $\quad \boldsymbol{u}_S^k = \alpha \boldsymbol{u}^k + (1-\alpha)\hat{\boldsymbol{u}}$
5: $\quad$ Compute $j^{k+1} = \operatorname{argmin}_{j\in J\setminus J'}\{c_j - \boldsymbol{u}_S^{k\intercal}\boldsymbol{a}_j\}$.
6: $\quad$ **if** $\{c_{j^{k+1}} - \boldsymbol{u}^{k\intercal}\boldsymbol{a}_{j^{k+1}}\} < 0$ **then**
7: $\quad\quad J' = J' \cup \{j^{k+1}\}$
8: $\quad$ **end if**
9: $\quad$ **if** $L(\boldsymbol{u}_S^k) > L(\hat{\boldsymbol{u}})$ **then**
10: $\quad\quad$ Set $\hat{\boldsymbol{u}} = \boldsymbol{u}_S^k$.
11: $\quad$ **end if**
12: $\quad k = k + 1$
13: $\quad$ **go to** 2
14: **end if**
15: **return** $(\boldsymbol{x}^k, \boldsymbol{u}^k)$

Theorem 2.4.2. *Algorithm 5 terminates after a finite number of iterations with an optimal primal and dual solution for the master problem, provided that we choose ϵ small enough.*

Proof. As the total number of columns of the master problem $|J|$ is finite, we can only have a finite number of iterations where we add a new column to the restricted master problem. We will show that if no new column is added to the restricted master problem, the gap $Z^k - L(\hat{\boldsymbol{u}}) < \infty$ will get smaller by at least a factor of $(1-\alpha)^{-1}$. As the algorithm terminates when this gap becomes less than ϵ, this means that the algorithm terminates after a finite number of iterations.

Suppose that no new column is added in iteration k, i.e. $\{c_{j^{k+1}} - \boldsymbol{u}^k \boldsymbol{a}_{j^{k+1}}\} \geq 0$. Denote by $L^k(\boldsymbol{u})$ the Lagrangian subproblem of the restricted master problem at iteration k and

by $L_+^k(\boldsymbol{u})$ the Lagrangian subproblem of the restricted master problem at iteration k where we additionally dualize the constraint corresponding to j^{k+1}. Assume to the contrary, that $L(\boldsymbol{u}_S^k) < L(\hat{\boldsymbol{u}}) + \alpha(Z^k - L(\hat{\boldsymbol{u}}))$. Observe that by the fact that the Lagrangian function is concave and the definition of $\boldsymbol{u}_S^k$, it holds that

$$\alpha L_+^k(\boldsymbol{u}^k) + (1-\alpha)L_+^k(\hat{\boldsymbol{u}}) \leq L_+^k(\alpha\boldsymbol{u}^k + (1-\alpha)\hat{\boldsymbol{u}}) = L_+^k(\boldsymbol{u}_S^k).$$

Furthermore, we have

$$L_+^k(\boldsymbol{u}_S^k) = L(\boldsymbol{u}_S^k) < L(\hat{\boldsymbol{u}}) + \alpha(Z^k - L(\hat{\boldsymbol{u}})) = \alpha Z^k + (1-\alpha)L(\hat{\boldsymbol{u}})$$

due to our assumption. Observe that $Z^k = L^k(\boldsymbol{u}^k)$ due to strong duality. Moreover, $L(\hat{\boldsymbol{u}}) \leq L_+^k(\hat{\boldsymbol{u}})$, as we have less restrictions in the restricted master problem than in the master problem. Thus, it follows that

$$\alpha Z^k + (1-\alpha)L(\hat{\boldsymbol{u}}) \leq \alpha L^k(\boldsymbol{u}^k) + (1-\alpha)L_+^k(\hat{\boldsymbol{u}}).$$

All in all, this yields

$$\alpha L_+^k(\boldsymbol{u}^k) + (1-\alpha)L_+^k(\hat{\boldsymbol{u}}) < \alpha L^k(\boldsymbol{u}^k) + (1-\alpha)L_+^k(\hat{\boldsymbol{u}}),$$

which implies

$$L_+^k(\boldsymbol{u}^k) < L^k(\boldsymbol{u}^k).$$

However, this is not possible, as $(c_{j^{k+1}} - \boldsymbol{u}^{k\intercal}\boldsymbol{a}_{j^{k+1}}) \geq 0$. Thus our assumption must have been wrong and it holds that $L(\boldsymbol{u}_S^k) \geq L(\hat{\boldsymbol{u}}) + \alpha(Z^k - L(\hat{\boldsymbol{u}}))$. In particular, this means that $L(\boldsymbol{u}_S^k) > L(\hat{\boldsymbol{u}})$ and therefore our center of stability will be updated to $\boldsymbol{u}_S^k$. The inequality also implies that $Z^k - L(\boldsymbol{u}_S^k) \leq Z^k - (L(\hat{\boldsymbol{u}}) + \alpha(Z^k - L(\hat{\boldsymbol{u}}))) = (Z^k - L(\hat{\boldsymbol{u}}))(1-\alpha)$, which means that the gap between the optimal solution of the restricted master problem and the Lagrangian function at the stability center is indeed getting smaller by a factor of at least $(1-\alpha)^{-1}$.

It remains to prove that it is possible to choose ϵ small enough so as to get optimal primal and dual solutions of the master problem at termination.

Let $\boldsymbol{x}^*$ be an optimal solution of the master problem. Notice that there are only finitely many basic feasible solutions. Therefore, there is a minimal difference between the objective value of the master problem corresponding to $\boldsymbol{x}^*$ and the objective value corresponding to any non-optimal basic feasible solution of the master problem, i.e. $\min\{||\boldsymbol{c}^\intercal\boldsymbol{x}^* - \boldsymbol{c}^\intercal\boldsymbol{x}|| : \boldsymbol{x} \text{ is a basic solution of the master problem}, \boldsymbol{c}^\intercal\boldsymbol{x} \neq \boldsymbol{c}^\intercal\boldsymbol{x}^*\} > 0$. Here $||.||$ is the Euclidean norm and $\boldsymbol{c} = (c_1, ..., c_n)$ comes from the objective function of the master problem. Hence we can choose ϵ such that

$$0 < \epsilon < \min\{||\boldsymbol{c}^\intercal\boldsymbol{x}^* - \boldsymbol{c}^\intercal\boldsymbol{x}|| : \boldsymbol{x} \text{ is a basic solution of the master problem}, \boldsymbol{c}^\intercal\boldsymbol{x} \neq \boldsymbol{c}^\intercal\boldsymbol{x}^*\}.$$

Then, at termination Z^k must be the optimal objective value of the master problem, as $Z^k \geq Z^* \geq L(\hat{\boldsymbol{u}})$ and $Z^k - L(\hat{\boldsymbol{u}}) \leq \epsilon$ at termination, but any non-optimal basic feasible solution would have $Z^k - Z^* > \epsilon$. Hence, also $\boldsymbol{x}^k$ and $\boldsymbol{u}^k$ must be the optimal primal and dual solution to the master problem. □

In [Pes+08] and [Pes+10] this method is computationally compared to the classical column generation approach for random instances of single machine scheduling problems of different sizes. The results show that the number of iterations and the overall running time of the stabilized procedure is greatly reduced by an average factor of about 6 for instances with 40 jobs and an average factor of about 11 for instances with 50 jobs.

2.4.3 Interior Point Stabilization

Another possibility of stabilization is to use interior dual solutions of the restricted master problem instead of extreme dual solutions.

One such method is the analytic center cutting plane method (ACCPM), as in [GV02] and [BVS07]. In the ACCPM we search for an optimal solution of the dual master problem in a certain polytope, which we call the localization $\mathcal{L}$, by iteratively computing the analytic center of the polytope and finding a cutting plane for this analytic center. As we use some kind of center of the polytope to cut off the polytope in every iteration, the hope is that we cut off a rather large part of the polytope in every iteration.

In the context of column generation, the initial polytope consists of some of the constraints of the master problem and a lower bound lb on the solution of the dual master problem. Furthermore, we search for a cutting plane among the constraints of the dual master problem and if no such constraint is violated, we know that the current analytic center is feasible for the dual master problem and have thus a new lower bound on the solution, which gives again a cutting plane for the localization polytope.

The analytic center of a polytope given by the inequalities $\boldsymbol{a}_i^\intercal \boldsymbol{x} \leq b_i$ for $i \in \{1, ..., m\}$ is defined as the minimizer $\operatorname{argmin}_{\boldsymbol{x}}\{-\sum_{i=1}^{m} \log(b_i - \boldsymbol{a}_i^\intercal \boldsymbol{x})\}$. Notice, that the domain of the function $-\sum_{i=1}^{m} \log(b_i - \boldsymbol{a}_i^\intercal \boldsymbol{x})$ is precisely the set $\{\boldsymbol{x} : \boldsymbol{a}_i^\intercal \boldsymbol{x} \leq b_i, \forall i \in \{1, ..., m\}\}$. When an inequality $\boldsymbol{a}_{i0}^\intercal \boldsymbol{x} \leq b_{i0}$ is added to the polytope, the function in the argmin above can be updated the by adding $-\log(b_{i0} - \boldsymbol{a}_{i0}^\intercal \boldsymbol{x})$. For the localization $\mathcal{L} = \{\boldsymbol{u} : \boldsymbol{a}_j^\intercal \boldsymbol{u} \leq c_j, \forall j \in J', \boldsymbol{b}^\intercal \boldsymbol{u} \geq lb\}$, where the notation is as in Problem (2.1), this means that the analytic center is given by $\operatorname{argmin}_{\boldsymbol{u}}\{-\sum_{i=1}^{|J'|} \log(c_i - \boldsymbol{a}_j^\intercal \boldsymbol{u}) - \log(\boldsymbol{b}^\intercal \boldsymbol{u} - lb)\}$.

The analytic center cutting plane algorithm can then be performed as in Algorithm 6.

As for the computation of the (approximate) stability center of the localization, one possibility would be to first find a point in the domain $\{\boldsymbol{x} : \boldsymbol{a}_i^\intercal \boldsymbol{x} \leq b_i, \forall i \in \{1, ..., n\}\}$ of the function and then use Newton method to minimize $-\sum_{i=1}^{n} \log(b_i - \boldsymbol{a}_i^\intercal \boldsymbol{x})$, as this function is convex, and get an approximate analytic center.

In [GV02] it is shown that ACCPM converges even for approximate solutions of the stability center. Furthermore, the authors state that the advantages of ACCPM rely on its applicability to a large variety of problems. Moreover, the performance is not conditioned by a very careful choice of parameters and it does not suffer when dealing with degenerate cases. However, the computation of the stability center in every iteration is time expensive

and ACCPM takes rather long to converge to an optimal solution.

We have presented here some of the main concepts of stabilization. However, there are many more stabilization techniques and combinations of them which have also shown to be useful computationally. It would be highly desirable to have a comparison between the various stabilization methods indicating which methods perform better under which conditions.

Algorithm 6 Analytic Center Cutting Plane Method for Column Generation

Input feasible dual master problem
Output approximation of optimal solution of the dual master problem

1: Find a lower bound lb of the objective value of the dual master problem.
2: Choose small J' such that the constraints form a polytope together with the lower bound.
3: Build localization $\mathcal{L}$. Compute upper bound ub of the objective value of the dual master problem.
4: (Approximately) compute the stability center $\boldsymbol{u}_{ST}$ of the localization $\mathcal{L}$.
5: **if** there exists a constraint in $J \setminus J'$ which is violated **then**
6: add one or more violated constraints to J'.
7: Update localization $\mathcal{L}$.
8: Update upper bound ub.
9: **if** $ub - lb < \epsilon$ **then go to** 18
10: **end if**
11: **go to** 4
12: **else**
13: **if** $\boldsymbol{b}^\mathsf{T}\boldsymbol{u}_{ST} > lb$ **then**
14: update lower bound lb and localization $\mathcal{L}$.
15: **if** $ub - lb < \epsilon$ **then go to** 18
16: **end if**
17: **go to** 4
18: **end if**
19: **end if**
20: **return** $\boldsymbol{u}_{ST}$

3 Branch-and-Price

3.1 Introduction to Branch-and-Price

So far, we have seen how to solve linear programs by using column generation. However, a great part of the usefulness of column generation comes from applying it in the process of solving integer linear programs. To be more precise, we can use column generation to solve the linear relaxation of an integer linear program in every node of the branch-and-bound tree of a branch and bound procedure. This approach is called branch-and-price. We are now going to introduce this method as in [LD05], [VW10] and [BS06].

We want to find the optimal solution of an integer problem with compact form (2.10). Before applying branch-and-price we make use of a Dantzig-Wolfe reformulation, such as the convexification (2.11) or the discretization (2.18) to receive an extensive formulation. Then we set up a branch-and-bound procedure. At the beginning of branch-and-price we have the root node of the branch-and-bound tree. Here we can get an initial global upper bound on the optimal objective value by finding a feasible solution for the compact or extensive formulation by using some heuristic. Then we solve the linear relaxation of the extensive formulation with column generation. The linear relaxation of the extensive formulation is the master problem of the root node. This yields a lower bound on the optimal objective value of the original problem. If this solution of the master problem corresponds to an integer solution of the compact formulation of the problem we have already found an optimal solution. Otherwise we branch, i.e. we create at least two child vertices in the branch-and-bound tree. Each new vertex is associated with its own master problem which is the master problem of its parent with some additional constraints. These constraints need to be chosen in a way that ensures that the current solution of the linear relaxation does not come up again. Also, we make sure that every feasible integer solution of the problem of the parent node corresponds to a feasible solution of one of the problems of a child vertex.

Then we go to a vertex in the branch-and-bound tree which we have not yet dealt with and solve the corresponding master problem using column generation. For the initial set of columns for the restricted master problem we take the set of columns which led to the solution of the master problem of the parent node. If this does not make the restricted master problem of the current node feasible we try to add more columns such that it gets feasible. When this is not possible, we have finished dealing with this vertex. If the solution of the current master problem corresponds to an integer solution of the original problem, we compare the objective value of this current integer solution with the current upper bound and update the upper bound if it is thereby improved. Otherwise, this gives us a lower bound on the objective value that could come from this branch of the tree, which we compare to

the global upper bound. If this lower bound is higher than the global upper bound, we know that the optimal integer solution of the original problem cannot be found in this branch and we can hence discard it. If this is not the case, we want to search this branch further, which we do by branching again, i.e. creating new vertices in the tree which we need to deal with at some later point in the branch-and-price procedure.

Then we move on to the next vertex in the tree which was not yet treated. After handling all vertices of the tree, we can be certain that the integer solution of the original problem which lead to the global upper bound is indeed an optimal solution.

All in all, the branch-and-price procedure looks as in algorithm 7.

Algorithm 7 Branch-and-Price

Input compact and extensive formulation of an integer linear program
Output an optimal solution $\boldsymbol{x}^*$ of the integer linear program and its objective value UB

1: Set global upper bound $UB = \infty$, or initialize the upper bound by the objective value of some heuristic solution of the integer linear program and also initialize $\boldsymbol{x}^*$ to that solution.
2: Initialize the branch-and-bound tree BT by one undiscovered root node to which the linear relaxation of the extensive formulation of the integer linear program is assigned as a problem.
3: **while** there exists an undiscovered vertex in BT **do**
4: Choose an undiscovered vertex and mark it as discovered.
5: Solve the problem assigned to the chosen vertex using column generation, yielding a solution $\boldsymbol{\lambda}$ with objective value v.
6: **if** $v < UB$ **then**
7: **if** $\boldsymbol{\lambda}$ corresponds to an integer solution $\boldsymbol{x}$ of the original problem **then**
8: Set $UB = v$ and $\boldsymbol{x}^* = \boldsymbol{x}$.
9: **else** branch, i.e. add at least two child nodes to the current vertex and assign to them the problems obtained from the problem assigned to the current vertex with additional constraints. Here, the additional constraints need to be such that $\boldsymbol{\lambda}$ is not feasible for these problems and all feasible integer solutions of the problem assigned to the current vertex are feasible for at least one of the problems assigned to the child nodes. Mark the new vertices as undiscovered.
10: **end if**
11: **end if**
12: **end while**
13: **return** $(\boldsymbol{x}^*, UB)$

We will now see, that it is especially important how we choose to perform the branching and how we carry out column generation in order to perform branch-and-price meaningfully, see [VW10].

3.2 Branching Strategies

The branching process must be such that the following properties are fulfilled, see [LD05].

1. the fractional solution which led to branching is not present in any of the new problems,
2. all integer solutions appear in at least one of the new problems,
3. the branch-and-price algorithm terminates in finite time.

Furthermore, we aim for the branching to

4. partition the solution space,
5. have branches of roughly equal size,
6. maintain the structure of the pricing problem.

In particular, 5 means that the solution space of the subproblems assigned to the new nodes have almost equal size and we therefore hope that the branch-and-bound tree will be balanced. With these guiding principles for branching, we now look at specific branching strategies, as in [DL11], [VW10] and [Gam10].

3.2.1 Branching on the Variables of the Extended Formulation

As we perform column generation for the linear relaxation of the extended formulation of a problem it is a somewhat natural idea to branch directly on the variables in the extended formulation. However, when we are given a solution $\boldsymbol{\lambda}^v$ of the current node v it is not always straightforward to see whether this corresponds to an integral solution of the original formulation of the problem. For instance, when the extended formulation resulted from a convexification of some compact formulation it can happen that $\boldsymbol{\lambda}^v$ is integral while the corresponding solution of the original problem is not and vice versa. Thus, we cannot branch depending only on the integrality of $\boldsymbol{\lambda}^v$ in such cases. Nevertheless, there are cases where we have a direct correspondence between integral solutions of the extended formulation of the problem and the original formulation. For example when Dantzig-Wolfe discretization was used to get the extended formulation. We now consider this case.

Suppose we have a fractional solution $\boldsymbol{\lambda}^v$ of the extended formulation of the master problem at the current node v. A first branching strategy could be to find an index s with λ_s^v fractional and enforce $\lambda_s \leq \lfloor \lambda_s^v \rfloor$ in one child node and $\lambda_s \geq \lceil \lambda_s^v \rceil$ in the other child node for a new solution $\boldsymbol{\lambda}$.

This is a valid branching strategy. However, it can lead to very unbalanced branching trees as the restriction $\lambda_s \leq \lceil \lambda_s^v \rceil$ is usually far less restrictive than $\lambda_s \geq \lceil \lambda_s^v \rceil$, because most variables λ_s will be zero in an optimal solution. Furthermore, when we add the restriction $\lambda_s \leq \lfloor \lambda_s^v \rfloor$ to the master problem, the corresponding column $\boldsymbol{a}_s$ might be a solution of the pricing problem and can be added again to the master problem. Thus, in order to enforce this

restriction we need to exclude this solution from the pricing problem. Adding this restriction to the pricing problem might destroy the structure of it. This can significantly increase the running time needed to solve the pricing problem, see [DL11] and [VW10].

3.2.2 Branching on the Variables of the Compact Formulation

In particular in the case where integrality on the variables of the extended formulation does not correspond to the same as integrality of the solution of the compact formulation, we can make the branching decisions based on the variables of the compact formulation of the problem.

Of course, we need a compact formulation for our problem in order to use this form of branching. In [Vil+05] it is shown that given an extended formulation of a problem it is possible to construct a compact formulation provided that the extensive formulation satisfies some rather weak conditions.

For the remainder of this section we use the notation from Section 2.2.3. When we have a solution $\boldsymbol{\lambda}^v$ of the current extended formulation (2.19) which comes from some compact formulation (2.10), this corresponds to a solution $\boldsymbol{x}^v = \sum_{g \in G^{A_2}} \boldsymbol{g}\lambda_g^v$ of the compact formulation of the problem. If $\boldsymbol{x}^v$ is fractional there exists and index i with $x_i^v = \sum_{\boldsymbol{g} \in G^{A_2}} g_i \lambda_g^v \notin \mathbb{Z}_{\geq 0}$. Then we can branch by adding one child node with $x_i \leq \lfloor x_i^v \rfloor$ as an additional restriction and another child node with $x_i \geq \lceil x_i^v \rceil$. There are two main possibilities how this can be enforced during the column generation procedure of these nodes. We now consider how to enforce $x_i \geq \lceil x_i^v \rceil$, the constraint $x_i \leq \lfloor x_i^v \rfloor$ can be dealt with analogously.

Enforcing Branching using Constraints in the Master Problem

Consider adding $\sum_{\boldsymbol{g} \in G^{A_2}} g_i \lambda_g \geq \lceil x_i^v \rceil$ to the master problem. This additional restriction corresponds to a new dual variable μ which needs to be included in the pricing problem. Suppose that the current pricing problem is $\min\{\boldsymbol{c}^\intercal \boldsymbol{g} - \boldsymbol{u}^{v\intercal} A_1 \boldsymbol{g} : \boldsymbol{g} \in G^{A_2}\}$, where $\boldsymbol{u}^v$ is the optimal dual solution of the current restricted master problem. Then the pricing problem of a new node is $\min\{\boldsymbol{c}^\intercal \boldsymbol{g} - \tilde{\boldsymbol{u}}^\intercal A_1 \boldsymbol{g} - \tilde{\mu} g_i : \boldsymbol{g} \in G^{A_2}\}$. Here $(\tilde{\boldsymbol{u}}, \tilde{\mu})$ is the optimal dual solution of the corresponding new restricted master problem, where $\tilde{\mu}$ corresponds to the newly added constraint. As only the objective function of the pricing problem changes, the structure of the pricing problem is not affected. Hence, in most cases we can still solve the pricing problem using the same methods as before the branching, e.g. many combinatorial algorithms still work. When the extended formulation comes from a Dantzig-Wolfe convexification, the new lower bound for the new node is given as $\min\{\boldsymbol{c}^\intercal \boldsymbol{x} : A_1 \boldsymbol{x} \geq \boldsymbol{b_1}, x_i \geq \lceil x_i^v \rceil, \boldsymbol{x} \in conv\{\boldsymbol{x} \in \mathbb{Z}^k : A_2 \boldsymbol{x} \geq \boldsymbol{b_2}\}\}$.

Enforcing Branching in the Generating Columns

Assume that the extended formulation comes from a Dantzig-Wolfe convexification or discretization. Then we can enforce the branching constraint $\sum_{p \in P} x_{p,i} \lambda_p + \sum_{r \in R} v_{r,i} \lambda_r \geq \lceil x_i^v \rceil$

by allowing only generators which satisfy this constraint. Here $x_{p,i}$ is the i-th component of the vector $\boldsymbol{x}_p$ for all $p \in P$ and $v_{r,i}$ is the i-th component of the vector $\boldsymbol{v}_r$ for all $r \in R$. This means that all points $p \in P$ must satisfy the constraint $x_{p,i} \geq \lceil x_i^v \rceil$ and all rays $r \in R$ must fulfill $v_{r,i} \geq 0$. Thus the rays are not affected, as $\boldsymbol{v}_r \geq \mathbf{0}$ for all rays r. Due to the convexity constraint of the points, the solution of the master problem must then also satisfy $\sum_{p \in P} x_{p,i}\lambda_p + \sum_{r \in R} v_{r,i}\lambda_r \geq \lceil x_i^v \rceil$. We can make sure that all points and rays of the restricted master problem fulfill the branching constraint by taking precisely the points and rays of the restricted master problem of the parent node which satisfy the branching constraint into the master problem. This can be done by setting all variables corresponding to points and rays of the parent node which do not fulfill the branching constraint to zero. In order to enforce this we call the set of all such points $\tilde{P}$ and the set of all such rays $\tilde{R}$ and add $\sum_{s \in \tilde{P} \cup \tilde{R}} \boldsymbol{\lambda}_s = 0$ to the restricted master problem. If this makes the restricted master problem infeasible, we need to add more columns which fulfill the constraint prior to starting column generation. Furthermore, we need to make sure that new columns added to the restricted master problem also satisfy the branching constraint. For this, we add the constraint $x_i \geq \lceil x_i^v \rceil$ to the pricing problem. Hence, the pricing problem becomes $\min\{\boldsymbol{c}^\intercal \boldsymbol{x} - \boldsymbol{u}^{v\intercal} A_1 \boldsymbol{x} - u_0^v : \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2},\, x_i \geq \lceil x_i^v \rceil\}$, where $(\boldsymbol{u}^v, u_0^v)$ is the optimal solution of the new dual master problem. However, as we add a restriction to the pricing problem, we might not be able to use the structure of the problem to solve it anymore. Furthermore, the points P and the rays R need to be changed such that they represent the polyhedron $conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2},\, x_i \geq \lceil x_i^v \rceil\}$. When the extended formulation comes from a Dantzig-Wolfe convexification, the lower bound for this node is $\min\{\boldsymbol{c}^\intercal \boldsymbol{x} : A_1\boldsymbol{x} \geq \boldsymbol{b_1}, \boldsymbol{x} \in conv\{\boldsymbol{x} \in \mathbb{Z}^k : A_2\boldsymbol{x} \geq \boldsymbol{b_2}, x_i \geq \lceil x_i^v \rceil\}\}$.

3.2.3 Branching on Aggregated Variables

When we have multiple identical blocks which can be aggregated in the compact formulation of the problem, we have seen how the Dantzig-Wolfe reformulation can reduce the symmetry of the problem. However, when we branch on one of the variables corresponding to one of the identical blocks in the compact formulation, this changes the block in the master problem or the corresponding pricing problem. Thus, the blocks are no longer identical. However, as the blocks are identical it is likely to happen that the restrictions on the identical blocks are just permuted on other branches. Hence, the symmetry of the problem is no longer dealt with.

When we are in the case where the integrality of the compact formulation of the problem is equivalent to the integrality of the extended formulation of the problem, we can branch on the aggregated variables in Problem (2.21). This reduces the symmetry. However, the integrality of the aggregated variables does not imply the integrality of the individual variables of the extended formulation. Nevertheless, we can still apply this branching method until all aggregated variables are integral and use other branching strategies afterwards.

The branching on aggregated variables can be applied by enforcing the constraints $\sum_{j \in S} \lambda_{\boldsymbol{g}}^j \geq \lceil \lambda_{\boldsymbol{g}}^{Sv} \rceil$ or $\sum_{j \in S} \lambda_{\boldsymbol{g}}^j \leq \lfloor \lambda_{\boldsymbol{g}}^{Sv} \rfloor$ to the master problem of the child nodes. This changes the

objective function of the pricing problem. Here the notation is as in Section 2.2.4 and $\lambda_{\boldsymbol{g}}^{Sv}$ is the sum of the aggregated variables of blocks corresponding to S in an optimal solution of the current master problem.

3.2.4 Ryan and Forster branching

Let us now look at a branching strategy which works well for problems with a so-called set partitioning structure, as described in [RF81], [Gam10] and [VW10]. A problem with set partitioning structure is a problem which can be written in the following form, see [RF81].

$$\begin{aligned} \min_{(x_1,...,x_{|J|})} \quad & \sum_{j\in J} c_j x_j \\ \text{s.t.} \quad & \sum_{j\in J} \boldsymbol{a}_j x_j = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}, \\ & x_j \in \{0,1\}, \quad \forall j \in J, \end{aligned} \tag{3.1}$$

where $c_j \in \mathbb{R}$ and $\boldsymbol{a}_j \in \{0,1\}^n$ for all $j \in J$. There are many known problems with this structure, including the bin packing problem and the vertex coloring problem, see [Gam10]. For the remainder of this section, let us consider problems with this structure.

The i-th constraint in Problem (3.1) is $\sum_{j\in J} a_{j,i} x_j = 1$, where $a_{j,i}$ is the i-th component of the vector $\boldsymbol{a}_j$. Then towards the Ryan and Forster branching strategy, we need to observe that this constraint is fulfilled by a feasible solution $\boldsymbol{x}$ if and only if $x_{j'} = 1$ for precisely one j' with $a_{j',i} = 1$, and $x_j = 0$ for all other j with $a_{j,i} = 1$. When we consider two constraints with index i' and i'' there can either

1. be one $x_{j'} = 1$ with $\boldsymbol{a}_{j',i'} = \boldsymbol{a}_{j',i''} = 1$,
2. or there are two variables $x_{j'} = x_{j''} = 1$ such that $\boldsymbol{a}_{j',i'} = \boldsymbol{a}_{j'',i''} = 1$ and $\boldsymbol{a}_{j',i''} = \boldsymbol{a}_{j'',i'} = 0$.

The Ryan and Forster branching is then based on the fact, that we have a fractional solution of a set partitioning problem (3.1) if and only if there exist two constraints i, i' such that

$$\sum_{j\in J:\boldsymbol{a}_{j,i}=\boldsymbol{a}_{j,i'}=1} x_j \in (0,1)$$

is fractional, see [RF81]. Hence, we can branch on this expression. We enforce

$$\sum_{j\in J:\boldsymbol{a}_{j,i}=\boldsymbol{a}_{j,i'}=1} x_j \geq 1 \tag{3.2}$$

on one child node and

$$\sum_{j\in J:\boldsymbol{a}_{j,i}=\boldsymbol{a}_{j,i'}=1} x_j \leq 0 \tag{3.3}$$

on the other child node. Thus, we are in the case where there exists one variable $x_{j'} = 1$ with $\boldsymbol{a}_{j',i'} = \boldsymbol{a}_{j',i''} = 1$ when we enforce Condition (3.2). And we are in the case where there are two variables $x_{j'} = x_{j''} = 1$ such that $\boldsymbol{a}_{j',i'} = \boldsymbol{a}_{j'',i''} = 1$ and $\boldsymbol{a}_{j',i''} = \boldsymbol{a}_{j'',i'} = 0$ when we enforce Condition (3.3). The first condition (3.2) is equivalent to

$$\sum_{j \in J : \boldsymbol{a}_{j,i} \neq \boldsymbol{a}_{j,i'}} x_j \leq 0, \tag{3.4}$$

as we have a problem with set partitioning structure and hence $a_{j,i} \in \{0, 1\}$ for all $j \in J, i \in \{1, ..., n\}$, $\sum_{j \in J} = a_{j,i} x_j = \sum_{j \in J} = a_{j,i'} x_j = 1$ and summands where $a_{j,i} = a_{j,i'} = 0$ do not contribute to both of these sums. Thus, we can enforce such a branching in the generating columns by adding the restriction $\boldsymbol{a}_{j,i} = \boldsymbol{a}_{j,i'}$ and $\boldsymbol{a}_{j,i} + \boldsymbol{a}_{j,i'} \leq 1$ to the pricing problem for Condition (3.2) and Condition (3.4), respectively, and remove all columns of the restricted master problem which do not satisfy the corresponding restrictions.

The main advantages of this branching method are, that it can handle Dantzig-Wolfe decompositions coming from multiple identical blocks and it guarantees an integral solution if no more branching is possible. The main disadvantage is that it can only be applied for problems with a set partitioning structure.

There exist considerably more branching strategies, for instance a rather generic branching strategy is described in [Van10]. This method can handle Dantzig-Wolfe decompositions which come from multiple identical blocks without reintroducing symmetry and the subproblem is only modified using bounds on its variables. Nonetheless, it has the disadvantage that it can be necessary to solve more than one subproblem for each type of identical blocks and that one node might have more than two children in the branch and bound tree.

3.3 Heuristics for Upper Bound

In [Lü11] it is emphasized that the use of heuristics can significantly improve column generation and branch-and-price. Heuristics can be used for various purposes in the context of branch-and-price. For instance, for solving the pricing problem, as this step is typically expensive. The heuristics for solving the pricing problem are rather problem specific, thus we will not go into further detail. Another useful application of heuristics in branch-and-price is in the computation of good global upper bounds. Having a good upper bound is very useful, as branches of the branch-and-bound tree can be discarded whenever the objective value of the corresponding master problem is higher than the upper bound.

For all of these heuristics, there is always a trade-off between the time to perform the heuristics and the quality of the bound. Furthermore, there is no guarantee to find a new or improve an existing upper bound. In order to mitigate this kind of disadvantages, we might set some kind of maximum running time or a maximum number of iterations for the heuristic.

We will now discuss a selection of the heuristics to get or improve the global upper bound for branch-and-price as described in [Ber08], [Ach07], [Puc11] and [Ber06].

3.3.1 Rounding Heuristics

At every node of the branch-and-bound tree in the branch-and-price procedure we get a solution $\boldsymbol{x}$ which is feasible for the linear relaxation of the original problem. If this solution is integer, it gives us a global upper bound for the problem. However, if it is fractional this is not the case. An idea to still obtain an upper bound is to round the value of every variable in the solution such that the new solution becomes integer and stays feasible. This gives a global upper bound for the original problem. By rounding a variable x_j up we mean rounding it to $\lceil x_j \rceil$ and by rounding down we mean rounding it to $\lfloor x_j \rfloor$. There are different methods to achieve this.

Rounding

One approach is to iteratively round fractional variables and if this leads to a violation of some constraint try to regain feasibility through rounding other fractional variables accordingly. More precisely, suppose we have an original problem of the form (2.10) and a fractional solution $\bar{\boldsymbol{x}}$. For simplicity we merge the constraints $A_1\boldsymbol{x} \geq \boldsymbol{b_1}$ and $A_2\boldsymbol{x} \geq \boldsymbol{b_2}$ to $A\boldsymbol{x} \geq \boldsymbol{b}$. We want to round the variables such that only few constraints are violated. Notice that rounding a variable x_i up can only cause violating the constraint $A_{k,\cdot}\boldsymbol{x} \geq b_k$ if $A_{k,i}$ is negative. And rounding down can only cause the violation of this constraint, if $A_{k,i}$ is positive. Here, $A_{k,\cdot}$ is the k-th row of A and $A_{k,i}$ is the i-th entry of $A_{k,\cdot}$. Thus, we want to first pick a fractional variable $\bar{x}_i$ with a highest number of $\max\{|\{k : A_{k,i} < 0\}|, |\{k : A_{k,i} > 0\}|\}$. Then we round this chosen variable in the direction where fewer entries of $A\bar{\boldsymbol{x}}$ are decreased. If after the rounding we are still feasible for $A\boldsymbol{x} \geq \boldsymbol{b}$ and still have fractional variables, we repeat this process. If however, there are violated constraints and fractional variables, we try to restore the linear programming feasibility by rounding the other fractional variables. In order to do so we select one violated constraint $A_{k,\cdot}\boldsymbol{x} \geq b_k$ and search for fractional variables x_i with $A_{k,i} \neq 0$ and round such a variable up if $A_{k,i} > 0$ and down if $A_{k,i} < 0$. If no such fractional variable exists, we stop the procedure as we cannot restore feasibility. When there are multiple such variables, we choose to round a variable which makes the fewest entries of $A\bar{\boldsymbol{x}}$ smaller when the rounding is performed. If there are no more fractional variables we have reached the end of the rounding procedure. When our rounded variables satisfy $A\boldsymbol{x} \geq \boldsymbol{b}$ we have found a feasible solution for the original problem.

Shifting

When we perform the rounding procedure described above and encounter a violated solution which cannot be restored by rounding the remaining fractional variables, we stop. However, we could also take variables which are already integer and change them to different integer values. This procedure is called shifting. Notice that the shifting procedure could cycle,

as shifting a variable for one violated constraint might lead to another constraint being violated and then shifting the variable back restores this second constraint. To prevent this from happening, we prefer variables which have not been shifted in the other direction yet, or at least not recently. Furthermore, we count the number of steps where the number of violated constraints has not reached a new lowest point since the last rounding of a fractional variable and stop after a certain number is exceeded. Notice that the shifting procedure can be significantly more expensive than the rounding procedure and should thus be performed less often.

Relaxation Enforced Neighborhood Search

In [Ber07] the relaxation enforced neighborhood search is described. In this heuristic, we take a feasible solution $\bar{\boldsymbol{x}}$ of the linear relaxation of the original problem (2.10) and search for the best way to round all fractional variables. This is done by creating a subproblem of the original problem, where we add the restrictions that for every index i, $\lfloor \bar{x}_i \rfloor \leq x_i \leq \lceil \bar{x}_i \rceil$. In particular, this includes fixing every variable which is already integral. Hence, our new problem looks as follows.

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{c}^\intercal \boldsymbol{x} \\ \text{s.t.} \quad & A_1 \boldsymbol{x} \geq \boldsymbol{b_1}, \\ & A_2 \boldsymbol{x} \geq \boldsymbol{b_2}, \\ & \boldsymbol{x} \geq \lfloor \bar{\boldsymbol{x}} \rfloor, \\ & \boldsymbol{x} \leq \lceil \bar{\boldsymbol{x}} \rceil, \\ & \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0}, \end{aligned} \tag{3.5}$$

If this new problem (3.5) is not feasible, we stop the heuristic. Otherwise we can solve the problem and get a solution which is also feasible for the original problem (2.10). Notice that the solution of Problem (3.5) is the best feasible solution of the original problem which can be obtained by rounding $\bar{\boldsymbol{x}}$. However, solving Problem (3.5) can be rather time expensive. Thus, this heuristic should only be used when we can solve Problem (3.5) in reasonable time, for instance when many variables are fixed, i.e. were already integral.

3.3.2 Diving Heuristics

The main observation on which diving heuristics build upon is, that we can get a feasible solution for our original problem (2.10) when we reach a leaf of the branch-and-bound tree. To get to the first leaf as soon as possible, we could therefore traverse the tree in a depth-first search order until we reach the first leaf. Furthermore, the main goal of diving heuristics is to find a feasible solution and not to have a balanced search tree. Thus, a diving heuristic will typically not use the same branching as during the branch-and-bound procedure, but rather use a rule which drives the branch to a feasible solution. The branching can be done by bounding fractional variables of the solution of the linear relaxation of the current problem. There are different approaches on the selection of the fractional variable to be bounded next.

For instance, one can take the fractional variable which is closest to an integer, or with a lowest non-zero number of $\min\{|\{k : A_{k,i} < 0\}|, |\{k : A_{k,i} > 0\}|\}$. In order to prevent too expensive dives, we keep track of how deep we are in the search tree and stop at a certain depth.

3.3.3 Feasibility Pump

Classic Feasibility Pump

In [BFL07] and [FGL05] the feasibility pump heuristic for finding a feasible solution for Problem (2.10) is described. It is based on the fact, that $\boldsymbol{x}$ is feasible for the original problem if it is both feasible for the linear relaxation of the problem and integral. The heuristic starts at the solution $\bar{\boldsymbol{x}}^{(0)}$ of the linear relaxation of the original problem. Then $\bar{\boldsymbol{x}}^{(0)}$ is rounded in every component to the nearest integer to get $\tilde{\boldsymbol{x}}^{(0)}$. If $\bar{\boldsymbol{x}}^{(0)} = \tilde{\boldsymbol{x}}^{(0)}$, this must be a solution of the original problem and we stop the procedure. Otherwise, we find the solution $\bar{\boldsymbol{x}}^{(1)}$ of the linear relaxation of the original problem which is as close to $\tilde{\boldsymbol{x}}^{(0)}$ as possible. Where with "close" we mean that the sum of the absolute differences of the variables is small, i.e. we measure the distance between $\tilde{\boldsymbol{x}}$ and $\bar{\boldsymbol{x}}$ as $\sum_{i=1}^{n} |\tilde{x}_i - \bar{x}_i|$. Then, $\bar{\boldsymbol{x}}^{(1)}$ is rounded again, which yields $\tilde{\boldsymbol{x}}^{(1)}$. If $\bar{\boldsymbol{x}}^{(1)} = \tilde{\boldsymbol{x}}^{(1)}$ we have found a feasible solution of the original problem and stop the procedure. Otherwise we continue to iteratively find the closest point which is feasible for the linear relaxation and round it. Notice, that every $\bar{\boldsymbol{x}}$ is feasible for the linear relaxation of the original problem, while every $\tilde{\boldsymbol{x}}$ is integral. When this method starts to cycle it is possible to change an integer solution to another close-by integer solution and continue the procedure from there. Also, we want to make sure to set an upper bound on the number of iterations.

Objective Feasibility Pump

A downside of the classical feasibility pump heuristic is that the objective function is only taken into account at the initialization. Therefore, in [AB07] a version of the feasibility pump heuristic is proposed which also considers the objective functions at higher iterations. For this version, we assume that the objective function is not always 0. The objective feasibility pump heuristics works similar to the classic feasibility pump heuristic. But, instead of finding a feasible solution $\bar{\boldsymbol{x}}$ of the linear relaxation which minimizes $\sum_{i=1}^{n} |\tilde{x}_i - \bar{x}_i|$ for the current integral $\tilde{\boldsymbol{x}}$, we want to find a feasible solution of the linear relaxation which minimizes a convex combination of this distance and the objective function. More precisely, we want to minimize $(1-\alpha)\sum_{i=1}^{n} |\tilde{x}_i - \bar{x}_i| + \alpha(\boldsymbol{c}^\intercal \bar{\boldsymbol{x}}\sqrt{n}/||\boldsymbol{c}||)$ for some $\alpha \in [0, 1]$. Here, $\sqrt{n}/||\boldsymbol{c}||$ is some kind of normalization for the objective function with $||.||$ being the Euclidean norm. In every iteration we multiply α by some constant in $[0, 1]$, which stays fixed during the heuristic, to get our new α. This has the effect, that we put a stronger emphasis on being integral in every iteration and thus allow for $\bar{\boldsymbol{x}}$ with a worse objective value. Again, this procedure is stopped if we find a feasible solution of the original problem or if we reach a certain number of iterations.

3.3.4 Quality of the Heuristics

In [Ber08] computational experiments show that every single heuristic on its own only has a rather small impact on the running time of branch-and-price. Except, for the objective feasibility bump heuristic. However, when using multiple different heuristics together, this reduces the running time significantly. For a more detailed discussion on the advantages and disadvantages of the individual heuristics, see [Ber06].

3.4 Branch-Price-and-Cut

During a branch-and-bound procedure, instead of branching on a fractional variable, it is also possible to add a feasible cut every now and then. This is known as branch-and-cut, see [Elf+01]. When we additionally solve the linear relaxation of the problems assigned to the nodes of the branch-and-bound tree by using column generation, we call this approach branch-price-and-cut. In other words, in order to solve the integer linear problem (2.10), we modify branch-and-price such that whenever we would branch, we decide between adding cutting planes and branching.

Algorithm 8 represents a generic branch-price-and-cut approach.

Suppose that at a certain node v of the branch-price-and-cut procedure we solve the linear relaxation of the extended formulation (2.19), which has the compact form (2.10). Let $\boldsymbol{\lambda}^v$ be a solution of the linear relaxation of the extended problem and let $\boldsymbol{x}^v$ be a corresponding feasible solution of the linear relaxation of the compact formulation. If $\boldsymbol{x}^v$ is fractional, it is not yet optimal for the compact formulation. Thus, we need to branch or cut. When adding cutting planes, we need to enforce them in the new problem. We will now see how to enforce different types of cutting planes as in [DL11] and [DDS11].

3.4.1 Cuts on the Variables of the Compact Formulation

Suppose that we find one or more cutting planes of the form $A_3\boldsymbol{x} \geq \boldsymbol{b_3}$ separating $\boldsymbol{x}^v$ from the optimal feasible solution of (2.10). This means that we now want to solve the following problem.

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{c}^\intercal \boldsymbol{x} \\ \text{s.t.} \quad & A_1\boldsymbol{x} \geq \boldsymbol{b_1}, \\ & A_2\boldsymbol{x} \geq \boldsymbol{b_2}, \\ & A_3\boldsymbol{x} \geq \boldsymbol{b_3}, \\ & \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0}, \end{aligned} \tag{3.6}$$

Then, similar to enforcing branching decisions on variables of the compact formulation, we can enforce the cuts either directly in the master problem or by using the generating columns.

Algorithm 8 Branch-Price-and-Cut

Input compact and extensive formulation of an integer linear program
Output an optimal solution $\boldsymbol{x}^*$ of the integer linear program and its objective value UB

1: Set global upper bound $UB = \infty$, or initialize the upper bound by the objective value of some heuristic solution of the integer linear program and also initialize $\boldsymbol{x}^*$ to that solution.
2: Initialize the branch-and-bound tree BT by one undiscovered root node to which the linear relaxation of the extensive formulation of the integer linear program is assigned as a problem.
3: **while** there exists an undiscovered vertex in BT **do**
4: Choose an undiscovered vertex and mark it as discovered.
5: Solve the problem assigned to the chosen vertex using column generation, yielding a solution $\boldsymbol{\lambda}$ with objective value v.
6: **if** $v < UB$ **then**
7: **if** $\boldsymbol{\lambda}$ corresponds to an integer solution $\boldsymbol{x}$ of the original problem **then**
8: Set $UB = v$ and $\boldsymbol{x}^* = \boldsymbol{x}$.
9: **else if** we want to add cuts **then**
10: add one or more cutting planes, i.e. add one child node to the current vertex and assign to it the problem of the current vertex with additional cutting constraints such that $\boldsymbol{\lambda}$ is not feasible for the problem assigned to the child node. Mark the new vertex as undiscovered.
11: **else** branch, i.e. add at least two child nodes to the current vertex and assign to them problems obtained from the problem of the current vertex by adding additional constraints. Here, the additional constraints need to be such that $\boldsymbol{\lambda}$ is not feasible for these problems and all feasible integer solutions of the problem assigned to the current vertex are feasible for at least one of the problems assigned to the child nodes. Mark the new vertices as undiscovered.
12: **end if**
13: **end if**
14: **end while**
15: **return** $(\boldsymbol{x}^*, UB)$

Enforcing Cuts in the Master Problem

Suppose we want to enforce $A_3\boldsymbol{x} \geq \boldsymbol{b_3}$ in the master problem. Then we add these constraints to the master problem. The linear relaxation of the following problem yields the new master problem.

$$\begin{aligned} \min_{\boldsymbol{\lambda}} \quad & \sum_{\boldsymbol{g} \in G^{A_2}} \boldsymbol{c}^\intercal \boldsymbol{g} \lambda_{\boldsymbol{g}} \\ \text{s.t.} \quad & \sum_{\boldsymbol{g} \in G^{A_2}} A_1 \boldsymbol{g} \lambda_{\boldsymbol{g}} \geq \boldsymbol{b_1}, \\ & \sum_{\boldsymbol{g} \in G^{A_2}} A_3 \boldsymbol{g} \lambda_{\boldsymbol{g}} \geq \boldsymbol{b_3}, \\ & \boldsymbol{\lambda} \in W^{A_2}. \end{aligned} \tag{3.7}$$

However, this yields additional dual variables and thus changes the pricing problem to $\min\{\boldsymbol{c}^\intercal \boldsymbol{g} - \boldsymbol{u}^\intercal A_1 \boldsymbol{g} - \boldsymbol{\mu}^\intercal A_3 \boldsymbol{g} : \boldsymbol{g} \in G^{A_2}\}$. Here $(\boldsymbol{u}, \boldsymbol{\mu})$ is the optimal dual solution of the corresponding new restricted master problem, where $\boldsymbol{\mu}$ corresponds to the newly added constraints.

Enforcing Cuts using Generating Columns

Enforcing $A_3\boldsymbol{x} \geq \boldsymbol{b_3}$ by using the generating columns looks as follows. We consider the case, where the extended formulation comes from a Dantzig-Wolfe convexification or discretization. The constraint is then $\sum_{p \in P} A_3 \boldsymbol{x}_p \lambda_p + \sum_{r \in R} A_3 \boldsymbol{v}_r \lambda_r \geq \boldsymbol{b_3}$. We want to enforce this constraint by allowing only columns which satisfy this constraint. This is done by updating the points P and the rays R such that they represent the polyhedron $conv\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}, A_3\boldsymbol{x} \geq \boldsymbol{b_3}\}$. Furthermore, we only keep points p of the restricted master problem of the parent node which satisfy $A_3\boldsymbol{x}_p \geq \boldsymbol{b_3}$ and the rays r which satisfy $A_3\boldsymbol{v}_r \geq \boldsymbol{0}$. The pricing problem changes to $\min\{\boldsymbol{c}^\intercal \boldsymbol{x} - \boldsymbol{u}^{v\intercal} A_1 \boldsymbol{x} - u_0^v : \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}, A_3\boldsymbol{x} \geq \boldsymbol{b_3}\}$, where $(\boldsymbol{u}^v, u_0^v)$ is the optimal solution of the new dual master problem. This might change the structure of the pricing problem, but can potentially provide a stronger dual bound.

3.4.2 Cuts on the Variables of the Extended Formulation

It is also possible to have cutting planes involving the variables of the extended formulation. This might be especially meaningful when we have integrality constraints on the variables in the extended formulation, for instance in the case of a Dantzig-Wolfe discretization. In the following we use the notation of Section 2.2.3. Suppose first that we have cutting planes of the form $\sum_{\boldsymbol{g} \in G^{A_2}} A_3 \boldsymbol{g} \lambda_{\boldsymbol{g}} \geq \boldsymbol{b_3}$. These constraints correspond to cutting planes $A_3\boldsymbol{x} \geq \boldsymbol{b_3}$ involving variables of the compact formulation and can thus be treated as in Section 3.4.1. Assume now that we have cutting planes of the form $\sum_{\boldsymbol{g} \in G^{A_2}} \boldsymbol{a}_{g,3} \lambda_{\boldsymbol{g}} \geq \boldsymbol{b_3}$, where $\boldsymbol{a}_{g,3} =$

$f(A_1\boldsymbol{g})$ for some function f. This yields the following new extended formulation.

$$\begin{aligned}
\min_{\boldsymbol{\lambda}} \quad & \sum_{\boldsymbol{g}\in G^{A_2}} \boldsymbol{c}^\intercal \boldsymbol{g}\lambda_{\boldsymbol{g}} \\
\text{s.t.} \quad & \sum_{\boldsymbol{g}\in G^{A_2}} A_1\boldsymbol{g}\lambda_{\boldsymbol{g}} \geq \boldsymbol{b_1}, \\
& \sum_{\boldsymbol{g}\in G^{A_2}} \boldsymbol{a}_{g,3}\lambda_{\boldsymbol{g}} \geq \boldsymbol{b_3}, \\
& \boldsymbol{\lambda} \in W^{A_2}.
\end{aligned} \tag{3.8}$$

Regarding the pricing problem of this formulation, we need to include the dual variables corresponding to the new constraints. This yields $\min\{\boldsymbol{c}^\intercal\boldsymbol{g} - \boldsymbol{u}^\intercal A_1\boldsymbol{g} - \boldsymbol{\mu}^\intercal f(A_1\boldsymbol{g}) : \boldsymbol{g} \in G^{A_2}\} = \min\{\boldsymbol{c}^\intercal\boldsymbol{g} - \boldsymbol{u}^\intercal A_1\boldsymbol{g} - \boldsymbol{\mu}^\intercal\boldsymbol{f} : \boldsymbol{g} \in G^{A_2},\ \boldsymbol{f} = f(A_1\boldsymbol{g})\}$. Here $(\boldsymbol{u}, \boldsymbol{\mu})$ is the optimal dual solution of the corresponding new restricted master problem, where $\boldsymbol{\mu}$ corresponds to the newly added constraints. The complexity of the pricing problem depends on the function f. If f is linear, we are again in the previous case of Section 3.4.1.

We can recreate a new compact formulation that gives us the new extended formulation after a Dantzig-Wolfe reformulation. To this end we introduce an additional variable $\boldsymbol{f} = f(A_1\boldsymbol{x})$. However, notice that the function f might not be linear.

$$\begin{aligned}
\min_{\boldsymbol{x}} \quad & \boldsymbol{c}^\intercal\boldsymbol{x} \\
\text{s.t.} \quad & A_1\boldsymbol{x} \geq \boldsymbol{b_1}, \\
& A_2\boldsymbol{x} \geq \boldsymbol{b_2}, \\
& \boldsymbol{f} \geq \boldsymbol{b_3}, \\
& \boldsymbol{f} = f(A_1\boldsymbol{x}), \\
& \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0},
\end{aligned} \tag{3.9}$$

From this new compact formulation we can derive the new extended formulation (3.8) by a Dantzig-Wolfe reformulation. We apply the Dantzig-Wolfe reformulation to $\{(\boldsymbol{x}, \boldsymbol{f}) : \boldsymbol{x} \in \mathbb{Z}^k_{\geq 0},\ A_2\boldsymbol{x} \geq \boldsymbol{b_2},\ \boldsymbol{f} = f(A_1\boldsymbol{x})\}$. Notice that the condition $\boldsymbol{f} = f(A_1\boldsymbol{x})$ does not affect the polyhedron $\{\boldsymbol{x} \in \mathbb{Z}^k_{\geq 0} : A_2\boldsymbol{x} \geq \boldsymbol{b_2}\}$, but only defines the value of the new variable $\boldsymbol{f}$. Thus, this yields Problem (3.8) as an extended formulation and $\min\{\boldsymbol{c}^\intercal\boldsymbol{g} - \boldsymbol{u}^\intercal A_1\boldsymbol{g} - \boldsymbol{\mu}^\intercal\boldsymbol{f} : \boldsymbol{g} \in G^{A_2},\ \boldsymbol{f} = f(A_1\boldsymbol{g})\}$ as a pricing problem.

For cuts with coefficients which can be represented as $f_2(\boldsymbol{f}, A_1\boldsymbol{g})$ for some function f_2, we apply this procedure again. We can repeat this any finite number of times, see [DDS11]. Such cuts have been successfully used for multiple applications, see for instance [BS06] and [PPS08].

4 The Fair Matching Over Time Problem

In [Lod+22] the fairness over time problem is described and the so-called ambulance allocation problem, a special fairness over time problem, is solved using branch-and-price and other methods. Using this as a motivation, we want to introduce a new specific fairness over time problem and solve it with a branch-and-price algorithm. Towards this, we first want to introduce the concept of fairness over time as in [Lod+22], [LSW23] and [LSW22].

4.1 Fairness Over Time

Suppose that some central decision maker has a set $\mathcal{X}$ of options to choose from. Furthermore, there are n stakeholders of this decision, and each of them will receive a certain utility from each of the options $\boldsymbol{x} \in \mathcal{X}$. These utilities can be represented using a utility-function $u : \mathcal{X} \to \mathbb{R}^n$, where $[u(\boldsymbol{x})]_i$ represents the utility obtained by the i-th stakeholder when option $\boldsymbol{x}$ is chosen by the central decision maker. The central decision maker wants to choose an option such that the utilities obtained by the stakeholders are as fair as possible, where fairness is defined by an unfairness function.

Definition 4.1.1 (Unfairness Function)**.** *An unfairness function is a function $\Phi : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ such that*

1. *$\Phi(y_1, ..., y_n) = 0$ if and only if $y_1 = y_2 = ... = y_n$ and*
2. *$\Phi(y_1, ..., y_n) = \Phi(y_{\pi(1)}, y_{\pi(2)}..., y_{\pi(n)})$, for all permutations $\pi \in \mathcal{S}_n$*

holds.

Example. *We can easily check that $\Phi(y_1, ..., y_n) = \max(y_1, ..., y_n) - \min(y_1, ..., y_n)$ is an unfairness function. First of all, if $y_1 = ... = y_n$, we know that $\max(y_1, ..., y_n) - \min(y_1, ..., y_n) = y_1 - y_1 = 0$. Suppose that $\max(y_1, ..., y_n) - \min(y_1, ..., y_n) = 0$. Then we have that $\min(y_1, ..., y_n) = \max(y_1, ..., y_n)$ and we already have $y_1 = ... = y_n$, as $y_i \leq \max(y_1, ..., y_n)$ and $y_i \geq \min(y_1, ..., y_n)$ for all $i \in \{1, ..., n\}$. Secondly, $\max(y_1, ..., y_n) - \min(y_1, ..., y_n)$ is permutation invariant, as the minimum function and the maximum function are both permutation invariant.*

This yields the single round fairness problem.

Definition 4.1.2 (Single Round Fairness Problem)**.** *Let $\mathcal{X}$ be a set of options, $u : \mathcal{X} \to \mathbb{R}^n$ a utility function and $\Phi : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ an unfairness function. Then the task is to find an option $\boldsymbol{x} \in \mathcal{X}$ which minimizes*

$$\min_{\boldsymbol{x} \in \mathcal{X}} \Phi([u(\boldsymbol{x})]_1, [u(\boldsymbol{x})]_2, ..., [u(\boldsymbol{x})]_n).$$

Often, such decisions have to be made multiple times. Then, based on these decisions each stakeholder will receive a respective utility in each round. When T decisions are made, where the options $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(T)}$ are chosen, the utilities obtained by the i-th stakeholder are $[u(\boldsymbol{x}^{(1)})]_i, [u(\boldsymbol{x}^{(2)})]_i, ..., [u(\boldsymbol{x}^{(T)})]_i$. To get the overall utility obtained by the i-th stakeholder through T rounds, we need to aggregate these utilities. For instance, we can take the average of the utilities and have $\frac{1}{T}\sum_{t=1}^{T}[u(\boldsymbol{x}^{(t)})]_i$ as the aggregated utility obtained by the i-th stakeholder. Then, we want to find a sequence of options such that the aggregated utilities obtained by the stakeholders are as fair as possible.

Definition 4.1.3 (Fairness Over Time Problem)**.** *Let $\mathcal{X}$ be a set of options, $u : \mathcal{X} \to \mathbb{R}^n$ a utility function, $\Phi : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ an unfairness function and T the number of decisions. Then the task is to find a sequence of options $(\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(T)}) \in \mathcal{X}^T$ which minimizes*

$$\begin{aligned} \min_{(\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(T)})} \quad & \Phi(y_1, ..., y_n) \\ \text{s.t.} \quad & y_i = \frac{1}{T}\sum_{t=1}^{T}[u(\boldsymbol{x}^{(t)})]_i, \ \forall i \in \{1, ..., n\}, \\ & \boldsymbol{x}^{(t)} \in \mathcal{X}, \ \forall t \in \{1, ..., T\}. \end{aligned} \tag{4.1}$$

4.2 The Fair Matching Over Time Problem

Imagine having a certain number of agents and the same number of tasks. When an agent performs one of the tasks, this will have some fixed cost. Searching for an assignment of all tasks to the agents such that each agent gets exactly one task and such that the total sum of the costs is minimized is called the assignment problem. However, when the agents get paid depending on the cost of their tasks or the tasks are funded by different parties it might be beneficial to choose a fair assignment. In [NBN22a] and [NBN22b] the balanced assignment problem is considered, in which we aim to find an assignment which minimizes the difference between the highest cost and the lowest cost associated with an agent-task pair.

Definition 4.2.1 (Balanced Assignment Problem)**.** *Let $G = (A \cup B, E)$ be a bipartite graph with $A \cap B = \emptyset$, $E \subseteq \{\{a, b\} : a \in A, b \in B\}$ and $|A| = |B|$. Furthermore, let $c : E \to \mathbb{R}$ be a cost function on the edges. Then the task is to find an assignment M, i.e. a perfect matching in G, which minimizes* $\max\{c(e) : e \in M\} - \min\{c(e) : e \in M\}$.

In [NBN22a] and [NBN22b] the article [Mar+84] is referenced, where an algorithm which solves the balanced assignment problem in $\mathcal{O}(n^4)$ time is described.

Furthermore, [NBN22a] and [NBN22b] show that in the case of non-negative weights it is possible to find all optimal solutions to the problem of finding a fair trade-off between the total cost of the assignment and the difference between the highest and lowest cost in polynomial time.

From now on, we will not consider the total cost of the assignment. We want to generalize the balanced assignment problem to arbitrary undirected graphs. The resulting problem looks as follows.

Definition 4.2.2 (Fair Matching Problem)**.** *Let $G = (V, E)$ be an undirected graph with weights $c : E \to \mathbb{R}$. The task is to find a perfect matching M which minimizes $\max\{c(e) : e \in M\} - \min\{c(e) : e \in M\}$ if such a matching exists.*

Definition 4.2.3 (Decision Version of the Fair Matching Problem)**.** *Let $G = (V, E)$ be an undirected graph with weights $c : E \to \mathbb{R}$ and k a constant. The question is whether there exists a perfect matching M with $\max\{c(e) : e \in M\} - \min\{c(e) : e \in M\} \leq k$.*

Theorem 4.2.4. *The decision version of the fair matching problem can be solved in $\mathcal{O}(\sqrt{n}m^2)$ time, where $|V| = n$ and $|E| = m$. In particular, the time complexity is $\mathcal{O}(n^{4.5})$ for dense graphs with $m = \mathcal{O}(m^2)$.*

Proof. We first sort the edges by their weight and rename them such that $c(e_1) \leq c(e_2) \leq ... \leq c(e_m)$. Then, we create a graph $H = (V, E')$, where E' is the set of all edges which have weight at most $c(e_1) + k$. Then we compute a maximum cardinality matching. If this is already a perfect matching we stop. Otherwise, we delete e_1 from the graph and add all edges with weight larger than $c(e_1) + k$ and at most $c(e_2) + k$ and search again for a perfect matching. This process is repeated until we have found a perfect matching, or until e_m was included in H, but no perfect matching was found in the new graph. Notice that throughout the whole algorithm, every edge is added and removed at most once to the graph H, as the edges are sorted. Thus, creating and maintaining the graph H takes at most $\mathcal{O}(m^2)$ time, as adding and removing edges can be done in at most $\mathcal{O}(m)$ for incidence lists, adjacency lists or adjacency matrices. Sorting the edges can be done in $\mathcal{O}(m \log m)$ time. The number of iterations is at most m and the maximum cardinality matching problem can be solve in $\mathcal{O}(\sqrt{n}m)$ time, see [Blu90]. This yields an overall running time of $\mathcal{O}(\sqrt{n}m^2)$. □

Now, we want to work towards an over-time version of the fair matching problem. Towards doing this, we first want to put this problem in the framework of a single round fairness problem defined in Section 4.1. In order to do so, we need to identify the utilities obtained by the stakeholders. Observe that for the balanced assignment problem we want the utilities to be fair among the agents and the parties funding the tasks. This means, that the utilities are actually received by the vertices and not by the edges. Consequently, we can define the utilities of the vertices to be the cost of the incident edges in the matching. This is well defined, as there is precisely one edge incident to every vertex in a perfect matching. For the framework of the fairness problems this yields the following. The set of options $\mathcal{X}$ corresponds to the set of perfect matchings in the given graph. We assume for simplicity that G contains a perfect matching. The utility function $u : \mathcal{X} \to \mathbb{R}^{|V(G)|}$ is given by $[u(M)]_v = \sum_{e \in M : v \in e} c(e)$ for all perfect matchings $M \in \mathcal{X}$ and all $v \in V$. And the unfairness function is given as $\Phi(y_1, ..., y_n) = \max(y_1, ..., y_n) - \min(y_1, ..., y_n)$. When we plug this into the single round fairness problem 4.1.2 we get precisely the fair matching problem 4.2.2.

We can now add the time component, which results in the fair matching over time problem.

Definition 4.2.5 (Fair Matching Over Time Problem)**.** *Let $G = (V, E)$ be an undirected graph containing a perfect matching with weights $c : E \to \mathbb{R}$ and $T \in \mathbb{Z}_{>0}$. The task is to find a sequence of perfect matchings $(M^{(1)}, M^{(2)}, ..., M^{(T)})$ that minimizes*

$$\max_{v \in V} \Big\{ \frac{1}{T} \sum_{t=1}^{T} u_v(M^{(t)}) \Big\} - \min_{v \in V} \Big\{ \frac{1}{T} \sum_{t=1}^{T} \sum_{e \in M^{(t)} : v \in e} c(e) \Big\}.$$

The fair matching over time problem with the undirected graph $G = (V, E)$ and weights $c : E \to \mathbb{R}$ can thus be formulated as follows.

$$\min_{(M^{(1)}, ..., M^{(T)})} \quad f - g \tag{4.2a}$$

$$\text{s.t.} \quad \sum_{e \,:\, v \in e} M_e^{(t)} = 1, \ \forall v \in V, \ \forall t \in \{1, ..., T\}, \tag{4.2b}$$

$$u_v^{(t)} = \sum_{e \,:\, v \in e} c(e) M_e^{(t)}, \ \forall v \in V, \ \forall t \in \{1, ..., T\}, \tag{4.2c}$$

$$y_v = \frac{1}{T} \sum_{t=1}^{T} u_v^{(t)}, \ \forall v \in V, \tag{4.2d}$$

$$f \geq y_v, \ \forall v \in V, \tag{4.2e}$$

$$g \leq y_v, \ \forall v \in V, \tag{4.2f}$$

$$M_e^{(t)} \in \{0, 1\} \tag{4.2g}$$

We overload the notation a bit and denote by $M^{(t)} = (M_e^{(t)})_{e \in E} \in \{0, 1\}^{|E|}$ the incidence vector of the matching $M^{(t)}$ for $t \in \{1, ..., T\}$. Constraints 4.2b together with $M_e^{(t)}$ being binary variables make sure that $M^{(t)}$ is a perfect matching for all t, Constraints (4.2c) compute $u_v(M^{(t)})$ and Constraints (4.2d) compute the average utility gained by the vertices. Constraints (4.2e) make sure that f is at least as large as the maximum of all y_v's and Constraints (4.2f) imply that g is at most as large as the minimum. As we minimize over $f - g$, this is equivalent to minimizing over $\max_{v \in V}\{y_v\} - \min_{v \in V}\{y_v\}$.

4.3 Solving the Fair Matching Over Time Problem Using Branch-and-Price

We build a branch-and-price algorithm which solves the fair matching over time problem.

We denote by $\mathcal{M}(G) \coloneqq \{M : M \text{ is a perfect matching in } G\}$ the set of all perfect matchings in G. We can reduce the symmetry in Problem (4.2a) by counting the number of occurrences of each $M \in \mathcal{M}(G)$ in the sequence $(M^{(1)}, M^{(2)}, ..., M^{(T)})$. This yields the following reformulation.

$$\min_{\boldsymbol{q}} \quad f - g \tag{4.3a}$$

$$\text{s.t.} \qquad y_v = \frac{1}{T} \sum_{j=1}^{k} q_j u_v^{(j)},\ \forall v \in V, \tag{4.3b}$$

$$\sum_{j=1}^{T} q_j = T, \tag{4.3c}$$

$$f \geq y_v,\ \forall v \in V, \tag{4.3d}$$

$$g \leq y_v,\ \forall v \in V, \tag{4.3e}$$

$$q_j \in \mathbb{Z}_{\geq 0},\ \forall j \in \{1, ..., k\}, \tag{4.3f}$$

if there are precisely k different perfect matchings $M_1, ..., M_k$ in G, i.e. $\mathcal{M}(G) = \{M_1, ..., M_k\}$ and $u_v^{(j)} = u_v(M_j)$ for j in $\{1, ..., k\}$. This means that the variable q_j counts the number of occurrences of the perfect matching M_j, $j \in \{1, ..., k\}$, in $(M^{(1)}, M^{(2)}, ..., M^{(T)})$.

The linear relaxation of Problem (4.3f) is the master problem at the root node of the branch-and-price tree. When we additionally take just a subset of the perfect matchings with indices $J' \subseteq \{1, .., k\}$, this yields a restricted master problem.

$$\min_{\boldsymbol{q}} \quad f - g \tag{4.4a}$$

$$\text{s.t.} \qquad y_v = \frac{1}{T} \sum_{j \in J'} q_j u_v^{(j)},\ \forall v \in V, \tag{4.4b}$$

$$\sum_{j=1}^{T} q_j = T, \tag{4.4c}$$

$$f \geq y_v,\ \forall v \in V, \tag{4.4d}$$

$$g \leq y_v,\ \forall v \in V, \tag{4.4e}$$

$$q_j \geq 0,\ \forall j \in J' \tag{4.4f}$$

The dual of the restricted master problem looks as follows.

$$\begin{aligned}
\max_{(\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\lambda},\mu)} \quad & \mu T && \text{(4.5a)}\\
\text{s.t.} \quad & \sum_{v\in V} \alpha_v = 1, && \text{(4.5b)}\\
& \sum_{v\in V} \beta_v = 1, && \text{(4.5c)}\\
& \lambda_v = \beta_v - \alpha_v,\ \forall v \in V, && \text{(4.5d)}\\
& \mu \leq -\frac{1}{T}\sum_{v\in V} \lambda_v u_v^{(j)},\ \forall j \in J', && \text{(4.5e)}\\
& \alpha_v \geq 0,\ \forall v \in V, && \text{(4.5f)}\\
& \beta_v \geq 0,\ \forall v \in V && \text{(4.5g)}
\end{aligned}$$

Here, we take $\{\alpha_v : v \in V\}$, $\{\beta_v : v \in V\}$, $\{\lambda_v : v \in V\}$ and μ as the dual variables corresponding to Constraints (4.4d), (4.4e), (4.4b) and (4.4c), respectively.

Let $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*, \boldsymbol{\lambda}^*, \mu^*)$ be an optimal solution of the dual of the restricted master problem. Then the reduced cost of a perfect matching M_j with $j \in \{1, .., k\}$ is

$$\begin{aligned}
c_j - \boldsymbol{u}^{*\intercal}\boldsymbol{a}_j &= 0 - (\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*, \boldsymbol{\lambda}^*, \mu^*)^\intercal(\mathbf{0}, \mathbf{0}, \frac{1}{T}u_{v_1}(M_j), ..., \frac{1}{T}u_{v_n}(M_j), 1)\\
&= -\frac{1}{T}\sum_{v\in V} \lambda_v^* u_v(M_j) - \mu^*,
\end{aligned} \tag{4.6}$$

where $V = \{v_1, ..., v_n\}$ and c_j, $\boldsymbol{a}_j$, $\boldsymbol{u}^*$ denote the cost coefficient, the column corresponding to the j-th variable of matching M_j in the master problem ,the optimal solution of the dual of the restricted master problem, respectively. This leads to the following pricing problem.

$$\min_{M \text{ perfect matching in } G} \quad -\frac{1}{T}\sum_{v\in V} \lambda_v^* u_v(M) - \mu^* \tag{4.7}$$

Notice that μ^* is a constant. Furthermore,

$$\sum_{v\in V} \lambda_v^* u_v(M) = \sum_{v\in V} \lambda_v^* \sum_{e\in M\,:\,v\in e} c(e) = \sum_{\{u,v\}\in M} c(\{u,v\})(\lambda_u^* + \lambda_v^*),$$

for every perfect matching M. Hence, the reduced cost of a perfect matching is the weight of the matching with respect to the weights $w(\{u,v\}) = c(\{u,v\})(-\lambda_u^* - \lambda_v^*)$ divided by T and reduced by μ^*. Therefore, we can get a matching with minimum reduced cost by solving a minimum weight perfect matching problem where the weights are defined as $w(\{u,v\}) = c(\{u,v\})(-\lambda_u^* - \lambda_v^*)$. Then we can get the minimum reduced cost by dividing the weight of this matching by T and subtracting μ^*. This solves the pricing problem.

After solving the current master problem, it can happen that the optimal solution contains a fractional variable $q^*_{j_0}$ for some j_0 in $\{1, ..., k\}$. In this case we will need to branch. For example, we can branch on the variable q_{j_0} and enforce $q_{j_0} \geq \lceil q^*_{j_0} \rceil$ in one child node and $q_{j_0} \leq \lfloor q^*_{j_0} \rfloor$ on the other child node. This can be done by adding these restrictions to the corresponding master problem to generate two new nodes of the branch-and-bound tree. We consider the node with $q_{j_0} \geq \lceil q^*_{j_0} \rceil$, the other node works similarly. The new constraint has a corresponding new dual variable σ and the dual restricted master problem in this node looks as follows.

$$\begin{aligned}
\max_{(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\lambda}, \mu, \sigma)} \quad & \mu T + \sigma \lceil q^*_{j_0} \rceil \\
\text{s.t.} \quad & \sum_{v \in V} \alpha_v = 1, \\
& \sum_{v \in V} \beta_v = 1, \\
& \lambda_v = \beta_v - \alpha_v, \\
& \mu \leq -\frac{1}{T} \sum_{v \in V} \lambda_v u_v^{(j)}, \; \forall j \in J' \setminus \{j_0\}, \\
& \mu + \sigma \leq -\frac{1}{T} \sum_{v \in V} \lambda_v u_v^{(j_0)}, \\
& \alpha_v \geq 0, \; \forall v \in V, \\
& \beta_v \geq 0, \; \forall v \in V, \\
& \sigma \geq 0
\end{aligned} \tag{4.8}$$

Let $(\boldsymbol{\alpha}', \boldsymbol{\beta}', \boldsymbol{\lambda}', \mu', \sigma')$ be an optimal dual solution for the dual problem (4.8). Then the pricing problem changes to the following.

$$\min_{M \text{ perfect matching in } G} \quad -\frac{1}{T} \sum_{v \in V} \lambda'_v u_v(M) - \mu' - \sigma' \mathbb{1}_{\{M = M_{j_0}\}}, \tag{4.9}$$

where $\mathbb{1}_{\{M = M_{j_0}\}}$ is the indicator function, which attains value 1 if $M = M_{j_0}$ and is 0 otherwise.

However, we do not want to add another column corresponding to the matching M_{j_0} to the restricted master problem. Thus, we search for a column with negative reduced cost among all other matchings.

$$\min_{M \text{ perfect matching in } G,\, M \neq M_{j_0}} \quad -\frac{1}{T} \sum_{v \in V} \lambda'_v u_v(M) - \mu' \tag{4.10}$$

In order to enforce that $M \neq M_{j_0}$, we can solve the minimum weight perfect matching problem and if M_{j_0} is the minimizer we compute the second best matching instead. When we are further down in the branch-and-bound tree we might need to compare the matching with all

the matchings we have branched on so far. In the worst case this might require computing the K-th best matching, where K is the depth of the search tree of the current node. In [CH87] it is shown that computing the K best different weighted perfect matchings can be done in $\mathcal{O}(n^3K)$ time, if K is a fixed constant. An easier to implement version runs in $\mathcal{O}(n^4K)$ time.

4.4 Implementation

The branch-and-price algorithm for the fair matching over time problem described above was implemented using the branch-and-price framework SCIP [Bol+24] with the interface PySCIPOpt [Mah+16] version $5.1.1$ from Python to SCIP version 9.1. The linear relaxations of the master problems were also solved using SCIP. For an example of how this branch-and-price framework can be used, see [Pri]. Furthermore, the Python package networkx [HSS08] version 3.3, which is a Python package for networks and graphs, was used to handle the weighted undirected graphs and to compute minimal weight matchings with a maximal number of edges. The K-th best matching algorithm was implemented as the $\mathcal{O}(n^4K)$ time algorithm in [CH87]. We now want to consider some computational experiments on this implementation. In the following part we present the results of the empirical analysis.

4.5 Computational Experiments

4.5.1 Two Very Small Examples

Firstly, we want to take a close look at a small illustrating example and, more precisely, to the solutions of the fair matching over time problem for these instances. Thus, we now consider specific instances of the fair matching over time problem. We consider the complete graph K_6 with weights as indicated in Figure 4.1. For the fair matching problem, we see that the matching which minimizes the unfairness yields an objective value of 1. When we consider the fair matching over time problem and vary the time horizon, Table 4.1 shows the optimal objective value and the number of different perfect matchings in an optimal solution. Notice that both the objective value and the number of different perfect matchings are not monotonically decreasing or increasing as T grows, but they are strongly correlated with T. However, observe that there is always an upper bound for the number of different matchings and thus the correlation between the number of different perfect matchings in an optimal solution and T is only present for reasonably large T. Furthermore, there might be a finite time horizon for which the optimal objective value among all time horizons can be reached, see [LSW23] and [LSW22]. Thus, also this correlation holds only for moderately large T.

Let us now consider in more depth an optimal solution of the fair matching over time problem with this graph and time horizon $T = 10$. The solution consists of the matchings in Figure 4.2, where the perfect matchings described in (a), (b) and (c) occur once and the perfect matching described in (d) is chosen 7 times. These perfect matchings yield the

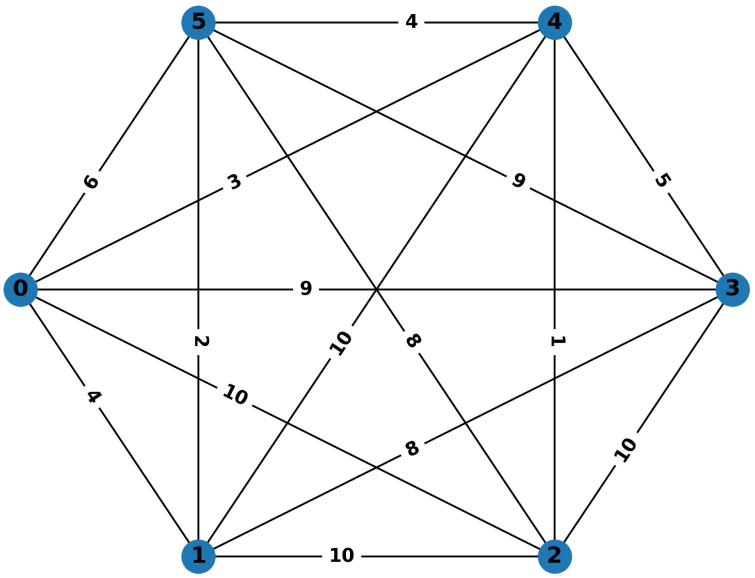


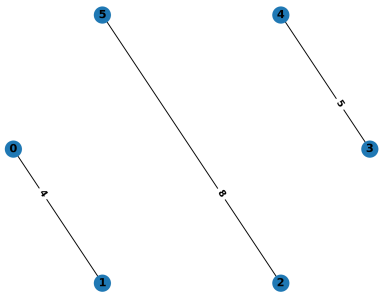
Figure 4.1: weighted graph K_6

Table 4.1: Objective value and number of different perfect matchings appearing in the optimal solution of the fair matching over time problem with instance including K_6, weights as in Figure 4.1 and $T = 1$ to $T = 15$.

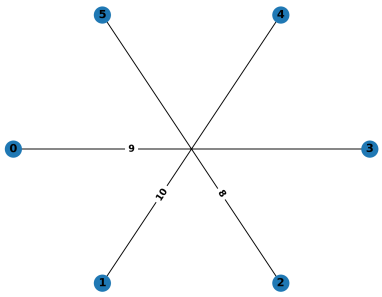
T	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
objective value	1	1	1	0.75	0.8	0.5	0.43	0.38	0.33	0.2	0.27	0.25	0.23	0.21	0.2
different matchings	1	1	2	3	3	2	2	2	4	4	4	5	4	4	4

following utilities for the vertices $(8.7, 8.8, 8.7, 8.6, 8.6, 8.8)$. Hence, the optimal objective value is 0.2. The weighted average weight of the matchings in the optimal solution is 26.1, i.e. when we sum the weight of each matching times the number of occurrences in the solution and divide by the time horizon, we get 26.1. For comparison, the matching of minimal weight in this graph has weight 12. Notice that there are $\frac{1}{k!}\frac{(2k)!}{2^{k+1}}$ perfect matchings in the complete graph K_{2k}. For $k = 3$ we have 15 perfect matchings in K_6.

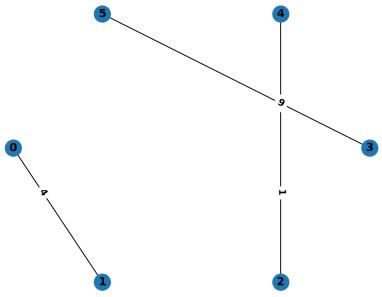
Next, we want to closely consider the solving process when the branch-and-price algorithm for the fair matching over time problem described above is used. More precisely, we want to look at the number of nodes of the branch-and-bound tree and the number of matchings generated during the branch-and-price process. To put this into perspective, we also look at the number of different matchings which are used in the optimal solution found and focus on



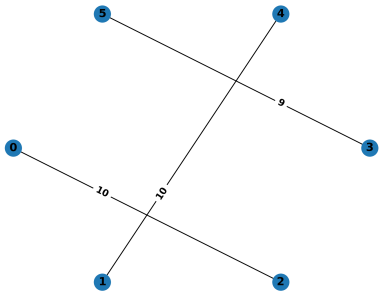
(a) This perfect matching occurs once.



(b) This perfect matching occurs once.



(c) This perfect matching occurs once.



(d) This perfect matching occurs seven times.

Figure 4.2: Perfect matchings of K_6 in the solution of the fair matching over time problem for $T = 10$.

graphs with a known number of perfect matchings. As mentioned above, the total number of perfect matchings in complete graphs can be easily calculated. This also holds for the complete bipartite graphs $K_{l,l}$, which has $l!$ perfect matchings. Also, we want the number of vertices to be such that most instances can be solved to optimality, while not being trivial. Therefore, we consider instances involving the complete graph K_6 and the complete bipartite graph $K_{4,4}$. We generate 10 instances of K_6 and $K_{4,4}$ with random weights which are i.i.d. uniformly distributed on $\{1, ..., 100\}$. The results of the branch-and-price algorithm applied to these instances are summarized in Table 4.2 and Table 4.3. In the first column, we enumerate the generated instances. In the second column the number of nodes of the branch-and-bound tree at the end of the branch-and-price procedure is listed. In the third column we count the number of different perfect matchings which occur in an optimal solution of the fair matching over time problem. In the fourth column we have the number of perfect matchings which were added to the master problem during the branch-and-price process. In the fifth column we note the status after the end of the branch-and-price procedure, where optimal means that the optimal solution was found within a time limit of 5 seconds. There, we can see that the number of perfect matchings generated for most of the K_6 instances is high compared to the total number of perfect matchings. For the $K_{4,4}$ instances the total number of generated perfect matchings is between 4 and 23, more precisely it is four times at most 8 and seven times at most 18. As the size of these instances is small, we can hope that for instances with more vertices, the total number of generated matchings would be even less compared to the total number of matchings. For all instances, the number of different matchings in the solution is way less than the total number of matchings. The number of nodes of the branch-and-bound tree differs quite a lot. However, we can see that the size of the branch-and-bound tree correlates with the number of generated matchings. This is as expected, because we only branch on the variables corresponding to generated matchings.

Table 4.2: Solving instances of the fair matching over time problem with graph K_6, random weights and $T = 10$ using the branch-and-price algorithm.

K_6	number of nodes in tree	number of matchings in solution	number of matchings generated	status
1	48	5	13	optimal
2	27	4	11	optimal
3	90	3	15	optimal
4	17	3	12	optimal
5	1	1	9	optimal
6	8	5	10	optimal
7	16	3	8	optimal
8	27	3	13	optimal
9	46	6	14	optimal
10	7	2	8	optimal

Table 4.3: Solving instances of the fair matching over time problem with graph $K_{4,4}$, random weights and $T = 10$ using the branch-and-price algorithm.

$K_{4,4}$	number of nodes in tree	number of matchings in solution	number of matchings generated	status
1	347	5	23	timelimit
2	4	2	6	optimal
3	3	6	8	optimal
4	195	6	21	optimal
5	155	4	16	optimal
6	251	5	18	optimal
7	174	6	14	optimal
8	1	3	4	optimal
9	409	5	23	optimal
10	1	1	7	optimal

4.5.2 Results on Random Graphs of Type G(n,p)

In this subsection, we consider the question, up to which size we can solve instances of the fair matching over time problem quickly by using our implementation of the branch-and-price algorithm. Towards this, we want to look at some different types of graphs and generate random instances of different sizes of the fair matching over time problem. We set a time limit of 5 seconds per instance. We generate 100 instances per parameter-setting of the random graphs and count the number of times the algorithm reaches the optimal solution within the time limit and the number of times that there exists a perfect matching in the graph. The instances which contain a perfect matching, but are not solved to optimality are not solved to optimality due to the time limit. We first look at binomial random graphs $G(n, p)$ for different numbers of vertices n and different probabilities p for an edge to be in the graph, see [FK15a]. As we also want to investigate how high we can set the time horizon, we try $T = 2, 10, 100, 1000, 10000$. The results can be seen in the Tables 4.4, 4.5, 4.6, 4.7 and 4.8. Observe that for dense graphs, the fairness over time problem is only solved to optimality by the branch-and-price algorithm within the time limit if the number of vertices is small. More precisely, for $n \leq 6$ the problem is almost always solved to optimality if $T \leq 100$, whereas for $T = 10000$ we solve to optimality 74 of 100 for $n = 6$ and $p = 1$ instances. When we have more sparse graphs, we can deal with many more vertices, probably as a consequence of the total number of edges decreasing and hence likely the total number of perfect matchings in the graph decreasing. Furthermore, the higher the time horizon, the more difficult the problem gets. However, we can clearly see that the number of vertices and the density of the graph have a greater influence on the number of solved instances within the time limit than the time horizon. In more detail, the dependency on T of the ratio of instances solved to optimality in 5 seconds to instances containing a perfect matching can be seen in Figures 4.3, 4.4 and 4.5 for random weights and the graphs $G(6, 1)$, $G(8, 0.6)$ and $G(20, 0.2)$, respectively. In the experiments we let T take the values in $\{2\} \cup \{10, 20, ..., 90\} \cup \{100, 200, ..., 900, 1000\}$. We can see a very weak dependency on

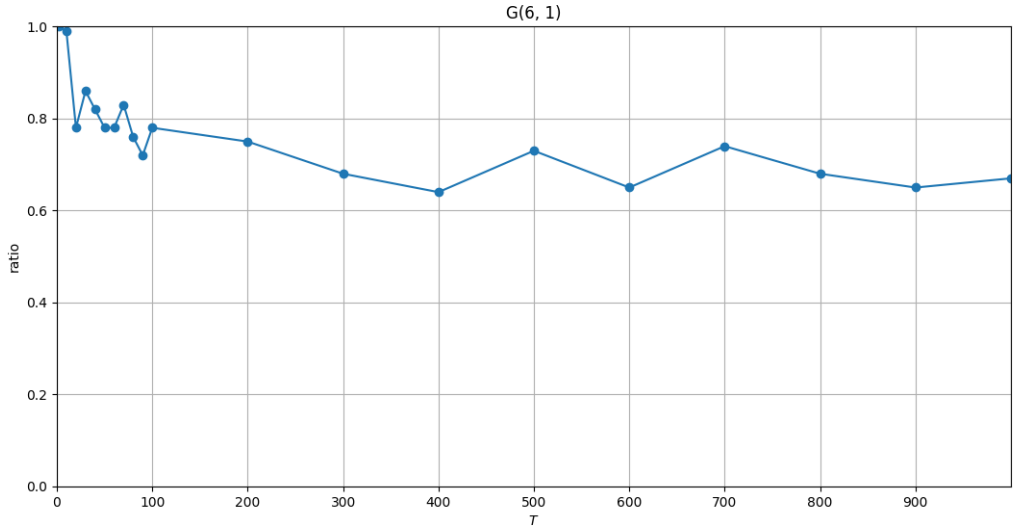


Figure 4.3: Ratio of instances solved to optimality in 5 seconds to instances containing a perfect matching out of 100 instances of the fair matching over time problem with graph $G(6,1)$, random weights and varying T, using the branch-and-price algorithm.

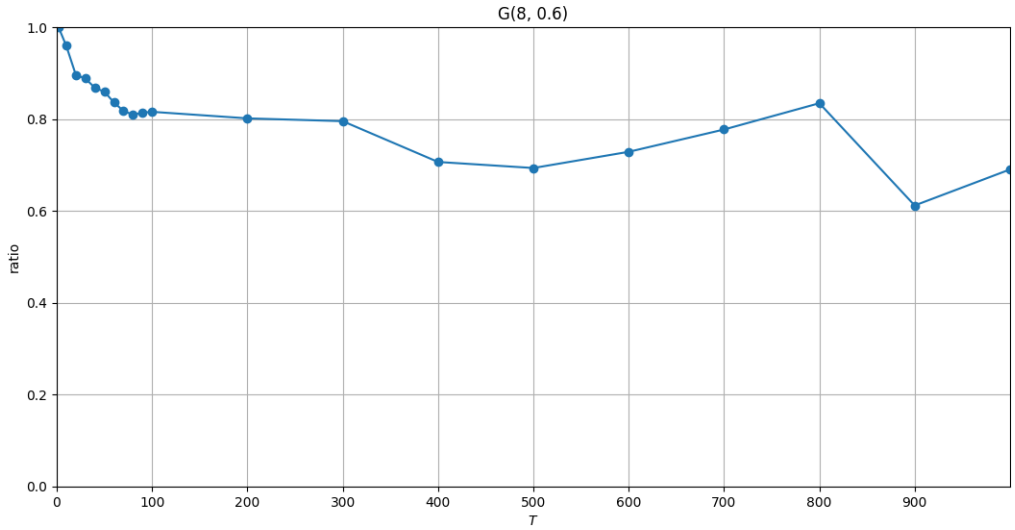
the time horizon. Owing to the fact that we chose the reformulation of the mixed integer linear problem in a way that reduces the symmetry introduced by adding the time horizon. The reformulation also leads to a high number of variables, coming from the large number of perfect matchings in a graph. To see how the algorithm performs when we set the time limit to 10 seconds, see Tables 4.9 and 4.10, where we again generated 100 instances for each parameter-setting of $G(n,p)$ with random weights and a time horizon of $T = 10$ and $T = 100$, respectively. In comparison to setting the time limit to 5 seconds, the algorithm typically solves some more instances whenever some, but not all instances are solved within a 5 seconds time limit. For instance, for $T = 10$ and $p = 0.6$ the ratio of instances solved to optimality to the instances containing a perfect matching is $1, 1, 0.81, 0.25, 0.04, 0$ when $n = 4, 6, 8, 10, 12, 14$, respectively. For the 10 seconds time limit we have a ratio of $1, 1, 0.93, 0.31, 0.04, 0.02$, for the same parameters.

4.5.3 Results on Random Bipartite Graphs B(n/2, n/2, p)

For all further investigations we choose our time horizon to be $T = 10$, generate random weights which are i.i.d. uniformly distributed on $\{1, ..., 100\}$, generate 100 instances per parameter-setting and set a time limit of 5 seconds for the branch-and-price algorithm.

In this subsection we will focus on bipartite graphs. In these graphs the pricing problem

Figure 4.4: Ratio of instances solved to optimality in 5 seconds to instances containing a perfect matching out of 100 instances of the fair matching over time problem with graph $G(8, 0.6)$, random weights and varying T, using the branch-and-price algorithm.



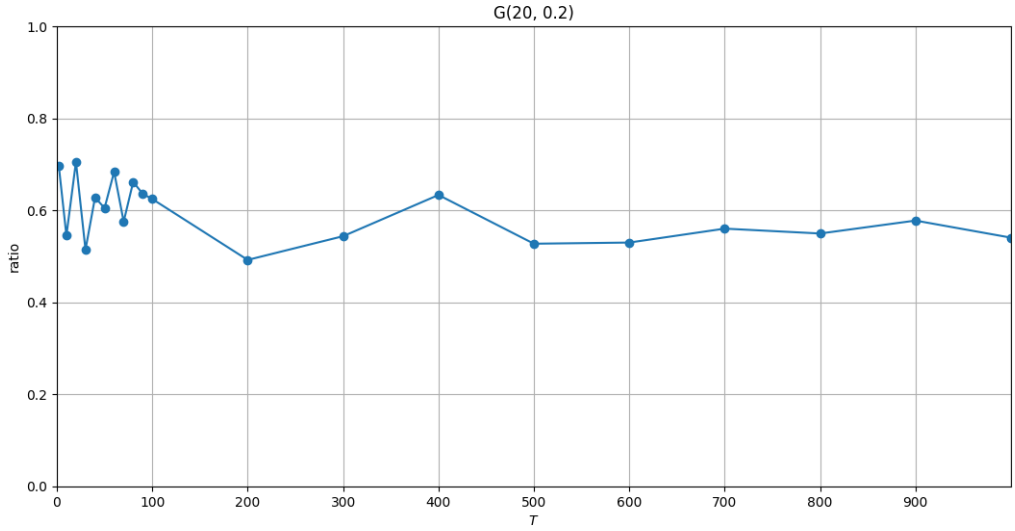


Figure 4.5: Ratio of instances solved to optimality in 5 seconds to instances containing a perfect matching out of 100 instances of the fair matching over time problem with graph $G(20, 0.2)$, random weights and varying T, using the branch-and-price algorithm.

is the assignment problem, which can be solved faster than the minimum weight perfect matching problem in general graphs. So we hope to be able to solve larger instances of the fair matching over time problem on bipartite graphs. For our next computational experiments, we generate bipartite binomial random graphs according to the $B(l, r, p)$ model, see [FK15b]. In order to have a perfect matching in a bipartite graph we need the left hand side and the right hand side of the graph to have equal sizes. Thus, we set $l = r = n/2$, where n is the even number of vertices. The ratio of instances which are solved optimally within 5 seconds to the instances which contain a perfect matching and the total number of instances which contain a perfect matching can be seen in Table 4.11. We can see that when we choose the same probability p as for the instances of $G(n, p)$, we can solve instances with a larger number of vertices. This may be caused by the smaller total number of edges and hence the likely smaller number of perfect matchings.

4.5.4 Results on Graphs With Small Maximum Degree and Small Average Degree

As the branch-and-price algorithm generates one matching after the other, the total number of matchings could be a good indicator for the running time of the algorithm. In [AF08a] and [AF08b] it is shown that $\prod_{i=1}^{n}(d_i!)^{\frac{1}{2d_i}}$ is an upper bound on the total number of perfect matchings in a graph with an even number n of vertices and degree sequence $(d_1, ..., d_n)$.

Motivated by this fact we want to investigate how our algorithm behaves for graphs with small maximal degrees and for graphs with a low expected average degree. More precisely, we generate random degree sequences, where the degrees are i.i.d. uniformly distributed on $\{2,3\}, \{2,3,4\}$ or $\{2,3,4,5\}$. Here, we exclude 1 as we want to have multiple possibilities to match every vertex to have a more interesting instance of the fair matching over time problem. Furthermore, we subtract 1 from the last degree if the sum of degrees is odd. If the resulting sequence cannot be achieved as a degree sequence of a graph, we generate a new sequence. Then, we generate a random graph with this degree sequence, as in [BKS09], using the networkx package. This yields a random graph with maximal degree $3, 4$ or 5, respectively. The performance of the branch-and-price algorithm on these graphs can be seen in Table 4.12. We can observe, that very few of the graphs do not have a perfect matching and even for $n = 100$ there are still 23 to 39 instances which were solved to optimality within 5 seconds. Thus, it seems that limiting the maximal degree does indeed help to greatly reduce the running time of the algorithm. Hence, this can be a good way to find larger instances of the fair matching over time problem which can be solved rather efficiently using the branch-and-price algorithm. For the graphs with expected average degree d, we again look at binomial random graphs $G(n,p)$, where $p = d/(n-1)$. When choosing $p = d/(n-1)$, the expected degree of a vertex u is $\mathbb{E}(\sum_{v \in V \setminus \{u\}} \mathbb{1}_{\{u,v\} \in E}) = \sum_{v \in V \setminus \{u\}} \mathbb{P}(\{u,v\} \in E) = (n-1)d/(n-1) = d$ and thus also the expected average degree is d. The according results can be seen in Table 4.13. Notice that there are again a lot of large instances which can be solved fast.

Table 4.4: Solving instances of the fair matching over time problem with graph $G(n,p)$, random weights and $T = 2$ using the branch-and-price algorithm. (ratio of instances solved to optimality in 5 seconds to instances containing a perfect matching, number of instances containing a perfect matching) out of 100 instances.

G(n,p)	p = 0.1	p = 0.2	p = 0.3	p = 0.4	p = 0.5	p = 0.6	p = 0.7	p = 0.8	p = 0.9	p = 1
n=4	(1, 2)	(1, 15)	(1, 27)	(1, 34)	(1, 71)	(1, 69)	(1, 88)	(1, 95)	(1, 99)	(1, 100)
n=6	(1, 1)	(1, 8)	(1, 25)	(1, 54)	(1, 74)	(1, 86)	(1, 99)	(1, 100)	(1, 100)	(1, 100)
n=8	(1, 3)	(1, 16)	(1, 38)	(1, 69)	(1, 92)	(1, 98)	(0.98, 99)	(0.87, 100)	(0.65, 100)	(0.28, 100)
n=10	(0, 0)	(1, 20)	(1, 58)	(0.99, 81)	(0.88, 93)	(0.48, 99)	(0.15, 100)	(0.02, 100)	(0, 100)	(0, 100)
n=12	(1, 4)	(1, 25)	(0.96, 77)	(0.76, 93)	(0.31, 99)	(0.08, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)
n=14	(1, 2)	(0.96, 28)	(0.84, 85)	(0.28, 98)	(0.06, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)
n=16	(0, 0)	(0.98, 40)	(0.55, 86)	(0.04, 100)	(0.01, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)
n=18	(1, 2)	(0.79, 58)	(0.26, 95)	(0.02, 100)	(0.01, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)
n=20	(1, 1)	(0.76, 67)	(0.1, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)

Table 4.5: Solving instances of the fair matching over time problem with graph $G(n,p)$, random weights and $T = 10$ using the branch-and-price algorithm. (ratio of instances solved to optimality in 5 seconds to instances containing a perfect matching, number of instances containing a perfect matching) out of 100 instances.

G(n,p)	p = 0.1	p = 0.2	p = 0.3	p = 0.4	p = 0.5	p = 0.6	p = 0.7	p = 0.8	p = 0.9	p = 1
n=4	(1, 1)	(1, 11)	(1, 21)	(1, 43)	(1, 54)	(1, 74)	(1, 88)	(1, 95)	(1, 100)	(1, 100)
n=6	(0, 0)	(1, 6)	(1, 36)	(1, 55)	(1, 73)	(1, 86)	(1, 98)	(1, 100)	(0.99, 100)	(0.97, 100)
n=8	(1, 2)	(1, 9)	(1, 42)	(1, 74)	(0.98, 94)	(0.81, 95)	(0.66, 99)	(0.44, 100)	(0.13, 100)	(0.02, 100)
n=10	(1, 1)	(1, 18)	(1, 53)	(0.92, 85)	(0.68, 97)	(0.25, 100)	(0.08, 100)	(0, 100)	(0, 100)	(0, 100)
n=12	(0, 0)	(1, 22)	(0.91, 70)	(0.61, 96)	(0.15, 100)	(0.04, 100)	(0.01, 100)	(0, 100)	(0, 100)	(0, 100)
n=14	(1, 1)	(1, 41)	(0.67, 80)	(0.18, 99)	(0.01, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)
n=16	(0, 0)	(0.91, 47)	(0.39, 96)	(0.12, 99)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)
n=18	(1, 1)	(0.78, 64)	(0.21, 94)	(0.05, 99)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)
n=20	(1, 2)	(0.59, 68)	(0.15, 98)	(0.03, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)

Table 4.6: Solving instances of the fair matching over time problem with graph $G(n,p)$, random weights and $T = 100$ using the branch-and-price algorithm. (ratio of instances solved to optimality in 5 seconds to instances containing a perfect matching, number of instances containing a perfect matching) out of 100 instances.

G(n,p)	p = 0.1	p = 0.2	p = 0.3	p = 0.4	p = 0.5	p = 0.6	p = 0.7	p = 0.8	p = 0.9	p = 1
n=4	(1, 3)	(1, 13)	(1, 18)	(1, 42)	(1, 56)	(1, 68)	(1, 85)	(1, 92)	(1, 100)	(1, 100)
n=6	(0, 0)	(1, 7)	(1, 26)	(1, 47)	(1, 74)	(1, 91)	(0.96, 96)	(0.95, 99)	(0.87, 100)	(0.7, 100)
n=8	(0, 0)	(1, 12)	(1, 45)	(1, 77)	(0.93, 88)	(0.64, 98)	(0.53, 100)	(0.28, 100)	(0.07, 100)	(0.01, 100)
n=10	(1, 1)	(1, 18)	(1, 62)	(0.76, 82)	(0.47, 94)	(0.20, 99)	(0.06, 100)	(0, 100)	(0, 100)	(0, 100)
n=12	(0, 0)	(1, 27)	(0.85, 74)	(0.48, 98)	(0.1, 100)	(0.03, 100)	(0, 100)	(0.01, 100)	(0, 100)	(0, 100)
n=14	(0, 0)	(0.93, 30)	(0.53, 80)	(0.18, 96)	(0.04, 100)	(0.01, 100)	(0.01, 100)	(0, 100)	(0, 100)	(0, 100)
n=16	(1, 1)	(0.87, 38)	(0.26, 87)	(0.13, 98)	(0.02, 100)	(0.03, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)
n=18	(1, 2)	(0.66, 53)	(0.26, 94)	(0.11, 99)	(0.01, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)
n=20	(1, 1)	(0.61, 69)	(0.23, 96)	(0.05, 100)	(0.01, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)

Table 4.7: Solving instances of the fair matching over time problem with graph $G(n, p)$, random weights and $T = 1000$ using the branch-and-price algorithm. (ratio of instances solved to optimality in 5 seconds to instances containing a perfect matching, number of instances containing a perfect matching) out of 100 instances.

G(n,p)	p = 0.1	p = 0.2	p = 0.3	p = 0.4	p = 0.5	p = 0.6	p = 0.7	p = 0.8	p = 0.9	p = 1
n=4	(1, 2)	(1, 15)	(1, 27)	(1, 37)	(1, 61)	(1, 75)	(1, 89)	(1, 96)	(1, 98)	(1, 100)
n=6	(1, 2)	(1, 13)	(1, 27)	(1, 44)	(1, 72)	(1, 88)	(0.96, 99)	(0.88, 97)	(0.78, 100)	(0.6, 100)
n=8	(1, 1)	(1, 11)	(0.98, 43)	(0.97, 72)	(0.86, 90)	(0.65, 96)	(0.42, 100)	(0.17, 100)	(0.04, 100)	(0, 100)
n=10	(0, 0)	(1, 13)	(0.93, 58)	(0.74, 92)	(0.45, 98)	(0.20, 99)	(0.05, 100)	(0, 100)	(0, 100)	(0, 100)
n=12	(1, 1)	(1, 24)	(0.72, 67)	(0.40, 94)	(0.10, 99)	(0.01, 100)	(0.02, 100)	(0.01, 100)	(0, 100)	(0, 100)
n=14	(1, 1)	(0.86, 37)	(0.49, 85)	(0.22, 98)	(0.1, 100)	(0.02, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)
n=16	(1, 2)	(0.72, 39)	(0.32, 94)	(0.12, 98)	(0.03, 100)	(0.01, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)
n=18	(1, 1)	(0.65, 52)	(0.35, 99)	(0.12, 100)	(0.04, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)
n=20	(1, 2)	(0.46, 67)	(0.24, 99)	(0.1, 100)	(0.03, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)

Table 4.8: Solving instances of the fair matching over time problem with graph $G(n,p)$, random weights and $T = 10000$ using the branch-and-price algorithm. (ratio of instances solved to optimality in 5 seconds to instances containing a perfect matching, number of instances containing a perfect matching) out of 100 instances.

G(n,p)	p = 0.1	p = 0.2	p = 0.3	p = 0.4	p = 0.5	p = 0.6	p = 0.7	p = 0.8	p = 0.9	p = 1
n=4	(1, 4)	(1, 8)	(1, 29)	(1, 40)	(1, 63)	(1, 78)	(1, 86)	(1, 94)	(1, 99)	(1, 100)
n=6	(1, 4)	(1, 10)	(1, 26)	(1, 50)	(1, 74)	(1, 94)	(0.96, 98)	(0.91, 100)	(0.82, 100)	(0.74, 100)
n=8	(1, 1)	(1, 11)	(1, 29)	(0.96, 69)	(0.94, 90)	(0.72, 95)	(0.51, 100)	(0.25, 100)	(0.08, 100)	(0.02, 100)
n=10	(1, 2)	(1, 18)	(0.9, 50)	(0.78, 85)	(0.46, 98)	(0.28, 100)	(0.1, 100)	(0.04, 100)	(0, 100)	(0.01, 100)
n=12	(1, 1)	(0.95, 20)	(0.81, 75)	(0.5, 98)	(0.16, 100)	(0.09, 100)	(0.05, 100)	(0.02, 100)	(0, 100)	(0, 100)
n=16	(0, 0)	(0.95, 22)	(0.64, 86)	(0.27, 100)	(0.18, 100)	(0.09, 100)	(0.01, 100)	(0, 100)	(0, 100)	(0, 100)
n=16	(1, 1)	(0.76, 38)	(0.46, 89)	(0.21, 99)	(0.08, 100)	(0.03, 100)	(0.02, 100)	(0, 100)	(0, 100)	(0, 100)
n=18	(1, 2)	(0.58, 55)	(0.38, 90)	(0.11, 100)	(0.06, 100)	(0.01, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)
n=20	(1, 1)	(0.61, 67)	(0.32, 98)	(0.15, 99)	(0.03, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 99)	(0, 100)

Table 4.9: Solving instances of the fair matching over time problem with graph $G(n,p)$, random weights and $T = 10$ using the branch-and-price algorithm. (ratio of instances solved to optimality in 10 seconds to instances containing a perfect matching, number of instances containing a perfect matching) out of 100 instances.

G(n,p)	p = 0.1	p = 0.2	p = 0.3	p = 0.4	p = 0.5	p = 0.6	p = 0.7	p = 0.8	p = 0.9	p = 1
n=4	(1.00, 2)	(1.00, 13)	(1.00, 19)	(1.00, 44)	(1.00, 52)	(1.00, 77)	(1.00, 80)	(1.00, 97)	(1.00, 99)	(1.00, 100)
n=6	(1.00, 1)	(1.00, 9)	(1.00, 23)	(1.00, 54)	(1.00, 69)	(1.00, 91)	(1.00, 97)	(1.00, 100)	(1.00, 100)	(0.99, 100)
n=8	(0, 0)	(1.00, 13)	(1.00, 50)	(1.00, 63)	(0.99, 89)	(0.93, 98)	(0.72, 100)	(0.55, 100)	(0.22, 100)	(0.07, 100)
n=10	(1.00, 2)	(1.00, 13)	(1.00, 46)	(0.89, 83)	(0.64, 97)	(0.31, 100)	(0.09, 100)	(0.01, 100)	(0.00, 100)	(0.00, 100)
n=12	(1.00, 1)	(1.00, 21)	(0.96, 72)	(0.65, 94)	(0.22, 99)	(0.04, 99)	(0.01, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)
n=14	(0, 0)	(1.00, 29)	(0.67, 85)	(0.28, 98)	(0.03, 100)	(0.02, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)
n=16	(0, 0)	(0.91, 44)	(0.38, 89)	(0.09, 100)	(0.01, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)
n=18	(0, 0)	(0.69, 59)	(0.22, 99)	(0.04, 100)	(0.01, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)
n=20	(1.00, 4)	(0.61, 59)	(0.18, 98)	(0.02, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)

Table 4.10: Solving instances of the fair matching over time problem with graph $G(n,p)$, random weights and $T = 100$ using the branch-and-price algorithm. (ratio of instances solved to optimality in 10 seconds to instances containing a perfect matching, number of instances containing a perfect matching) out of 100 instances.

G(n,p)	p = 0.1	p = 0.2	p = 0.3	p = 0.4	p = 0.5	p = 0.6	p = 0.7	p = 0.8	p = 0.9	p = 1
n=4	(1.00, 4)	(1.00, 5)	(1.00, 24)	(1.00, 41)	(1.00, 56)	(1.00, 75)	(1.00, 81)	(1.00, 98)	(1.00, 100)	(1.00, 100)
n=6	(0, 0)	(1.00, 17)	(1.00, 29)	(1.00, 59)	(1.00, 77)	(1.00, 93)	(1.00, 96)	(0.96, 100)	(0.85, 100)	(0.79, 100)
n=8	(1.00, 2)	(1.00, 10)	(1.00, 42)	(1.00, 64)	(0.92, 92)	(0.78, 98)	(0.52, 100)	(0.29, 100)	(0.10, 100)	(0.02, 100)
n=10	(1.00, 1)	(1.00, 17)	(0.98, 48)	(0.85, 84)	(0.53, 94)	(0.23, 100)	(0.04, 99)	(0.02, 100)	(0.01, 100)	(0.00, 100)
n=12	(1.00, 1)	(1.00, 18)	(0.84, 74)	(0.39, 96)	(0.17, 100)	(0.01, 100)	(0.04, 100)	(0.01, 100)	(0.00, 100)	(0.01, 100)
n=14	(1.00, 2)	(0.94, 31)	(0.51, 83)	(0.25, 96)	(0.03, 100)	(0.03, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)
n=16	(0, 0)	(0.84, 55)	(0.36, 92)	(0.09, 100)	(0.04, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)
n=18	(1.00, 3)	(0.73, 52)	(0.29, 98)	(0.13, 100)	(0.02, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)
n=20	(1.00, 3)	(0.44, 72)	(0.27, 96)	(0.07, 98)	(0.00, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)	(0.00, 100)

Table 4.11: Solving instances of the fair matching over time problem with graph $B(n/2, n/2, p)$, random weights and $T = 10$ using the branch-and-price algorithm. (ratio of instances solved to optimality in 5 seconds to instances containing a perfect matching, number of instances containing a perfect matching) out of 100 instances.

B	p = 0.1	p = 0.2	p = 0.3	p = 0.4	p = 0.5	p = 0.6	p = 0.7	p = 0.8	p = 0.9	p = 1
n=4	(1, 3)	(1, 6)	(1, 23)	(1, 33)	(1, 41)	(1, 52)	(1, 68)	(1, 87)	(1, 96)	(1, 100)
n=6	(1, 1)	(1, 6)	(1, 22)	(1, 29)	(1, 52)	(1, 65)	(1, 91)	(1, 93)	(1, 98)	(1, 100)
n=8	(1, 1)	(1, 3)	(1, 12)	(1, 40)	(1, 68)	(1, 84)	(0.99, 98)	(0.99, 100)	(0.92, 100)	(0.72, 100)
n=10	(0, 0)	(1, 1)	(1, 19)	(1, 43)	(0.95, 75)	(0.94, 89)	(0.69, 98)	(0.53, 100)	(0.24, 100)	(0.11, 100)
n=12	(0, 0)	(1, 2)	(1, 12)	(0.98, 46)	(0.84, 83)	(0.59, 98)	(0.28, 99)	(0.05, 100)	(0.06, 100)	(0, 100)
n=14	(0, 0)	(1, 3)	(0.92, 25)	(0.87, 55)	(0.59, 95)	(0.26, 100)	(0.15, 100)	(0.05, 100)	(0.01, 100)	(0, 100)
n=16	(0, 0)	(1, 3)	(0.94, 32)	(0.70, 67)	(0.41, 95)	(0.13, 99)	(0.1, 100)	(0.03, 100)	(0, 100)	(0, 100)
n=18	(0, 0)	(1, 1)	(0.83, 35)	(0.55, 77)	(0.26, 94)	(0.05, 99)	(0.06, 100)	(0.01, 100)	(0, 100)	(0, 100)
n=20	(0, 0)	(0.89, 9)	(0.71, 59)	(0.43, 86)	(0.21, 97)	(0.06, 100)	(0.01, 100)	(0.01, 100)	(0.01, 100)	(0, 100)
n=30	(0, 0)	(0.81, 21)	(0.37, 91)	(0.21, 99)	(0.01, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)
n=40	(1, 1)	(0.67, 55)	(0.20, 96)	(0.05, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)
n=50	(1, 2)	(0.52, 82)	(0.07, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)	(0, 100)

Table 4.12: Solving instances of the fair matching over time problem with a graph with random degree sequence with maximal degree of $3, 4$, or 5, random weights and $T = 10$ using the branch-and-price algorithm. (ratio of instances solved to optimality in 5 seconds to instances containing a perfect matching, number of instances containing a perfect matching) out of 100 instances.

	max degree $= 3$	max degree $= 4$	max degree $= 5$
n=10	(1, 100)	(0.99, 99)	(0.95, 100)
n=20	(0.94, 97)	(0.67, 99)	(0.42, 100)
n=30	(0.69, 97)	(0.31, 98)	(0.27, 100)
n=40	(0.54, 99)	(0.37, 99)	(0.29, 100)
n=50	(0.44, 95)	(0.4, 100)	(0.28, 100)
n=60	(0.38, 98)	(0.3, 100)	(0.28, 100)
n=70	(0.37, 99)	(0.25, 100)	(0.37, 99)
n=80	(0.35, 97)	(0.23, 100)	(0.32, 100)
n=90	(0.33, 100)	(0.39, 99)	(0.37, 100)
n=100	(0.24, 96)	(0.40, 98)	(0.33, 100)

Table 4.13: Solving instances of the fair matching over time problem with graph G(n, d/(n-1)) with expected average degree of d, random weights and $T = 10$ using the branch-and-price algorithm. (ratio of instances solved to optimality in 5 seconds to instances containing a perfect matching, number of instances containing a perfect matching) out of 100 instances.

G_d	d = 2	d = 2.5	d = 3	d = 3.5	d = 4	d = 4.5	d = 5	d = 5.5	d=6	d=6.5	d=7
n=10	(1.00, 30)	(0.98, 41)	(0.99, 67)	(0.89, 82)	(0.82, 90)	(0.70, 99)	(0.47, 98)	(0.29, 100)	(0.06, 100)	(0.00, 100)	(0.00, 100)
n=20	(1.00, 1)	(0.94, 16)	(0.81, 27)	(0.68, 44)	(0.51, 76)	(0.44, 94)	(0.24, 92)	(0.19, 100)	(0.11, 99)	(0.04, 100)	(0.02, 100)
n=30	(1.00, 1)	(0.80, 5)	(0.79, 14)	(0.79, 28)	(0.55, 60)	(0.46, 76)	(0.30, 88)	(0.25, 92)	(0.09, 96)	(0.04, 100)	(0.03, 100)
n=40	(0, 0)	(1.00, 2)	(0.82, 11)	(0.60, 20)	(0.68, 44)	(0.42, 71)	(0.44, 80)	(0.33, 91)	(0.14, 94)	(0.06, 98)	(0.08, 100)
n=50	(0, 0)	(1.00, 1)	(1.00, 2)	(0.70, 23)	(0.62, 45)	(0.59, 70)	(0.56, 78)	(0.36, 91)	(0.16, 95)	(0.13, 100)	(0.07, 98)
n=60	(0, 0)	(0, 0)	(1.00, 4)	(0.67, 6)	(0.61, 31)	(0.64, 55)	(0.59, 71)	(0.43, 86)	(0.19, 91)	(0.07, 100)	(0.07, 98)
n=70	(0, 0)	(0, 0)	(0.00, 2)	(0.71, 7)	(0.86, 22)	(0.62, 40)	(0.59, 56)	(0.36, 70)	(0.18, 95)	(0.02, 96)	(0.00, 99)
n=80	(0, 0)	(0, 0)	(1.00, 1)	(1.00, 8)	(0.62, 21)	(0.70, 44)	(0.52, 61)	(0.47, 75)	(0.18, 94)	(0.06, 96)	(0.05, 100)
n=90	(0, 0)	(0, 0)	(0, 0)	(1.00, 3)	(0.62, 13)	(0.74, 34)	(0.58, 60)	(0.44, 70)	(0.17, 90)	(0.03, 97)	(0.05, 98)
n=100	(0, 0)	(0, 0)	(0, 0)	(1.00, 2)	(0.80, 15)	(0.48, 27)	(0.54, 50)	(0.30, 64)	(0.17, 89)	(0.10, 94)	(0.04, 98)

5 Conclusion

We discussed column generation and branch-and-price approaches which are very powerful tools to solve large linear programs and mixed integer linear programs. Both tools are generic, but there are numerous possibilities to adapt them to the given problem and enhance the algorithmic performance. First of all, choosing the right reformulation of the problem is a crucial factor. For this, Dantzig-Wolfe decompositions can often be used. Furthermore, it is indispensable to have a pricing problem which can be solved rather efficiently. In the context of column generation it is essential to handle the tailing-off effect properly using an appropriate stabilization technique. For branch-and-price we saw that selecting a suitable branching strategy can further improve the algorithm.

Based on the techniques mentioned above we derived a branch-and-price algorithm for the fair matching over time problem. The fair matching over time problem is a generalization of the balanced assignment problem, where we consider general graphs and add a time horizon. We introduced a particular formulation of the fair matching over time problem, which successfully deals with the symmetry introduced by the time horizon. Another advantage and important feature of this formulation is that the pricing problem can be solved by a minimum weight perfect matching problem. We implemented our algorithm using the branch-and-price framework SCIP with the interface PySCIPOpt from Python. We tested its performance on randomly generated instances of the fair matching over time problem. For the instances, we experiment with different random graphs, including bipartite graphs and graphs with low degrees and generate the weights randomly. We tested different orders of graphs with various densities, time horizons and time limits for the solving process. We observed that the branch-and-price algorithm can solve instances with very large time horizons quickly. However, it seems that the algorithm does not scale well with the total number of perfect matchings in the graph. The latter seems to be a crucial bottleneck for our algorithm. In particular, our algorithm is able to solve large instances of the fair matching over time problem on graphs with a limited maximal degree or average degree of the vertices. Clearly, such graphs contain a moderate number of perfect matchings in general.

In the following we briefly mention some open questions for further research. First, it would be desirable to better understand the influence of different components of the algorithm to its performance. In particular, we assume that reformulations of the problem, stabilization methods and branching strategies do have a crucial impact on the performance of the algorithm. Second, the performance of a branch-and-price algorithm for the fair matching over time problem with different unfairness functions or different ways to aggregate the utilities over time can be explored. The dependency of the optimal objective value on the time horizon can be further investigated, for instance by considering a probabilistic formulation

as in [LSW23] and [LSW22] to observe the behavior for large T. The overall welfare can be considered in the fair matching over time problem by allowing only perfect matchings which exceed a certain weight. Moreover, it can be investigated whether adding cuts to the branch-and-price algorithm improves its performance. As for the usage of branch-and-price, one could presumably also set up a branch-and-price algorithm for fairness over time problems other than the fair matching over time problem and the ambulance relocation problem in [Lod+22] in a similar manner and get a decently efficient algorithm provided that the pricing problem can still be solved rather fast.

All in all, it can be said that column generation and branch-and-price are very powerful tools which work well for solving the fair matching problem over time.

Bibliography

[AB07] T. Achterberg and T. Berthold. "Improving the Feasibility Pump". In: *Discrete Optimization* 4 (Mar. 2007), pp. 77–86. DOI: 10.1016/j.disopt.2006.10.004.

[Ach07] T. Achterberg. "Constraint Integer Programming". PhD thesis. Technische Universität Berlin, Jan. 2007. DOI: 10.14279/depositonce-1634. URL: https://www.researchgate.net/publication/230595676_Constraint_Integer_Programming.

[AF08a] N. Alon and S. Friedland. *The Maximum Number of Perfect Matchings in Graphs with a Given Degree Sequence*. 2008. arXiv: 0803.2578 [math.CO]. URL: https://arxiv.org/abs/0803.2578.

[AF08b] N. Alon and S. Friedland. "The Maximum Number of Perfect Matchings in Graphs with a Given Degree Sequence". In: *Electronic Journal Combinatorics* 15 (Apr. 2008). DOI: 10.37236/888.

[BAC05] H. Ben Amor and J. Valério de Carvalho. "Cutting Stock Problems". In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M. M. Solomon. Boston, MA: Springer US, 2005, pp. 131–161. ISBN: 978-0-387-25486-9. DOI: 10.1007/0-387-25486-2_5.

[BADF04] H. Ben Amor, J. Desrosiers, and A. Frangioni. *Stabilization in Column Generation*. Les Cahiers du GERAD G-2004-62. GERAD, Montréal QC H3T 2A7, Canada: Groupe d'études et de recherche en analyse des décisions, Aug. 2004, pp. 1–31. eprint: https://www.gerad.ca/papers/G-2004-62.pdf?locale=en. URL: https://www.gerad.ca/en/papers/G-2004-62. published.

[Ber06] T. Berthold. "Primal Heuristics for Mixed Integer Programs". Diplomarbeit. Technische Universität Berlin, 2006. URL: https://www.researchgate.net/publication/258846101_Primal_Heuristics_for_Mixed_Integer_Programs.

[Ber07] T. Berthold. "RENS - Relaxation Enforced Neighborhood Search". In: 2007. URL: https://api.semanticscholar.org/CorpusID:56072527.

[Ber08] T. Berthold. "Heuristics of the Branch-Cut-and-Price-Framework SCIP". In: *Operations Research Proceedings 2007*. Ed. by J. Kalcsics and S. Nickel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 31–36. ISBN: 978-3-540-77903-2.

[BFL07] L. Bertacco, M. Fischetti, and A. Lodi. "A Feasibility Pump Heuristic for General Mixed-Integer Problems". In: *Discrete Optimization* 4 (Mar. 2007), pp. 63–76. DOI: 10.1016/j.disopt.2006.10.001.

[BFT11] D. Bertsimas, V. Farias, and N. Trichakis. "The Price of Fairness". In: *Operations Research* 59 (Feb. 2011), pp. 17–31. DOI: 10.1287/opre.1100.0865.

[BJ98] F. Barahona and D. Jensen. "Plant location with minimum inventory". In: *Mathematical Programming* 83 (Sept. 1998), pp. 101–111. DOI: 10.1007/BF02680552.

[BKS09] M. Bayati, J. H. Kim, and A. Saberi. "A Sequential Algorithm for Generating Random Graphs". In: *Algorithmica* 58 (July 2009), 860–910. ISSN: 1432-0541. DOI: 10.1007/s00453-009-9340-1.

[Blu90] N. Blum. "A New Approach to Maximum Matching in General Graphs". In: *Automata, Languages and Programming*. Ed. by M. S. Paterson. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 586–597. ISBN: 978-3-540-47159-2.

[BLW18] M. Bastubbe, M. E. Lübbecke, and J. T. Witt. "A Computational Investigation on the Strength of Dantzig-Wolfe Reformulations". In: *17th International Symposium on Experimental Algorithms (SEA 2018)*. Ed. by Gianlorenzo D'Angelo. Vol. 103. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018, 11:1–11:12. ISBN: 978-3-95977-070-5. DOI: 10.4230/LIPIcs.SEA.2018.11. URL: http://dagstuhl.sunsite.rwth-aachen.de/volltexte/2018/8946/pdf/LIPIcs-SEA-2018-11.pdf.

[Bol+24] S. Bolusani, M. Besançon, K. Bestuzheva, A. Chmiela, J. Dionísio, T. Donkiewicz, J. van Doornmalen, L. Eifler, M. Ghannam, A. Gleixner, C. Graczyk, K. Halbig, I. Hedtke, A. Hoen, C. Hojny, R. van der Hulst, D. Kamp, T. Koch, K. Kofler, J. Lentz, J. Manns, Mexi G., E. Mühmer, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, M. Turner, S. Vigerske, D. Weninger, and L. Xu. *The SCIP Optimization Suite 9.0*. Technical Report. Optimization Online, 2024. URL: https://optimization-online.org/2024/02/the-scip-optimization-suite-9-0/.

[Bri+05] O. Briant, C. Lemaréchal, Ph. Meurdesoif, S. Michel, N. Perrot, and F. Vanderbeck. *Comparison of Bundle and Classical Column Generation*. Research Report RR-5453. An updated version of this paper has appeared in : Mathematical Programming, Ser. A, 2006 DOI 10.1007/s10107-006-0079-z. INRIA, 2005, p. 31. URL: https://inria.hal.science/inria-00070554.

[Bri+08] O. Briant, C. Lemaréchal, P. Meurdesoif, S. Michel, N. Perrot, and F. Vanderbeck. "Comparison of Bundle and Classical Column Generation". In: *Mathematical Programming* 113 (June 2008), pp. 299–344. DOI: 10.1007/s10107-006-0079-z.

[BS06] G. Belov and G. Scheithauer. "A Branch-and-Cut-and-Price Algorithm for One-Dimensional Stock Cutting and Two-Dimensional Two-Stage Cutting". In: *European Journal of Operational Research* 171 (2006), pp. 85–106. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2004.08.036. URL: https://www.sciencedirect.com/science/article/pii/S0377221704006150.

[BVS07] S. Boyd, L. Vandenberghe, and J. Skaf. "Analytic Center Cutting-Plane Method". In: 2007. URL: https://api.semanticscholar.org/CorpusID:5915628.

[CH87] C. R. Chegireddy and H. W. Hamacher. "Algorithms for finding K-best perfect matchings". In: *Discrete Applied Mathematics* 18 (1987), pp. 155–165. ISSN: 0166-218X. DOI: 10.1016/0166-218X(87)90017-5. URL: https://www.sciencedirect.com/science/article/pii/0166218X87900175.

[DDS11] G. Desaulniers, J. Desrosiers, and S. Spoorendonk. "Cutting Planes for Branch-and-Price Algorithms". In: *Networks* 58 (Dec. 2011), pp. 301–310. DOI: 10.1002/net.20471.

[Des+24] J. Desrosiers, M. E. Lübbecke, G. Desaulniers, and J.-B. Gauthier. *Branch-and-Price*. Les Cahiers du GERAD G-2024-36. GERAD, Montréal QC H3T 2A7, Canada: Groupe d'études et de recherche en analyse des décisions, June 2024, pp. 1–689. eprint: https://www.gerad.ca/papers/G-2024-36.pdf?locale=en. URL: https://www.gerad.ca/en/papers/G-2024-36. published.

[Des+95] J. Desrosiers, Y. Dumas, M. Solomon, and F. Soumis. "Time Constrained Routing and Scheduling". In: *Network Routing* (Jan. 1995), pp. 35–139.

[DJ07] Z. Degraeve and R. Jans. "A New Dantzig-Wolfe Reformulation and Branch-and-Price Algorithm for the Capacitated Lot Sizing Problem with Set Up Times". In: *Operations Research* 55 (Oct. 2007), pp. 909–920. DOI: 10.1287/opre.1070.0404.

[DL05] J. Desrosiers and M. E. Lübbecke. "A Primer in Column Generation". In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M. M. Solomon. Boston, MA: Springer US, 2005, pp. 1–32. ISBN: 978-0-387-25486-9. DOI: 10.1007/0-387-25486-2_1.

[DL11] J. Desrosiers and M. E. Lübbecke. "Branch-Price-and-Cut Algorithms". In: *Wiley Encyclopedia of Operations Research and Management Science* (Jan. 2011). DOI: 10.1002/9780470400531.eorms0118.

[DW60] G. B. Dantzig and P. Wolfe. "Decomposition Principle for Linear Programs". In: *Operations Research* 8 (1960), pp. 101–111. DOI: 10.1287/opre.8.1.101. eprint: https://doi.org/10.1287/opre.8.1.101.

[Elf+01] M. Elf, C. Gutwenger, M. Jünger, and G. Rinaldi. "Branch-and-Cut Algorithms for Combinatorial Optimization and Their Implementation in ABACUS". In: vol. 2241. Jan. 2001, pp. 157–222. ISBN: 978-3-540-42877-0. DOI: 10.1007/3-540-45586-8_5.

[FGL05] M. Fischetti, F. Glover, and A. Lodi. "The Feasibility Pump". In: *Mathematical Programming* 104 (Sept. 2005), pp. 91–104. DOI: 10.1007/s10107-004-0570-3.

[FK15a] A. Frieze and M. Karoński. "Random Graphs". In: *Introduction to Random Graphs*. Cambridge University Press, 2015, 3–18.

[FK15b] A. Frieze and M. Karoński. "Spanning Subgraphs". In: *Introduction to Random Graphs*. Cambridge University Press, 2015, 81–109.

[Fra02] A. Frangioni. "Generalized Bundle Methods". In: *SIAM Journal on Optimization* 13 (2002), pp. 117–156. DOI: 10.1137/S1052623498342186.

[Gam10] G. Gamrath. "Generic Branch-Cut-and-Price". Diplomarbeit. Technische Universität Berlin, 2010. URL: https://www.zib.de/userpage/gamrath/publications/gamrath2010_genericBCP.pdf.

[Geo74] A. M. Geoffrion. "Lagrangean Relaxation for Integer Programming". In: *Approaches to Integer Programming*. Ed. by M. L. Balinski. Berlin, Heidelberg: Springer Berlin Heidelberg, 1974, pp. 82–114. ISBN: 978-3-642-00740-8. DOI: 10.1007/BFb0120690.

[GG63] P.C. Gilmore and R. Gomory. "A Linear Programming Approach to the Cutting Stock Problem—Part II". In: *Operations Research* 11 (Dec. 1963). DOI: 10.1287/opre.11.6.863.

[GGBM13] J. Gondzio, P. González-Brevis, and P. Munari. "New Developments in the Primal–Dual Column Generation Technique". In: *European Journal of Operational Research* 224 (2013), pp. 41–51. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2012.07.024. URL: https://www.sciencedirect.com/science/article/pii/S0377221712005656.

[GV02] J.-L. Goffin and J.-P. Vial. "Convex Nondifferentiable Optimization: A Survey Focused on the Analytic Center Cutting Plane Method". In: *Optimization Methods and Software* 17 (2002), pp. 805–867. DOI: 10.1080/10556780210000608290a.

[HSS08] A. Hagberg, P. J. Swart, and D. A. Schult. *Exploring Network Structure, Dynamics, and Function Using NetworkX*. Tech. rep. Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[Hui+05] D. Huisman, R. Jans, M. Peeters, and A. P.M. Wagelmans. "Combining Column Generation and Lagrangian Relaxation". In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M. M. Solomon. Boston, MA: Springer US, 2005, pp. 247–270. ISBN: 978-0-387-25486-9. DOI: 10.1007/0-387-25486-2_9.

[Kan60] L. V. Kantorovich. "Mathematical Methods of Organizing and Planning Production". In: *Management Science* 6 (1960), pp. 366–422. URL: https://api.semanticscholar.org/CorpusID:62611375.

[KV12] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. 5th. Springer Publishing Company, Incorporated, 2012. ISBN: 3642244874.

[Lü11] M. E. Lübbecke. "Column Generation". In: Jan. 2011. ISBN: 9780470400531. DOI: 10.1002/9780470400531.eorms0158.

[LD05] M. E. Lübbecke and J. Desrosiers. "Selected Topics in Column Generation". In: *Operations Research* 53 (Dec. 2005), pp. 1007–. DOI: 10.1287/opre.1050.0234.

[LNS24] M. Lucci, G. Nasini, and D. Severín. "Solving the List Coloring Problem through a branch-and-price algorithm". In: *European Journal of Operational Research* 315 (June 2024), 899–912. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2024.01.038.

[Lod+22] A. Lodi, P. Olivier, G. Pesant, and S. Sankaranarayanan. "Fairness Over Time in Dynamic Resource Allocation With an Application in Healthcare". In: *Mathematical Programming* 203 (Nov. 2022), 285–318. ISSN: 1436-4646. DOI: 10.1007/s10107-022-01904-6.

[LSW22] A. Lodi, S. Sankaranarayanan, and G. Wang. *A Framework for Fair Decision-making Over Time with Time-invariant Utilities*. 2022. arXiv: 2212.10070 [math.OC]. URL: https://arxiv.org/abs/2212.10070.

[LSW23] A. Lodi, S. Sankaranarayanan, and G. Wang. "A Framework for Fair Decision-Making Over Time With Time-Invariant Utilities". In: *European Journal of Operational Research* (2023). ISSN: 0377-2217. DOI: 10.1016/j.ejor.2023.11.030. URL: https://www.sciencedirect.com/science/article/pii/S0377221723008718.

[LW18] M. E. Lübbecke and J. T. Witt. "The Strength of Dantzig–Wolfe Reformulations for the Stable Set and Related Problems". In: *Discrete Optimization* 30 (2018), pp. 168–187. ISSN: 1572-5286. DOI: https://doi.org/10.1016/j.disopt.2018.07.001. URL: https://www.sciencedirect.com/science/article/pii/S1572528617302608.

[LY84] D. Luenberger and Y. Ye. "Linear and Nonlinear Programming". In: *American Journal of Agricultural Economics* 67 (Jan. 1984). DOI: 10.2307/1240727.

[Mah+16] S. Maher, M. Miltenberger, J. P. Pedroso, D. Rehfeldt, R. Schwarz, and F. Serrano. "PySCIPOpt: Mathematical Programming in Python with the SCIP Optimization Suite". In: *Mathematical Software – ICMS 2016*. Springer International Publishing, 2016, pp. 301–307. DOI: 10.1007/978-3-319-42432-3_37.

[Mar+84] S. Martello, W. R. Pulleyblank, P. Toth, and D. de Werra. "Balanced Optimization Problems". In: *Operations Research Letters* 3 (1984), pp. 275–278. ISSN: 0167-6377. DOI: 10.1016/0167-6377(84)90061-0. URL: https://www.sciencedirect.com/science/article/pii/0167637784900610.

[NBN22a] M. H. Nguyen, M. Baiou, and V. H. Nguyen. "Nash Balanced Assignment Problem". In: *Combinatorial Optimization*. Ed. by I. Ljubić, F. Barahona, S. S. Dey, and A. R. Mahjoub. Cham: Springer International Publishing, 2022, pp. 172–186. ISBN: 978-3-031-18530-4.

[NBN22b] M. H. Nguyen, M. Baiou, and V. H. Nguyen. "Nash Balanced Assignment Problem". In: *7th International Symposium on Combinatorial Optimization (ISCO)*. Online Conference, France, May 2022. URL: https://hal.science/hal-03656133.

[NW88] G. Nemhauser and L. Wolsey. In: *Integer and Combinatorial Optimization*. John Wiley & Sons, Ltd, 1988. ISBN: 9781118627372. DOI: 10.1002/9781118627372.ch14. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118627372.ch14. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118627372.ch14.

[Pes+08] A. A. Pessoa, E. Uchoa, M. Poggi de Aragão, and R. Rodrigues. "Algorithms over Arc-time Indexed Formulations for Single and Parallel Machine Scheduling Problems". In: 2008. URL: https://api.semanticscholar.org/CorpusID:1041396.

[Pes+10] A. A. Pessoa, E. Uchoa, M. Poggi de Aragão, and R. Rodrigues. "Algorithms over Arc-time Indexed Formulations for Single and Parallel Machine Scheduling Problems". In: *Mathematical Programming Computation* 2 (2010), pp. 259–290. DOI: 10.1007/s12532-010-0019-z.

[PPS08] B. Petersen, D. Pisinger, and S. Spoorendonk. "Chvátal-Gomory Rank-1 Cuts Used in a Dantzig-Wolfe Decomposition of the Vehicle Routing Problem with Time Windows". In: *The Vehicle Routing Problem: Latest Advances and New Challenges*. Ed. by B. Golden, S. Raghavan, and E. Wasil. Boston, MA: Springer US, 2008, pp. 397–419. ISBN: 978-0-387-77778-8. DOI: 10.1007/978-0-387-77778-8_18.

[Pri] *PySCIPOpt. GitHub*. https://github.com/scipopt/PySCIPOpt/blob/master/tests/test_pricer.py. Accessed: 2024-08-08.

[Puc11] C. Puchert. "Primal Heuristics for Branch-and-Price Algorithms". master thesis. Technische Universität Darmstadt, 2011. URL: https://www.or.rwth-aachen.de/files/research/theses/2011/Puchert_Primal_Heuristics_for_Branch-and-Price_Algorithms.pdf.

[RF81] D. Ryan and B. Foster. "An Integer Programming Approach to Scheduling". In: *Computer Scheduling of Public Transport* 1 (Jan. 1981), pp. 269–.

[Van00] F. Vanderbeck. "On Dantzig-Wolfe Decomposition in Integer Programming and ways to Perform Branching in a Branch-and-Price Algorithm". In: *Operations Research* 48 (Feb. 2000), pp. 111–. DOI: 10.1287/opre.48.1.111.12453. URL: https://inria.hal.science/inria-00342641.

[Van05] F. Vanderbeck. "Implementing Mixed Integer Column Generation". In: *Column Generation*. Ed. by G. Desaulniers, J. Desrosiers, and M. M. Solomon. Boston, MA: Springer US, 2005, pp. 331–358. ISBN: 978-0-387-25486-9. DOI: 10.1007/0-387-25486-2_12.

[Van10] F. Vanderbeck. "Branching in Branch-and-Price: a Generic Scheme". In: *Mathematical Programming* 130 (Jan. 2010). DOI: 10.1007/s10107-009-0334-1. URL: https://inria.hal.science/inria-00311274.

[Van+94] P. H. Vance, C. Barnhart, E. L. Johnson, and G. L. Nemhauser. "Solving Binary Cutting Stock Problems by Column Generation and Branch-and-Bound". In: *Computational Optimization and Applications* 3 (1994), pp. 111–130. URL: https://api.semanticscholar.org/CorpusID:42319347.

[Vil+05] D. Villeneuve, J. Desrosiers, M. E. Lübbecke, and F. Soumis. "On Compact Formulations for Integer Programs Solved by Column Generation". In: *Annuals of Operations Research* 139 (2005), pp. 375–388. DOI: 10.1007/s10479-005-3455-9.

[VS06] F. Vanderbeck and M.W.P. Savelsbergh. "A Generic View of Dantzig–Wolfe Decomposition In Mixed Integer Programming". In: *Operations Research Letters* 34 (2006), pp. 296–306. ISSN: 0167-6377. DOI: 10.1016/j.orl.2005.05.009. URL: https://inria.hal.science/inria-00342623.

[VW10] F. Vanderbeck and L. A. Wolsey. "Reformulation and Decomposition of Integer Programs". In: *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Ed. by M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 431–502. ISBN: 978-3-540-68279-0. DOI: 10.1007/978-3-540-68279-0_13. URL: https://inria.hal.science/inria-00392254.