FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

# Branch-and-Price for the Steiner Tree Problem with Revenues, Budget and Hop Constraints

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Computational Intelligence**

eingereicht von

**Markus Sinnl**
Matrikelnummer 0726419

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Ao. Univ. Prof. Dr. Günther R. Raidl
Mitwirkung: Univ.-Ass. Dr. Markus Leitner

Wien, 31.08.2011

(Unterschrift Verfasser) (Unterschrift Betreuer)

# Erklärung zur Verfassung der Arbeit

Markus Sinnl
Wollmannsberg 43, 2003 Leitzersdorf

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

# Abstract

This thesis deals with the Steiner tree problem with revenues, budget and hop constraints (STPRBH), an $\mathcal{NP}$-hard combinatorial optimization problem with applications in telecommunications network design. An instance of the STPRBH is defined by a connected graph with a dedicated root node, a set of nodes with nonnegative revenues and positive costs assigned to edges. Furthermore, a budget $B \geq 0$ and a hop limit $H \in \mathbb{N}$ are given. The set of feasible solutions is given by all Steiner trees containing the root node, where every path from the root node to any other node in the tree contains at most $H$ edges. Furthermore, the total edge costs of such a tree must be lower or equal to the given budget $B$. The goal is to find a feasible solution with maximum revenue, i.e. to maximize the sum of revenues associated with nodes which are part of the solution.

Several formulations for the STPRBH based on integer linear programming using exponentially many variables are presented. Furthermore, branch-and-price approaches based on these formulations are introduced that allow for solving instances of the STPRBH to proven optimality. The practical implementations of these branch-and-price approaches do, however, suffer from various problems. Thus, the applicability of various attempts to improve their efficiency like stabilization techniques, different pricing strategies and heuristic methods to generate initial solutions is analyzed. Furthermore, promising methods are correspondingly adapted and applied to the STPRBH.

Tests on previously existing benchmark instances show that the presented branch-and-price approaches are competitive with existing exact methods based on branch-and-cut when the hop limit is rather restrictive or if the number of nodes with positive revenue is relatively small. Furthermore, when the budget $B$ does not play a role (i.e. is high enough to pose no restriction), branch-and-price usually outperforms branch-and-cut. It should be noted, however, that this specific variant of the STPRBH is not $\mathcal{NP}$-hard. For instances with a large hop limit or a large number of nodes with positive revenue the proposed branch-and-price approaches are not yet competitive to branch-and-cut. Due to the implemented stabilization and acceleration methods a significant speed-up of branch-and-price has, however, been achieved for these instances too.

# Kurzfassung

Diese Diplomarbeit behandelt das Steiner tree problem with revenues, budget and hop constraints (STPRBH), ein $\mathcal{NP}$-schweres kombinatorisches Optimierungsproblem mit Anwendungen unter anderem im Design von Telekommunikationsnetzen. Eine Instanz des STPRBH besteht aus einem ungerichteten Graphen mit einem Wurzelknoten, Knoten, die nicht-negativen Ertrag generieren, falls sie in einer Lösung mit dem Wurzelknoten verbunden sind und Kanten mit positivem Gewicht. Weiters ist ein Budget $B \geq 0$ und ein Hoplimit $H \in \mathbb{N}$ Teil einer Instanz. Erlaubte Lösungen sind alle Steiner-Bäume dieses Graphen, die den Wurzelknoten enthalten und in denen jeder Pfad vom Wurzelknoten bis zu einem anderen Knoten im Baum höchstens aus $H$ Kanten besteht. Weiters darf die Summe der Gewichte der Kanten im Baum höchstens $B$ betragen. Ziel ist es, eine gültige Lösung mit möglichst hohem Gewinn, der durch die Summe der Erträge der in der Lösung vorhandenen Knoten definiert ist, zu finden.

Es werden mehrere Formulierungen für das STPRBH als ganzzahliges lineares Programm (ILP) mit exponentiell vielen Variablen vorgeschlagen. Außerdem werden branch-and-price Verfahren, die auf diesen Formulierungen basieren und das exakte Lösen von Instanzen des STPRBH erlauben, eingeführt. Bei der Anwendung dieser branch-and-price Ansätze in der Praxis treten aber verschiedenste Probleme auf. Deshalb wird die Anwendbarkeit verschiedener Möglichkeiten deren Effizienz zu verbessern analysiert. Diese Möglichkeiten umfassen Stabilisierungstechniken, verschiedene Pricing-Strategien und Heuristiken, um Startlösungen zu generieren.

Tests auf schon existierenden Benchmark-Instanzen zeigen, dass die vorgeschlagenen branch-and-price Ansätze kompetitiv mit schon existierenden exakten Verfahren, die auf branch-and-price basieren, sind, falls das Hoplimit oder die Anzahl der Knoten mit positivem Ertrag vergleichsweise klein ist. In Instanzen, in denen das Budget $B$ keine Rolle spielt (d.h. hoch genug ist, um keine Einschränkung darzustellen), sind die branch-and-price Verfahren sogar meist klar besser als die branch-and-cut Verfahren. Es muss aber beachtet werden, dass diese spezifische Variante des STPRBH nicht $\mathcal{NP}$-schwer ist. Für Instanzen mit großem Hoplimit oder einer großen Anzahl von Knoten mit positivem Ertrag sind die vorgeschlagenen branch-and-price Verfahren noch nicht kompetitiv mit den branch-and-cut Verfahren. Durch die implementierten Stabilisierungsmethoden und Beschleunigungsmethoden wurde aber eine signifikante Beschleunigung der branch-and-price Verfahren erreicht.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

CHAPTER 1

# Introduction

A central problem when designing and planning modern communication networks consists of selecting certain entities with minimum total cost for establishing the necessary connections. An example would be the cheapest network connecting a server to all clients. This problem can be modeled as *Steiner tree problem on graphs*. Often, however, there are more restrictions and constraints necessary to model a real-world problem accurately. Connected clients could generate (different) revenues, leading to the goal of maximizing the difference between revenue gained by connecting clients and the total costs of realizing the network. This more general problem is known as *prize-collecting Steiner tree problem* and *Steiner tree problem with profits* [14].

Maximizing the difference between the collected revenue and the total costs frequently does not describe real-life situations in an adequate way. On the contrary to above assumptions, in real world the available budget is often fixed and companies are interested in maximizing the obtained revenue while not exceeding the given budget.

Another problem frequently encountered by telecommunication companies is the need to provide good Quality-of-Service (QoS) for their customers. QoS demands can be addressed in the following way: If each link in a network has the same reliability $\alpha$ (i.e. probability that the link is working) and the reliability of the links is independent, the probability that a connection consisting of $H$ links has no failure is $\alpha^H$ [17, 38]. Moreover, in a network with low traffic, the maximum delay between the server and any client is directly proportional to the number of links between the server and the client [37, 38]. Such a limit for the number of links between a dedicated root node and any other node in a network can be modeled as hop constraints [14, 17, 37, 38].

Thus, additionally considering the budget constraint and the hop constraints, we end up with a more realistic model, the *Steiner tree problem with revenues, budget and hop constraints* (STPRBH), which is the topic of this thesis.

Given a central server or some existing infrastructure, the goal is to connect those clients, which generate the maximum revenue, while a given budget-constraint for establishing this network must hold. Moreover the connections between the server and each client must provide

good QoS. To achieve this, every connection between the server and a client is not allowed to exceed a given number of links.

More formally, the STPRBH is defined as follows:

**Definition 1.1** (STPRBH)**.**
*We are given a graph $G = (V, E)$ with vertex set $V$, edge set $E$, and a dedicated root node $0 \in V$. Furthermore, we are given a cost function $c_e > 0$, $\forall e \in E$, assigning a positive cost value to each edge and a revenue function $r_v \geq 0$, $\forall v \in V$, assigning a nonnegative revenue to every node. Each instance is further defined by the maximum available budget $B \geq 0$, and the hop limit $H \in \mathbb{N}$.*

*A feasible solution to the STPRBH is a tree $G_S = (V_S, E_S)$, $V_S \subseteq V$, $0 \in V_S$, $E_S \subseteq E$, connecting all nodes of $V_S$, which does not violate the budget and hop constraint, i.e. $\sum_{e \in E_S} c_e \leq B$, and the path between any node $v \in V_S$ and the root node $0$ does not contain more than $H$ edges. The objective is to maximize the total revenue, i.e.*

$$\max \sum_{t \in V_S} r_t.$$

The node set of a STPRBH instance can be separated into two disjoint sets of terminal nodes and Steiner nodes according to the revenue of the nodes.

**Definition 1.2** (Terminal nodes of an STPRBH instance)**.**
*Given an instance $I = (G = (V, E), 0, c : E \to \mathbb{Q}^+, r : V \to \mathbb{Q}_0^+, H, B)$ of the STPRBH, the set of terminal nodes $T$ of this instance is defined as the union of the set of all nodes with positive revenue and the root node $0$, i.e. $T = \{v \in V \mid r_v > 0\} \cup \{0\}$*

**Definition 1.3** (Steiner nodes of an STPRBH instance)**.**
*Given an instance $I = (G = (V, E), 0, c : E \to \mathbb{Q}^+, r : V \to \mathbb{Q}_0^+, H, B)$ of the STPRBH, the set of Steiner nodes of $S$ of this instance is defined as the set of nodes with zero revenue without the root node, i.e. $S = \{v \in V \mid r_v = 0 \wedge v \neq 0\}$*

We observe that there exists an optimal solution for each instance such that only the root node and terminals nodes are leaves, i.e. have node degree one.

Figure 1.1 with $H = 2$, $B = 15$ and the root node marked as double-circle shows an exemplary instance of the STPRBH. The node number corresponds to the revenue generated by this node and the number next to each edge gives the cost of the corresponding edge. Nodes with uppercase letters are Steiner nodes, i.e. they have zero revenue.

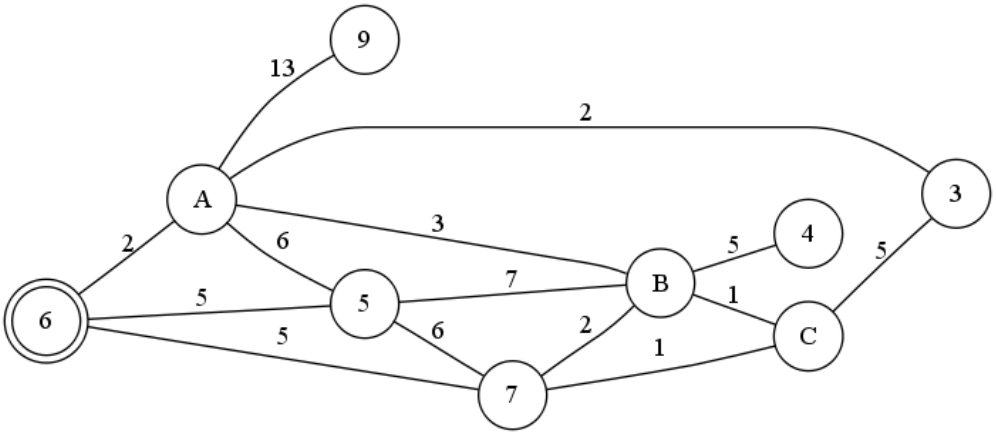



Figure 1.1: Graph of an exemplary instance of the STPRBH.

Figure 1.2 shows an optimal solution $S$ of this instance with a total revenue of 21. The nodes $V_S$ in the solution are marked shaded and bold and edges $E_S$ in the solution are marked bold. It is easy to see that the node with revenue four cannot be part of a feasible solution, since it does not fulfill the hop limit (i.e. every path from the root node to this node has at least length three). Moreover, the node with revenue nine cannot be connected in the optimal solution, since connecting this node would consume all of the available budget and therefore no other node could be connected. Hence, connecting node nine would yield a total revenue of fifteen, while the solution shown in Figure 1.2 yields a total revenue of 21. Since the latter connects all terminal nodes, except nodes four and nine, it is optimal.

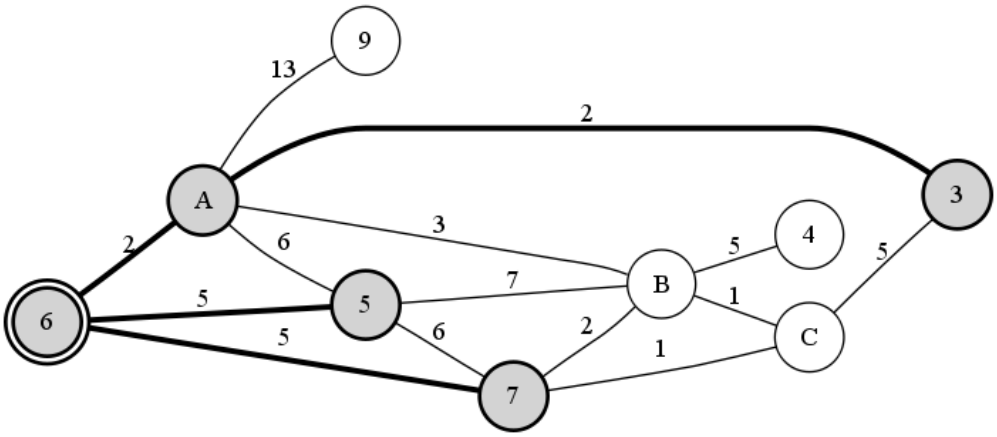



Figure 1.2: Solution of the instance given in Figure 1.1 with $H = 2, B = 15$.

The STPRBH is an extension of the well-known Steiner tree problem (STP) on a graph, which is $\mathcal{NP}$-hard [31]. The STPRBH is also $\mathcal{NP}$-hard, under the condition that $B$ is smaller

than the sum of the edge-costs [69].

This thesis introduces branch-and-price methods based on path-formulations for the STPRBH and studies possibilities to improve the efficiency of them.

## 1.1 Outline of the Thesis

The remainder of this thesis is organized as follows: Chapter 2 introduces the theoretical background of the methods used in this thesis, i.e. combinatorial optimization and integer linear programming (ILP). Moreover, different strategies to solve ILPs like branch-and-bound, branch-and-cut and branch-and-price are discussed in this chapter. Chapter 3 gives an overview over previous and related work.

In Chapter 4, an undirected and a directed ILP formulation with exponentially many variables and a way to solve these formulations by means of branch-and-price are presented. Also the pricing subproblem is described in this chapter.

Chapter 5 discusses various attempts to improve the efficiency of the suggested branch-and-price algorithms like stabilization techniques, different pricing strategies and heuristic methods to generate initial solutions. Preprocessing techniques are also discussed in this chapter. In Chapter 6, the computational results obtained by the various approaches (and combinations thereof) are presented.

Chapter 7 concludes this thesis by a summary and an outlook to potential further work.

CHAPTER 2

# Preliminaries

## 2.1 Combinatorial Optimization

A combinatorial optimization problem (COP), also called discrete optimization problem, consists of finding the optimum value of a function on a set of finite or countable infinite variables. This function is called objective (or cost) function and the set of variables is normally constrained by other functions. All subsets of this set of variables, which fulfill all constraints are feasible solutions and the set of feasible solution of an COP forms the feasible region for this COP. The following is a formal definition of an optimization (minimization) problem, taken from [68] and a formal definition of a combinatorial optimization problem.

**Definition 2.1** (Optimization problem [68])**.**
*An instance of an optimization problem is a pair $(F, c)$, where $F$ is any set, the domain of feasible points; $c$ is the cost function, a mapping*

$$c : F \to \mathbb{R}.$$

*The problem is to find an $f \in F$ for which*

$$c(f) \leq c(y) \text{ for all } y \in F.$$

*Such a point f is called an optimal solution to the given instance. An optimization problem is a set I of instances of an optimization problem.*

**Definition 2.2** (Combinatorial optimization problem [68])**.**
*A combinatorial optimization problem is an optimization problem, where $F$ is finite.*

A wide range of problems falls in the category of combinatorial optimization, because the only requirement for a problem to be a COP is a finite representation in a computer. Graphs can be represented in a finite way by either an adjacency list or an adjacency matrix, therefore problems like the traveling salesman problem, the minimum spanning tree problem, network

flow problems [2] and variants of these problems are all COPs. Other type of problems which are COPs are for example scheduling [13] or knapsack problems [48].

There are many ways to solve COPs. The first idea which may come to ones mind is enumerating the whole search space, i.e. listing all feasible solutions and comparing the value of their objective functions. This clearly works due to the finiteness of the solution space, but is very slow in general for even moderate sized instances of COPs [68]. Therefore a rich variety of other (better) methods for solving COPs emerged, which can be mainly parted into (meta-)heuristic methods and exact methods. Both methods can in principle be applied to all kind of COPs.

Heuristic methods try to intelligently construct a solution of good quality in a fast way. An example for heuristics are greedy algorithms, which use local information to build a solution. In general there is no guarantee about the quality of the solutions found by heuristic methods. Metaheuristic methods try to improve one or more given (feasible) solution(s) by various strategies of iteratively changing the elements of the solution. Again, there is no guarantee that a metaheuristic finds the optimal solution or even a solution bounded in a fixed ratio to the optimal solution for all instances of a COP. Examples for metaheuristics are variable neighborhood search [64], ant colony optimization [25] or genetic algorithms [35]. See also [9, 33] for a broader overview on the topic of metaheuristics.

The other group of methods to solve COPs are exact methods like integer linear programming. The main idea behind this kind of methods is to provide an "intelligent" form of enumeration, which results in better runtime than the brute-force method of enumerating the whole search space. Since this thesis mainly deals with methods based on integer linear programming, a more in-depth look of it will be given in the next sections together with a discussion of linear programming and other related topics. More on linear programming can for example be found in [21, 22, 68], an excellent treatment of integer linear programming is given in [79], moreover [74] gives a general introduction to convex programming, which contains both linear programming and integer linear programming as subset.

Aside from these two big groups of methods, other variants like hybrid methods, which combine exact and (meta-)heuristic methods, approximation algorithms, which do not necessarily give the optimal solution, but a solution within a fixed ratio of the optimal solution [76], or simply algorithms developed especially for a problem like the Ford-Fulkerson algorithm for network flows [29] or the Hungarian method for the assignment problem [53], do exist.

## 2.2 Mathematical Background

This section recalls some mathematical definitions and theorems, which will be used later on. The definitions are taken from [21, 22, 68, 74].

**Definition 2.3** (Convex combination and Convex hull [22])**.**
*Given vectors* $\mathbf{x_1}, \ldots, \mathbf{x_k} \in \mathbb{R}^n$*,*

$$x = a_1\mathbf{x_1} + \ldots + a_k\mathbf{x_k} \tag{2.1}$$

*with* $a_i \geq 0, 1 \leq i \leq k$ *and* $\sum_{i=1}^{k} a_i = 1$*, is called convex combination of these vectors. The set of all such convex combinations is called convex hull of these vectors.*

For the remainder of this thesis, vectors will not be explicitly denoted in bold whenever it is clear from the context whether we are concerned with a scalar or a vector.

Informally, a set is convex if every two elements of this set can be connected with a line segment, which also lies in the set. A line segment connecting two vectors $x_1, x_2$ is defined as all vectors $y$, which are a solution of the following equation

$$y = \lambda x_1 + (1 - \lambda)x_2 \tag{2.2}$$

with $0 \leq \lambda \leq 1$. Given a set $\mathcal{S}$, this equation can be used to easily check if this set is convex. Moreover, it can be used to show that the intersection of two convex sets is also a convex set. Next, hyperplanes and halfspaces are defined. Note that in the definition of hyperplanes and halfspaces, there is only one vector $x$ in $\mathbb{R}^n$, in contrast to the $k$ vectors in the previous definitions

**Definition 2.4** (Hyperplane [22])**.**
*The set of vectors $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ fulfilling the equation*

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n = b \tag{2.3}$$

*with at least one $a_i \neq 0, 1 \leq i \leq n$, is called a hyperplane.*

**Definition 2.5** (Halfspace [22])**.**
*The set of vectors $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ fulfilling the equation*

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq b \tag{2.4}$$

*with at least one $a_i \neq 0, 1 \leq i \leq n$, is called a halfspace.*

Both hyperplanes and half-spaces are also convex sets and therefore also their intersections are convex sets. Moreover, given any two disjoint convex sets, a hyperplane separating these sets can be found. Such a hyperplane is called separating hyperplane.

**Definition 2.6** (Convex Polyhedron and Convex Polytope [22])**.**
*A convex polyhedron is the set of points common to one or more halfspaces. If a convex polyhedron is bounded, it is called convex polytope.*

For the rest of this thesis, the term polyhedron will be used to denote both convex polyhedrons and convex polytopes.

Formulating a COP more mathematically, we get

$$\begin{aligned} \min \quad & c(x) \\ \text{s.t.} \quad & f_i(x) \leq b_i \quad i = 0, \ldots, m \end{aligned} \tag{2.5}$$

where $x$ denotes a vector in $\mathbb{Q}^n$, $c$ denotes the objective function and the $f_i$ denote the constraint functions. If the objective function and the constraint functions are all linear, this means satisfying

$$f_i(\alpha x + \beta y) = \alpha f_i(x) + \beta f_i(y) \tag{2.6}$$

the COP is a linear program. A linear program is therefore a special case of a general convex program. An important fact about convex programs is, that a local optimum is also a global optimum. Moreover, the set of vectors corresponding to feasible solutions (i.e. solutions which fulfills all constraint functions) is also a convex set.

## 2.3 Linear Programming

This section and its subsections introduce linear programming, following [21, 22, 68]. We begin with the definition of a linear program (LP) in standard form.

**Definition 2.7** (Linear Program in standard form [68])**.** *Given* $A \in \mathbb{Q}^{(m \times n)}$, $b \in \mathbb{Q}^m$ *and* $c \in \mathbb{Q}^n$

$$\begin{aligned} \textit{min} \quad & c^T x = z \\ \textit{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{2.7}$$

*is called Linear Program in standard form*

As usually, $c^T x$ will be called objective function, $x \in \mathbb{Q}^n$ solution vector and $A$ constraint matrix. A solution vector corresponding to an optimal solution will often be denoted with $x^*$. There are many other forms of LPs possible, e.g. equality instead of inequality constraints. With the help of slack variables, these different forms can be transformed into each other. The set of solution vectors to an LP forms a polyhedron $\mathcal{P}$.

$$\mathcal{P} = \{x \mid Ax \leq b, x \geq 0\} \tag{2.8}$$

Example 2.1 gives an example of an LP in two variables.

**Example 2.1.**

$$\begin{aligned} \textit{min} \quad & 3x_1 + 3x_2 \\ \textit{s.t.} \quad & 2x_1 + x_2 \leq 3 \\ & x_1 + 2x_2 \leq 4 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{aligned} \tag{2.9}$$

Figure 2.1 shows the polyhedron of the LP in the example. The feasible region is shaded in gray.

Example 2.2 shows how the LP of Example 2.1 can be transformed into standard form with the help of slack variables $s_1$ and $s_2$, respectively.

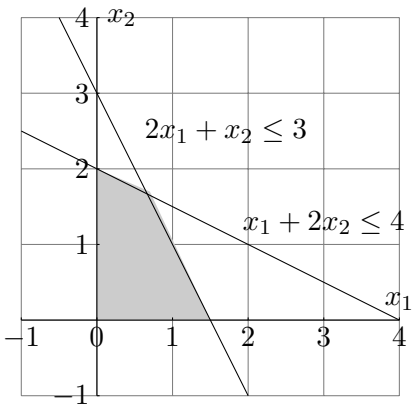



Figure 2.1: The polyhedron corresponding to the LP of Example 2.1.

**Example 2.2.** *By introducing slack variables $s_1$, $s_2$, we get the LP*

$$\begin{aligned} \textit{min} \quad & 3x_1 + 3x_2 \\ \textit{s.t.} \quad & 2x_1 + x_2 + s_1 = 3 \\ & x_1 + 2x_2 + s_2 = 4 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \\ & s_1 \geq 0 \\ & s_2 \geq 0 \end{aligned} \tag{2.10}$$

*which is equivalent to the LP of Example 2.1*

At least one optimal solution of an LP must lie on the corners of the polyhedron formed by the feasible solutions to the LP. This fact is used in the simplex method for solving LPs. The algorithm was invented by Dantzig in 1947 (first published in 1949 [19]) and has a good runtime performance in general, although it is not a polynomial time algorithm [21, 68]. The simplex method is covered in more detail in the next section.

In 1979, Khachian [49] introduced the ellipsoid method. The method tries to find an ellipsoid, which contains a solution and iteratively replaces this ellipsoid by a smaller one. The ellipsoid method was the first polynomial time algorithm for LP, but its practical performance is rather poor and therefore it has mainly theoretical value [21, 22, 68].

Today, variants of the simplex method and various interior point methods, which start out from a point in the interior of the feasible polyhedron and became competitive in 1984 [22] by the introduction of Karmarkars interior point method [47], are used to solve LPs.

### The Simplex Method

In this section, an outline of the well-known simplex method for solving Linear Programs will be given, following [21, 22]. As already mentioned, the simplex method uses the fact, that at least one optimal solution of an LP corresponds to a vertex (extreme point) of the polyhedron caused by the LP.

To describe this mathematically, we need the term basic feasible solution: Consider an LP in matrix form, i.e. $min\{c^T x | Ax = b, x \geq 0\}$, it can be transformed into the so called canonical form, where $Ix_B + \bar{A}x_N = \bar{b}$ replaces $Ax = b$ with the same solution as in the original form. Note that $I$ denotes the identity matrix and $x$ is split into $x_B$ called the basic variables and $x_N$ the non-basic variables. Given such a partition in basic and non-basic variables, a basic feasible solution can be found by setting the non-basic variables to zero and then reading off the values for the basic variables. Note that a basic feasible solution lies on a vertex of the polyhedron $\{Ax = b, x \geq 0\}$.

Another important term is reduced cost (also called relative costs): These costs are the coefficients $\bar{c}$ of the variables in the objective function when the problem is of canonical form. Note that these costs depend on the variables in the basis, hence the name relative costs. By increasing the variable associated with a negative reduced cost coefficient, we can improve the solution, since we have a minimization problem. Note that only non-basic variables are eligible for increasing, because the cost coefficient of a basic variable is zero. As a result of increasing the value of a non-basic variable, this variable enters the basis, since it gets a positive value. Furthermore, a basic variable leaves the basis, because it gets a value of zero.

The simplex method now simply moves from one basic feasible solution to another basic feasible solution. The process is repeated as long as there is a non-basic variable with negative reduced cost.

Note that the simplex method needs a basic feasible solution to start with. This problem of finding a starting solution can also be modeled as linear program and is called phase one of the simplex method, the movement from one basic feasible solution to another until optimality is called phase two.

There exists many variations of the simplex method, like the revised simplex method, which does not store the whole tableaux, but generates columns on-the-fly (see column generation in Section 2.4), or the dual simplex method, which works on the dual program (see next section regarding duality).

### Duality Theory

To every LP a dual program, which is also an LP, can be found. The original LP is called primal program in this context. This primal-dual connection plays a central role in the duality theory of linear programming.

Definition 2.8 gives a formal definition of the dual.

**Definition 2.8** (Dual of a Linear Program [68])**.**

*Given a LP in standard form (see Definition 2.7), the dual is defined as follows:*

$$\begin{aligned} \textit{max} \quad & b^T\pi = v \\ \textit{s.t.} \quad & A^T\pi \leq c \\ & \pi \geq 0 \end{aligned} \tag{2.11}$$

Example 2.3 shows the dual program of the LP from Example 2.1:

**Example 2.3.**

$$\begin{aligned} \textit{max} \quad & 3y_1 + 4y_2 \\ \textit{s.t.} \quad & 2y_1 + y_2 \leq 3 \\ & y_1 + 2y_2 \leq 3 \\ & y_1 \leq 0 \\ & y_2 \leq 0 \end{aligned} \tag{2.12}$$

As seen in the example, a dual program can also be obtained, if the primal program is not in standard form. Table 2.1, where $A_j$ denotes the $j\text{-}th$ column of the constraint matrix $A$, summarizes the rules needed for conversion [21, 68].

Table 2.1: Primal-Dual conversion rules [21, 68]

| PRIMAL | DUAL |
|---|---|
| $min\ c^T x$ | $max\ b\pi^T$ |
| $a_i^T x = b_i$ | $\pi_i$ unbounded |
| $a_i^T x \geq b_i$ | $\pi_i \geq 0$ |
| $a_i^T x \leq b_i$ | $\pi_i \leq 0$ |
| $x_j$ unbounded | $\pi_i A_j = c_j$ |
| $x_j \geq 0$ | $\pi_i A_j \leq c_j$ |
| $x_j \leq 0$ | $\pi_i A_j \geq c_j$ |

Using this relationship between primal and dual programs, two central results in linear programming, the weak and strong duality theorems, can be stated.

**Theorem 2.1** (Weak Duality Theorem [68])**.** *If x is a feasible solution to the primal and $\pi$ is a feasible solution to the dual, then*

$$\pi^T b \leq c^T x$$

This means, every feasible solution to the dual problem is a lower bound for the primal (assuming that the primal problem is a minimization problem, otherwise the dual is an upper bound).

**Theorem 2.2** (Strong Duality Theorem [68])**.** *If an LP has an optimal solution, so does its dual and the optimal costs of both are equal*

## 2.4 Integer Linear Programming

From a practical point of view, it is often necessary to restrict (some) of the solution variables of an LP to integers. Consider for example variables $x_{ij}$, which indicate that a machine $i$ is built in location $j$. If we get a fractional value like $0.574$ for $x_{ij}$ in the optimal solution, it is not of much help. Moreover, there are many formulations for COPs, in which the solution values are restricted to be either $0$ or $1$ to indicate if an object is part of the solution or not. Therefore integer linear programming (ILP), also known in the literature as integer programming (IP) plays an important role in the field of combinatorial optimization. Unless otherwise mentioned, this section and its subsections are based on [79].

An ILP in standard form is defined as following.

**Definition 2.9** (Integer Linear Program [79])**.**
*Given $A \in \mathbb{Q}^{(n \times m)}$, $b \in \mathbb{Q}^m$ and $c \in \mathbb{Q}^n$,*

$$\begin{aligned} \textit{min} \quad & c^T x \\ \textit{s.t.} \quad & Ax \leq b \\ & x \geq 0 \\ & x \in \mathbb{Z} \end{aligned} \tag{2.13}$$

*is called an integer linear program or ILP for short.*

The LP-relaxation of an ILP, a notion which will be important later on, is defined as following:

**Definition 2.10** (LP-relaxation of an ILP [79])**.**
*Given an ILP $min\{c^T x | Ax \leq b, x \geq 0, x \in \mathbb{Z}\}$. The LP-relaxation of this ILP is*

$$\begin{aligned} \textit{min} \quad & c^T x \\ \textit{s.t.} \quad & Ax \leq b \\ & x \geq 0 \end{aligned} \tag{2.14}$$

*i.e. the same program without the solution variables constrained to integers.*

Unfortunately, solving ILPs is $\mathcal{NP}$-hard [31]. The simplest strategy for getting an integer solution vector from a fractional result, rounding the fractional values of the solution vector to the nearest integer, does not necessarily give the optimal solution for an integer solution vector and can even lead to infeasibility of the solution. For an illustration of this problem, see Figure 2.2.

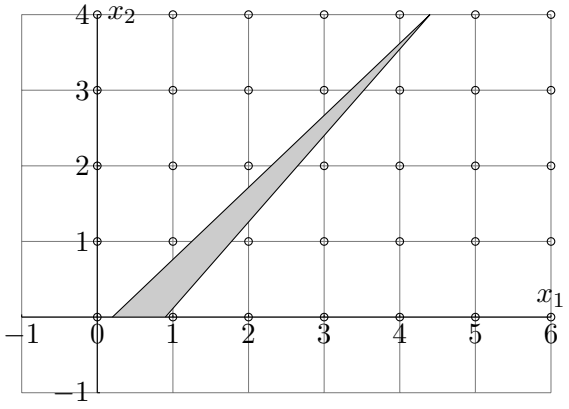



Figure 2.2: A LP with no feasible integer solution

Hence, more sophisticated approaches for solving ILPs are needed. Some of these approaches are presented in the next subsections. For sake of presentation, the ILPs are considered to be minimization problems in standard form, but the approaches can easily be generalized to all other forms of ILPs.

However, there is a special case of ILPs, where we get integer solutions without the use of such methods: If the constraint matrix $A$ of an ILP is total unimodular (see Definition 2.11) and $b \in \mathbb{Z}$, then the solution of the LP-relaxation of an ILP is automatically integral. Example for problems with a total unimodular constraint matrix are the transportation problem [21] or the linear assignment problem [21].

**Definition 2.11** (Unimodularity and Total Unimodularity [79])**.**
*A square, integer matrix B is called unimodular (UM) if it has determinant -1 or 1. An integer matrix A is called totally unimodular (TUM) if every square, nonsingular submatrix of A is unimodular.*

There are two more definitions [79], which are important for solving COPs with ILP. The first is concerned with the representation of the problem as integer linear program, which is called formulation.

**Definition 2.12** (Formulation of a COP as ILP [79])**.** *A polyhedron $P \subseteq \mathbb{R}^{n+p}$ is a formulation for a set $X \subseteq (\mathbb{Z}^n \times \mathbb{R}^p)$ iff $X = P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$*

Usually, there are many ways to formulate a problem as ILP, therefore it is a natural question to ask, which formulation should be preferred. The following definition gives an answer to this question.

**Definition 2.13** (Better formulation [79])**.**
*Given a set $X \subseteq \mathbb{R}^n$ and two formulations $P_1, P_2$ for $X$, $P_1$ is a better formulation than $P_2$, if $P_1 \subset P_2$*

### Cutting Plane Algorithms

Cutting plane algorithms were the first methods for solving ILPs. The method was first used in 1954 by Fulkerson, Dantzig and Johnson [20] in order to solve an instance of a traveling salesman problem (TSP), where 49 cities in different states in the United States had to be visited. Their LP formulation of the TSP has exponentially many inequalities. Therefore, they started with a "smaller" formulation without all inequalities, solved it and then checked, if some of the TSP constraints are violated in the solution. If yes, the violated inequalities are added to the "smaller" formulation and the whole procedure is repeated. If at some point, the solution does not violate any TSP constraint, we get an optimal solution for the instance and are done (note that the integrallity of the solution is achieved by their TSP constraints).

In 1958 Gomory [36] stated a cutting plane procedure in order to solve ILPs. The idea behind this method is to solve the LP-relaxation (see Definition 2.10) of the ILP, which provides an lower bound for the optimal solution of the ILP. Then check, if the solution vector consists only of integers. If not, a linear constraint, which does not exclude any feasible solution to the ILP (such a linear constraint is called a "cutting plane", hence the name), is added to the LP-relaxation. The whole procedure repeats, until we find an solution vector that consists of integers only.

Adding cutting planes (for both described uses) can be considered as "row generation": The constraints of an LP can be written in matrix form in the following way:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} \leq \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_m \end{pmatrix} \tag{2.15}$$

A cutting plane is a linear inequality $a_{c1}x_1 + a_{c2}x_2 + \ldots + a_{cn}x_n \leq b_c$. Therefore, adding a cutting plane results in an LP with one more row:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \\ a_{c1} & a_{c2} & \dots & a_{cn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} \leq \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_m \\ b_c \end{pmatrix} \tag{2.16}$$

For more technical details, see for example [21, 22, 68].

### Branch-and-Bound

This technique has its origins in a paper from Land and Doig [54] written in 1960. Like cutting plane algorithms, it also works on the LP-relaxation of the ILP. However, in contrast to cutting plane algorithms, one does not add cutting planes to the LP-relaxation in case $x^*$ (the solution value of the current LP-relaxation) has some non-integer values. Instead, the LP-relaxation is split in two mutually exclusive subproblems (which are also LPs) caused by a fractional value in $x^*$ and these subproblems need to be resolved. This process, the branching part of branch-and-bound, is repeated and the result of it is a binary tree like the example in Figure 2.3. At some

point, in every branch, we either get an integer solution or the LP is infeasible, in both cases the branching-process at this particular branch is stopped. After every branch-process has stopped, the minimal solution value among all the branches is the optimal solution to this ILP.

For example, if $x_i^0 = 5.6$, where $x^0$ denotes the optimal solution vector at the first iteration of branch-and-bound, then $x_i \leq 5$ is added to one subproblem and $x_i \geq 6$ to the other.

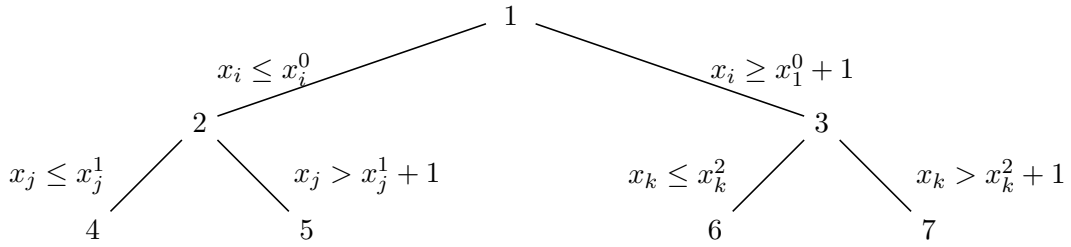



Figure 2.3: A branch tree

However, using this branching-process on its own is usually not effective, because a large portion of the search-space will be searched this way. Here bounding comes into play to speed things up. It relies on the facts, that the solution of the LP-relaxation provides a lower bound on the optimal ILP solution (a lower bound obtained in any other way, e.g. with heuristics, works too) and the current best integral solution $x^I$, provides an upper bound. With this information, the branching-tree can be pruned at some branches, leading to branch-and-bound,

Suppose we have obtained some solution $x^I$ during branch-and-bound. Then at all the branches, where the solution value $c^T x^*$ of the LP-relaxation is greater than $c^T x^I$, branching can be stopped, because no better solution value for the ILP can be found at this branches. This is called pruning by bound. Moreover, we can stop at some branch, if $c^T x^* = c^T x^I$, this is pruning by optimality. Pruning by infeasibility at a branch happens, if the LP-relaxation at this branch is infeasible.

Algorithm 1 shows a generic branch-and-bound code for a minimization problem $P$. In the pseudocode, $nodes$ denotes the set of active nodes with formulations $P_i$, $upper$ holds the global upper bound, $lower_i$ the lower bound at node $i$, $solution_i$ denotes the solution at node $i$ and $best$ is the solution with the currently best solution value.

**Algorithm 1** A generic branch-and-bound template

```
1:  procedure BRANCHANDBOUND(Problem P)
2:      nodes ← ∅ ∪ P
3:      best ← NULL
4:      upper ← ∞
5:      while nodes ≠ ∅ do
6:          choose a node i ∈ nodes to branch
7:          nodes ← nodes \ {i}
8:          SOLVE(P_i)
9:          if P_i is infeasible then
10:             do nothing (i.e. prune by infeasibility)
11:         else
12:             if lower_i ≥ upper then
13:                 do nothing (i.e. prune by bound)
14:             else
15:                 if solution_i is a feasible solution to the original problem then
16:                     upper ← lower_i (i.e. prune by optimality)
17:                     best ← solution_i
18:                 else
19:                     generate subproblems P_{i_1}, ..., P_{i_k}
20:                     nodes ← nodes ∪ {P_{i_1}, ..., P_{i_k}}
21:                 end if
22:             end if
23:         end if
24:     end while
25:     best solution value: upper
26:     best solution: best
27: end procedure
```

It should be noted, that the use of branch-and-bound is not restricted to solve ILPs, but can be used to solve any COP where lower (or upper bounds) can be obtained and the solution set can be partitioned [68, 79].

### Branch-and-Cut

Branch-and-cut combines cutting plane algorithms with branch-and-bound. This combination was first proposed by Padberg and Rinaldi [66, 67] for the TSP. Before the introduction of this technique, solving an ILP, where cutting planes caused by violated inequalities were added, was done by restarting the whole solution process from scratch when a cut was added. The combination of both phases with the branch-and-cut method often gives in a much smaller branch-and-bound tree, which in turn results a shorter runtime [79].

### Column Generation and Branch-and-Price

Column generation essentially implements the opposite idea of cutting plane methods: Instead of starting with a formulation with a small number of inequalities and gradually adding inequalities caused by violated constraints, column generation starts with a small number of variables and gradually adds variables aiming to improve the objective value. Historically, column generation has its roots in Dantzig-Wolfe decomposition [23], which allows the transformation of a "normal" linear program in an equivalent formulation with a huge number of variables. The first application of column generation was by Gilmore and Gomory [32] for the cutting-stock problem.

More formally, consider the following LP, called master problem (MP) in the context of column generation (this section follows [3, 24, 61, 68]).

$$\begin{aligned} \min \quad & \sum_{j \in J} c_j \lambda_j \\ \text{s.t.} \quad & \sum_{j \in J} a_j \lambda_j \geq b \quad (\pi) \\ & \lambda_j \geq 0 \qquad\qquad j \in J \end{aligned} \tag{2.17}$$

In each iteration of the simplex method, we are looking for a non-basic variable with negative reduced cost to enter the basis, i.e. given the dual variables $\pi$, we are trying to find a $j^* \in J$, s.t. $j^* = argmin_{j \in J} c_j - \pi^T a_j$. However, if $|J|$ is huge, this explicit pricing can turn out to be too costly. Therefore we start with a small subset $J' \subseteq J$ of columns, the resulting program is called the restricted master problem (RMP). Let $\mathcal{A}$ denote the set of coefficients of the $j \in J$. The pricing operation (also called pricing subproblem or pricing oracle) consists of solving the following subproblem:

$$\bar{c}^* = min_{j \in J} \{c(a_j) - \pi^T a_j | a_j \in \mathcal{A}\}$$

where $c(a_j)$ denotes a function to compute the cost $c_j$ given $a_j$. If $\bar{c}^* \geq 0$, i.e. there is no variable with negative reduced costs $\bar{c}_j, \forall j \in J$, and we are finished, otherwise we add the column corresponding to $\bar{c}^*$ to the RMP. Note that it is not necessary to find the variable with the most negative reduced costs, any variable with negative reduced costs works. Naturally, if we have a maximization problem instead of an minimization problem, we are looking for non-basic variables with positive reduced costs to price in and stop, if we only find $\bar{c}_j, \forall j \in J$, which are less than or equal to zero.

Often, the pricing subproblems are (variants of) well-known combinatorial problems and therefore can be solved by means of problem-specific combinatorial algorithms instead of (integer) linear programming.

It should be noted that column generation on its own is a technique for linear programming. For the use in integer linear programming it is usually embedded in a branch-and-bound framework and called branch-and-price. At every node of the branch-and-bound tree, column generation is performed. It is important to do column generation after branching, because the restriction of some variable due to branching may lead to some columns with negative reduced costs, which were not present in the original MP. Branching-schemes, which are an important part of branch-and-price are discussed in Section 5.2

CHAPTER 3

# Previous & Related Work

The STPRBH has been introduced by Costa [15] in 2006. In [17], four exact methods based on integer linear programming are presented: An undirected and a directed formulation using subtour-elimination constraints, a formulation with Miller-Zemlin-Tucker inequalities and a variation thereof called Garcia-Gouveia hop formulation are given. The use of Miller-Zemlin-Tucker constraints in context of hop constraints has been proposed by Gouveia for the *minimum spanning tree problem with hop constraints* (MSTH) [37], and Voß adapted the formulation to the *Steiner tree problem with hop constraints* (STPH) [77]. The constraints used in the Garcia-Gouveia hop formulation have been introduced by Garcia in [30] and Gouveia in [40]. The exact methods presented in [17] are capable of solving instances which up to 500 nodes and 625 edges in reasonable time.

To solve bigger instances approximately, Costa et. al proposed a greedy algorithm, a destroy-and-repair algorithm and a tabu search in [16]. The greedy algorithm starts out with a solution consisting only of the root node and then gradually adds terminal nodes using hop-constrained paths. The terminals are picked in a greedy strategy depending on the revenue of the terminal and the costs of the chosen path to connect the terminal.

The destroy-and-repair algorithm starts with a feasible solution for the STPRBH generated by their greedy algorithm and tries to improve the current solution by exchanging edges in the solution with edges not in the solution. The algorithm sets the cost of one edge in the current solution to infinity and then reruns the greedy algorithm. This is done for all edges incident to a leaf node in the current solution, and the solution with the best objective value is kept. The whole procedure is repeated until we do not get a better solution.

The tabu search consists of two main moves *add* and *remove*. The add move adds terminals to the solution using hop-constrained paths (i.e. like the greedy algorithm) and the added paths are allowed to violate the budget. The remove move does the opposite, it deletes branches of the solution to get a budget-feasible solution. There is also a third move, *destroy*, which is only used occasionally. This move deletes whole subtrees of the solution to get out of local optima.

In general, column generation and branch-and-price algorithms are seldom used for Steiner tree and spanning tree problems: Gouveia et al. [41] have tried to solve the LP-relaxation of a

formulation for the *rooted delay-constrained minimum spanning tree problem* (RDCMST) with column generation. An instance of a RDCMST consists of a graph with edge-costs, edge-delays and a dedicated root node. Furthermore, a delay bound is part of the instance. A feasible solution to the RDCMST is a tree, which connects every node in the graph to the root node in such a way that for every node the sum of the edge-delays in the path from the root node to the node does not exceed the delay bound. The goal is to find a feasible solution with minimum edge-cost. Column generation has turned out to be not competitive to two other approaches tested in [41]. One of these two other methods consists of using Lagrangian relaxation in combination with a primal heuristic. The other approach solves the constrained shortest path problem for every terminal on layered graphs using a multicommodity flow formulation. This approach using layered graph has been extended for the full formulation of the RDCMST by Gouveia et al. [44].

The *rooted delay-constraint minimum Steiner tree problem* (RDCSTP) is closely related with the RDCMST. In contrast to the RDCMST, only the set of terminal nodes needs to be connected in the solution tree instead of all nodes. For the RDCSTP, using stabilized branch-and-price, Leitner et al. [57, 58] have been able to outperform other approaches to solve this problem (i.e. layered graphs) in many cases. Since the RDCMST is the special case of the RDCSTP, where every node is a terminal node, the stabilized branch-and-price algorithm proposed for the RDCSTP should also be competitive for the RDCMST.

When not restricting ourselves to branch-and-price methods, we get a broader picture. In particular, one can identify three problems, which are strongly related to the STPRBH: First and foremost, there are the *Steiner tree problem with hop constraints* (STPH) and the *Steiner tree problem with revenues and budget* (STPRB), which is just the STPRBH without revenues, budget and hop constraints, respectively. Moreover there is the *prize collecting Steiner tree problem* (PCSTP), also known as the *Steiner tree problem with profits* (STPP), where the goal is to maximize the difference between revenues and edge costs, instead of having a budget constraint on the edge costs. The *node-weighted Steiner tree problem* (NWSTP) is a slight variant of the PCSTP, where a given set of terminal nodes must be part of any feasible solution (often this set only consists of a single node, i.e. a root node).

The NWSTP has first been introduced by Segev [72]. Segev uses subgradient optimization and Lagrangian relaxation to obtain lower bounds, which are used in a branch-and-bound algorithm. Magnanti and Wolsey give various ILP formulations for the NWSTP in [62] and compare the values of the LP-relaxations for these formulations. Klau et al. [50] use a combination of a memetic algorithm and integer linear programming to solve the PCSTP. There exist branch-and-cut algorithms to solve the PCSTP by Lucena et al. [60] and by Ljubic et al. [59]. In [14] Costa et al. present a survey of different variants of the PCSTP and in [10], Chapvoska and Punnen give a survey on other variants of the PCSTP.

In [77], Voß discusses a tabu search for the STPH. Gouveia uses variable redefinition in [39] to strengthen a multicommodity flow model for the STPH and the related MSTH. For the MSTH more work has been done: Aside from the already mentioned use of Miller-Zemlin Tucker constraints [37], Gouveia et al. describe Lagrangian relaxation approaches in [42] and an approach based on layered graphs in [43]. Moreover Gouveia discusses multicommodity flow models in [38] and there is a general survey by Dahl et al. [18] for this problem.

Concerning approximation algorithms for variants of the Steiner tree problems, Bienstock et

al. [8] mention the PCSTP in a paper about the *prize collecting traveling salesman problem* and present a factor tree approximation algorithm based on LP-rounding. Goemans and Williamson give a factor two approximation algorithm for the PCSTP in [34]. Klein and Rawi [51] present a $2\ ln\ |T|$ approximation algorithm for the NWSTP, $|T|$ denotes the number of terminals. Johnson al. in [46] give a factor $(5 + \epsilon)$ approximation algorithm for the STPRB.

Previous work for stabilization and acceleration of column generation and branch-and-price is discussed in Chapter 5.

CHAPTER 4

# ILP Formulations for the STPRBH

In this chapter we will describe several different ILP formulations for the STPRBH. These formulations use variables corresponding to hop-constrained paths, i.e. each variable models a path between the root node and some terminal consisting of at most $H$ arcs or edges. Since there can be exponentially many paths in a graph, these formulations may have exponentially many variables and will therefore be solved using branch-and-price. When maximizing the revenue, there may exist optimal solutions to the presented formulations which do not specify a tree. Thus we present an alternative objective function, which guarantees that the solutions are trees as well as postprocessing techniques, which can transform the non-tree solutions in tree solutions with the same objective value. Moreover, the pricing subproblem is also discussed.

## 4.1 Undirected Path-Formulation

We start with an undirected formulation, although it is well-known that undirected formulations for Steiner tree problems are usually weaker than their directed counterparts [11, 12, 52, 70]. The latter are based on modeling the problem as Steiner arborescence. However, undirected formulations need only roughly half the number of variables compared with directed ones, since the edges have to be doubled to arcs in a directed formulation.

In the following, binary variables $y_v, \forall v \in V$, indicate if a node $v \in V$ is connected to the root in a solution and binary variables $x_e, \forall e \in E$, denote if an edge $e \in E$ is part of the solution. A path $p$ is a subset of the edges, i.e. $p \subseteq E$. The set of hop-constrained paths from the root node 0 to a terminal $t \in T$ is denoted with $P_t$, i.e.

$$P_t = \{p \subseteq E \mid p \text{ forms a path from 0 to t with } |p| \leq H)\}.$$

Moreover, the set of all hop-constrained paths from the root node to the terminals is denoted with $P$, i.e. $P = \bigcup_{t \in T} P_t$. Variables $\lambda_p, \forall p \in P$, denote, if path $p$ is realized in a solution.

Using this notation, we get the following master problem:

$$\text{(UPF)} \quad \max \sum_{t \in T} r_t y_t \tag{4.1}$$

$$\text{s.t.} \quad y_t - \sum_{p \in P_t} \lambda_p \leq 0 \qquad \forall t \in T \tag{4.2}$$

$$\sum_{p \in P_t | e \in p} \lambda_p - x_e \leq 0 \qquad \forall t \in T,\ \forall e \in E \tag{4.3}$$

$$x_e - y_v \leq 0 \qquad \forall e = \{u, v\} \in E \tag{4.4}$$

$$\sum_{v \in V} y_v - \sum_{e \in E} x_e = 1 \tag{4.5}$$

$$\sum_{e \in E} c_e x_e \leq B \tag{4.6}$$

$$y_v \in \{0, 1\} \qquad \forall v \in V \tag{4.7}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \tag{4.8}$$

$$\lambda_p \geq 0 \qquad \forall p \in P \tag{4.9}$$

The objective function (4.1) maximizes the revenue of all connected terminals. The convexity constraints (4.2) ensure that there is at least one path connecting the root with a terminal in the solution and the coupling constraints (4.3) link the paths with the edges used by them. Constraints (4.4) are linking constraints between edge and node variables, while (4.5) together with the fact that every path contains the root node, ensures that the graph induced by a solution contains a feasible solution. The budget constraint (4.6) ensures that the total costs due to the selected edges do not exceed the given budget B.

Variables $\lambda_p$ are only restricted to be nonnegative, because in an optimal solution they are binary anyway due to the rest of the formulation. This is for the following reasons: First, any value bigger than 1 for any $\lambda_p$ is not possible due to (4.3). Moreover, suppose the value of some $\lambda_p$ is fractional. Then there must be another $\lambda_{p'}$ to the same terminal $t$, to fulfill (4.2), since $y_t$ is binary. This means, there must be another path $p'$ to terminal $t$, and such a path $p'$ must contain at least two different edges than $p$, so at least two more $x_e$ must be 1 due to (4.3). However, both paths start in the same node (i.e. the root node) and end in the same node (i.e. terminal $t$), therefore one less $y_v$ is set to 1 in (4.4) than $x_e$ are set to 1. This results in a violation of (4.5).

Formulation (UPF) and all following formulations do not specify a tree, because there can be cycles consisting of edges, which are part of no path. These cycles form connected components without the root node and thus can be easily dealt with in postprocessing (i.e. only keep the connected component, which contains the root). Moreover, there can be leaves, which are non-terminals, i.e. unnecessary, but a solution containing such leaves does not violate the definition of the STPRBH.

The possibility of obtaining non-tree feasible solutions when using path-based formulations for the STPRBH is in contrast to path-formulations of other Steiner tree problems like the RD-CMSTP [57, 58]. This is easily explained by the fact, that in such problems, the objective is to minimize the total edge costs. Thus the optimal solution does not contain redundant edges, i.e.

is a tree (assuming all the edge-costs are positive).

Due to this observation, we can guarantee a tree solution in our non-tree formulations by replacing the original objective function with the following one:

$$\max \sum_{t \in T} r_t y_t - \sum_{e \in E} \epsilon x_e$$

If a small enough value for $\epsilon > 0$ is chosen, every optimal solution for the original objective value is also an optimal solution for the modified one. Moreover, since $-\sum_{e \in E} \epsilon x_e$ is part of the modified objective function, the solution will not contain any redundant edges and thus is a tree. By replacing the edges with arcs, this technique also works for the directed formulations, which will be presented in the next section.

Since the set of feasible paths $P$ is exponentially large, we cannot solve $(\mathrm{UPF})$ directly, but use column generation for solving its linear relaxation. Therefore, we start with a small subset of all path-variables $\bigcup_{t \in T} \tilde{P}_t = \tilde{P} \subseteq P$. At least one path must be in $\tilde{P}$ in the beginning (e.g. the empty path connecting the root node to itself), otherwise we can get an infeasible starting solution.

Further variables are added on demand according to the solution of the pricing subproblem. The column generation is then embedded in a branch-and-bound procedure to solve $(\mathrm{UPF})$, i.e. we use branch-and-price.

The restricted master problem $(\mathrm{UPF})^{(\mathrm{RMP})}$ is defined on the path-variables in $\tilde{P}$ and the upper bounds on variables $x_{ij}$ are dropped, but otherwise corresponds to the linear relaxation of $(\mathrm{UPF})$. The dual variables to the constraints are given in parentheses.

$$(\mathrm{UPF})^{(\mathrm{RMP})} \quad \max \sum_{t \in T} r_t y_t \tag{4.10}$$

$$\text{s.t.} \quad y_t - \sum_{p \in \tilde{P}_t} \lambda_p \leq 0 \qquad (\mu_t) \quad \forall t \in T \tag{4.11}$$

$$\sum_{p \in \tilde{P}_t | e \in p} \lambda_p - x_e \leq 0 \qquad (\pi_e^t) \quad \forall t \in T,\ \forall e \in E \tag{4.12}$$

$$x_e - y_v \leq 0 \qquad (\alpha_e) \quad \forall e = \{i, v\} \in E \tag{4.13}$$

$$\sum_{v \in V} y_v - \sum_{e \in E} x_e = 1 \qquad (\gamma) \tag{4.14}$$

$$\sum_{e \in E} c_e x_e \leq B \qquad (\beta) \tag{4.15}$$

$$0 \leq y_v \leq 1 \qquad (\zeta_v) \quad \forall v \in V \tag{4.16}$$

$$x_e \geq 0 \qquad \forall e \in E \tag{4.17}$$

$$\lambda_p \geq 0 \qquad \forall p \in \tilde{P} \tag{4.18}$$

To define the pricing subproblem, let us start with the generic formula for reduced costs: $\bar{c}_j = c_j - \pi^T A_j$ (remember that $\pi$ stands for the complete vector of dual solutions and $A_j$ for

the $j-th$ column in the coefficient matrix $A$ of the MP). We need to find the reduced costs $\bar{c}_p$ of path-variables $\lambda_p$ corresponding to $p \in P_t \setminus \tilde{P}_t$. Since the coefficients of $\lambda_p$ in the objective function is zero, $c_p$ is zero. Moreover, the first dual variable of every path $p \in P_t$ is $\mu_t$ and its coefficient is minus one, see (4.2). Furthermore, the coefficient in every constraints (4.3) corresponding to an edge $e \in p$ with dual variable $\pi_e^t$, the coefficient of $\lambda_p$ is one. Inserting this in the generic formula gives us the reduced costs $\bar{c}_p$ for variables $p \in P_t \setminus \tilde{P}_t$, for some terminal $t \in T$ as

$$\bar{c}_p = \mu_t - \sum_{e \in p} \pi_e^t.$$

In the pricing subproblem, we need to identify a variable $p \in P_t$ with positive reduced costs, since we have a maximization problem. We might as well search for the variable with most positive reduced costs. Thus, we need to compute

$$(t^*, p^*) = \text{argmax}_{t \in T, p \in P_t} \mu_t - \sum_{e \in p} \pi_e^t$$

Hence, the variable with maximal reduced costs for a terminal $t \in T$ can be determined by computing a cheapest feasible path from 0 to $t$ using edge costs $\pi_e^t$. Since all edge costs are strictly nonnegative, this hop-constrained cheapest path problem can be solved in polynomial time. The problem is covered in detail in Section 4.3. If at least one variable with positive reduced costs does exist, we can add it to $(\text{UPF})^{(\text{RMP})}$ which in turn needs to be resolved.

## 4.2 Directed Path-Formulation

In the directed path formulation, the undirected graph gets replaced by a directed graph by introducing two arcs for every edge, i.e. $A = \{(u,v),(v,u) \mid \{u,v\} \in E\}$. Additionally, a correspondingly defined cost function $c_{ij} = c_{ji} = c_e, \forall e = \{i,j\} \in E$ is used. The binary variables $x_{ij}, \forall (i,j) \in A$, indicate whether an arc is part of a directed solution. Moreover, the binary variables $y_v, \forall v \in V$, indicate if a node $v$ is part of a solution.

The set of all directed hop-constrained paths from 0 to $t \in T$ is denoted with $P_t$. Each of $P_t$ is represented by its arc set, i.e.

$$P_t = \{p \subseteq A \mid p \text{ forms a directed path from 0 to } t \text{ with } |p| \leq H)\}.$$

Again, path-variables $\lambda_p, \forall p \in P$, indicate if a hop-constrained path $p$ is realized in the solution.

We get the following master problem $(\text{DPFT})$:

$$\text{(DPFT)} \quad \max \sum_{t \in T} r_t y_t \tag{4.19}$$

$$\text{s.t.} \quad y_t - \sum_{p \in P_t} \lambda_p \leq 0 \qquad \forall t \in T \tag{4.20}$$

$$\sum_{p \in P_t | (i,j) \in p} \lambda_p - x_{ij} \leq 0 \qquad \forall t \in T,\ \forall (i,j) \in A \tag{4.21}$$

$$\sum_{(i,j) \in A} x_{ij} - y_v \leq 0 \qquad \forall v \in V \tag{4.22}$$

$$\sum_{v \in V} y_v - \sum_{(i,v) \in A} x_{iv} \leq 1 \tag{4.23}$$

$$\sum_{(i,j) \in A} c_{ij} x_{ij} \leq B \tag{4.24}$$

$$y_v \in \{0,1\} \qquad \forall v \in V \tag{4.25}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{4.26}$$

$$\lambda_p \geq 0 \qquad \forall p \in P \tag{4.27}$$

The objective function (4.19) is the same as in the undirected formulation. Constraints (4.20) are the convexity constraints and Constraints (4.21) are the coupling constraints, both with the same meaning as in the undirected formulation. Constraints (4.22) make sure that any node has at most one incoming arc and together with Constraint (4.23) and the rest of the formulation, these constraints make sure that in the graph induced by a solution, the connected component containing the root node is a tree. Again, we have the budget constraints (4.24) and variables $\lambda_p,\ \forall p \in P$, have no upper bounds since they automatically become integral.

We now present some formulations, which do not use variables $y_v, \forall v \in V \setminus T$. We begin with (DPF), a slight modification of (DPFT):

$$\text{(DPF)} \quad \max \sum_{t \in T} r_t y_t \tag{4.28}$$

$$\text{s.t.} \quad y_t - \sum_{p \in P_t} \lambda_p \leq 0 \qquad \forall t \in T \tag{4.29}$$

$$\sum_{p \in P_t | (i,j) \in p} \lambda_p - x_{ij} \leq 0 \qquad \forall t \in T,\ \forall (i,j) \in A \tag{4.30}$$

$$\sum_{(i,t) \in A} x_{it} - y_t \leq 0 \qquad \forall t \in T \tag{4.31}$$

$$\sum_{(i,j) \in A} x_{ij} \leq 1 \qquad \forall j \in V \setminus T \tag{4.32}$$

$$\sum_{(i,j) \in A} c_{ij} x_{ij} \leq B \tag{4.33}$$

$$y_t \in \{0,1\} \qquad \forall t \in T \tag{4.34}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{4.35}$$

$$\lambda_p \geq 0 \qquad \forall p \in P \tag{4.36}$$

In contrast to $(\mathrm{DPFT})$, Constraint (4.23) is dropped and Constraints (4.22) are replaced by Constraints (4.31) and (4.32), respectively, since variables $y$ are defined on terminal nodes only. Formulation $(\mathrm{DPF})$ does not describe a tree for the connected component containing the root node, because there can be an incoming arc to the root node in a feasible solution. This can be easily dealt with in postprocessing or by replacing (4.31) with $\sum_{(i,0) \in A} x_{i0} = 0$.

When dropping the Inequalities (4.31) and (4.32), we get a formulation where feasible solutions can contain more than one path to a terminal (in addition to the possible non-tree solutions of $(\mathrm{DPF})$). However, this does not change the objective function value and we can easily transform such a non-tree solution into a tree solution, e.g. by deleting all incoming arcs of a terminal, except the one lying on the path with the fewest arcs to this terminal. The formulation will be denoted $(\mathrm{DPFnt})$.

The LP-relaxation of $(\mathrm{DPF})$ can be strengthened by adding constraints (4.37) to it.

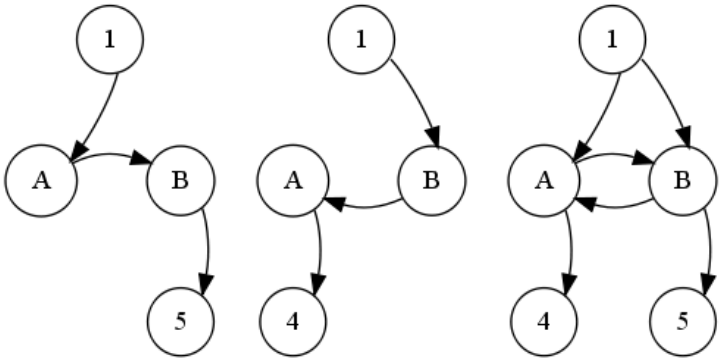
$$x_{ij} + x_{ji} \leq 1 \quad \forall \{i,j\} \in E \tag{4.37}$$

Next, we show that constraints (4.37) added to $(\mathrm{DPF})$ give a stronger formulation according to Definition 2.13. In the following proof, Figure 4.1 will be used. In this figure, node 1 is the root node, nodes 4 and 5 are terminal nodes and nodes $A$ and $B$ are Steiner nodes. Let $\mathcal{P}_{(4.37)}$ denote the polyhedron associated with the LP-relaxation of $(\mathrm{DPF})$ with constraints (4.37) and $\mathcal{P}_{(\mathrm{DPF})}$ be the polyhedron associated with the LP-relaxation of $(\mathrm{DPF})$.

**Proposition 4.1.** *Constraints* (4.37) *added to* $(\mathrm{DPF})$ *give a stronger formulation.*

*Proof.* First, observe that $\mathcal{P}_{(4.37)} \subseteq \mathcal{P}_{(\mathrm{DPF})}$, since the latter is the same formulation, but without Constraints (4.37). We now show that this inclusion is strict, by giving a point, which lies in $\mathcal{P}_{(\mathrm{DPF})}$, but not in $\mathcal{P}_{(4.37)}$.

Consider a path $\{(1, A), (A, B), (B, 5)\}$ with associated path-variable $\lambda_1$ to terminal $5$ and a path $\{(1, B), (B, A), (A, 4)\}$ to terminal $4$ with path-variable $\lambda_2$. Suppose $y_4 = y_5 = 0.1$ and also $\lambda_1 = \lambda_2 = 0.1$, so Constraints (4.29) are satisfied. The arc-variables have the following values: $x_{1A} = x_{1B} = x_{A4} = x_{B5} = 0.1$ and $x_{BA} = x_{AB} = 0.9$. Clearly, constraints (4.30) also hold. Moreover, constraints (4.48) and (4.49) also hold, so this point is in $\mathcal{P}_{(\mathrm{DPF})^{(\mathrm{RMP})}}$. However, it is not in $\mathcal{P}_{(4.37)}$, because $x_{AB} + x_{BA} \geq 1$. □




(a) Path $p_1$ corresponding to $\lambda_1$ (b) Path $p_2$ corresponding to $\lambda_2$ (c) Graph induced by $p_1$ and $p_2$

Figure 4.1: Paths and graph for the proof $\mathcal{P}_{(4.37)} \subset \mathcal{P}_{DPF}$

Note that this does not necessarily imply that $(\mathrm{DPF})$ with (4.37) gives a better LP-relaxation value than $(\mathrm{DPF})$.

Moreover, it is possible to give a directed formulation of the problem without using variables $y_t, \forall t \in T$, by replacing the $y_t$ by a convex combination of all incoming edges to the terminal $t$. This gives the following master problem $(\mathrm{DPF2})$:

$$\text{(DPF2)} \quad \max \sum_{t \in T \setminus \{0\}} r_t \sum_{(i,t) \in A} x_{it} + r_0 \tag{4.38}$$

$$\text{s.t.} \quad \sum_{(i,t) \in A} x_{it} - \sum_{p \in P_t} \lambda_p \leq 0 \qquad \forall t \in T \tag{4.39}$$

$$\sum_{p \in P_t | (i,j) \in p} \lambda_p - x_{ij} \leq 0 \qquad \forall t \in T,\ \forall (ij) \in A \tag{4.40}$$

$$\sum_{(i,j) \in A} x_{ij} \leq 1 \qquad \forall j \in V \tag{4.41}$$

$$\sum_{(i,j) \in A} c_{ij} x_{ij} \leq B \tag{4.42}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{4.43}$$

$$\lambda_p \geq 0 \qquad \forall p \in P \tag{4.44}$$

Note that this time, Constraints (4.41) (a modification of (4.31) and (4.32)) can not be dropped, because variables $x_{ij}$ are part of the objective function.

Naturally, we are facing the same problem as in the undirected formulation, i.e. the set $P$ of feasible paths can be exponentially large, so branch-and-price needs to be used. The restricted master problem $(\text{DPF})^{(\text{RMP})}$ for (DPF) is described in the following. Note that we can again drop the upper bound on the $x_{ij}$ variables like in the undirected case.

$$(\text{DPF})^{(\text{RMP})} \quad \max \sum_{t \in T} r_t y_t \tag{4.45}$$

$$\text{s.t.} \quad y_t - \sum_{p \in \tilde{P}_t} \lambda_p \leq 0 \qquad (\mu_t) \quad \forall t \in T \tag{4.46}$$

$$\sum_{p \in \tilde{P}_t | (i,j) \in p} \lambda_p - x_{ij} \leq 0 \qquad (\pi_{ij}^t) \quad \forall t \in T,\ \forall (i,j) \in A \tag{4.47}$$

$$\sum_{(i,t) \in A} x_{it} - y_t \leq 0 \qquad (\alpha_t) \quad \forall t \in T \tag{4.48}$$

$$\sum_{(i,j) \in A} x_{ij} \leq 1 \qquad (\alpha_j) \quad \forall j \in V \setminus T \tag{4.49}$$

$$\sum_{(i,j) \in A} c_{ij} x_{ij} \leq B \qquad (\beta) \tag{4.50}$$

$$0 \leq y_t \leq 1 \qquad (\zeta_t) \quad \forall t \in T \tag{4.51}$$

$$x_{ij} \geq 0 \qquad \forall (i,j) \in A \tag{4.52}$$

$$\lambda_p \geq 0 \qquad \forall p \in \tilde{P} \tag{4.53}$$

The RMP for (DPF2) is

$$
\begin{aligned}
(\mathrm{DPF2})^{(\mathrm{RMP})} \quad & \max \sum_{t\in T\setminus\{0\}} r_t \sum_{(i,t)\in A} x_{it} + r_0 && && && (4.54)\\
\text{s.t.} \quad & \sum_{(i,t)\in A} x_{it} - \sum_{p\in P_t} \lambda_p \le 0 && (\mu_t) && \forall t \in T && (4.55)\\
& \sum_{p\in \tilde{P}_t | (i,j)\in p} \lambda_p - x_{ij} \le 0 && (\pi^t_{ij}) && \forall t \in T,\ \forall (i,j) \in A && (4.56)\\
& \sum_{(i,j)\in A} x_{ij} \le 1 && (\alpha_j) && \forall j \in V && (4.57)\\
& \sum_{(i,j)\in A} c_{ij} x_{ij} \le B && (\beta) && && (4.58)\\
& 0 \le x_{ij} \le 1 && (\eta_{ij}) && \forall (i,j) \in A && (4.59)\\
& \lambda_p \ge 0 && && \forall p \in \tilde{P} && (4.60)
\end{aligned}
$$

For more details, we refer to Sections 4.1 and 4.3, because apart from searching for a directed hop-constrained cheapest path instead of an undirected one, the column generation procedure is the same for undirected and directed formulations.

## 4.3 The Pricing Subproblem: Hop-Constrained Cheapest Path

For solving the pricing subproblem, we need to identify a hop-constrained cheapest path (HCCP) for each terminal $t \in T$, given nonnegative arc-costs $c_{ij} \ge 0, \forall (i,j) \in A$ . The HCCP is a special case of the shortest path problem with resource constraints (SPPRC), which in general is $\mathcal{NP}$-hard [41, 45]. A good overview of this topic can be found in [28, 45]. For the HCCP with nonnegative costs, polynomial time algorithms do exist. Let $f(i,h)$ denote the cost of the cheapest path from node 0 to node $i$ which uses exactly $h$ hops, the following well-known recurrence [41], can be used to compute $f(j,h), \forall j \in V, 1 \le h \le H$:

$$
f(j,h) = \begin{cases} 0 & j = 0, 0 \le h \le H \\ min_{(i,j)\in E}\{c_{ij} + f(i,h-1)\} & \forall j \in V \setminus \{0\}, 1 \le h \le H \end{cases} \tag{4.61}
$$

We use Algorithm 2, a slightly modified version of the dynamic-programming algorithm presented in [41], which runs in time $O(H|E|)$. Note that this a polynomial algorithm, because a hop limit $H > |V|$ does not make sense. The algorithm does not solve the recurrence for all pairs of nodes and hops, since this is not necessary in our case, because for a set of edge-costs $c$, we only need to find a hop-constrained cheapest path from the root node to one particular terminal node $k$. Note that we only consider hop-constrained cheapest paths starting from the root node and thus automatically assume for the rest of the thesis that such a path starts at the root node without specifically mentioning it.

In Algorithm 2, the set $S_h$ contains the nodes reachable in $h$ hops and $mC_i$ saves the cost of the cheapest path from the root 0 to the node $i$. In $p(i,h)$, the predecessor of node $i$ for a path with $h$ hops to node $i$ is stored and in $h_{cheapest}$, the hops of the cheapest path to node $k$ is saved. This data is necessary for reconstruction of the path, once the algorithm has finished. Note that $c$ in the input is a vector of arc-costs, the cost for an arc $(i,j)$ is extracted with $c_{ij}$.

After the initialization in lines 2-9, we update $mC_i$ (lines 10-15), if the cost of the path with $h$ hops to node $i$ is not bigger than the minimal cost of the path to node $k$ found so far and is cheaper than any other path to node $j$ with less than $h$ hops.

Next (lines 16-32), we check for every node in the set $S_h$, if we can use one of the outgoing arcs of $i$ to find a path to a node $j$, which is cheaper than any other path to $j$ found so far. Moreover, we check also if this new path has costs smaller than $mC_k$, because if the cost is bigger, this path has no use in finding a cheapest path to $k$. The same holds, if the path costs more as the externally given upper bound $max$. If we found a cheaper path, we update $S_{h+1}$ and set $f(j,h+1)$ to the cost of this path. Moreover, if the end node is $k$, we also set $mC_k$ to the cost of the path. This is necessary, to get the correct value $mC_k$, if the cheapest path has exactly $H$ hops, because we do not get back to lines 10-15 anymore, since the loop terminates.

The presented algorithm is for an undirected graph, but can also used for a directed graph by changing the edge set $E$ to the arc set $A$ in line 18. The cheapest path can easily by reconstructed after the algorithm has finished by backtracking the $p(i,h)$ starting from $p(k,h_{cheapest})$. A good upper bound can speed up the algorithm considerably. In our case, such an upper bound is provided with $\mu_t$, because if the cheapest path costs more than $\mu_t$, it cannot result in positive reduced costs and therefore is of no use. More details on pricing can be found in Chapter 5, where the branch-and-price algorithm is discussed.

**Algorithm 2** Hop-constrained cheapest path

```
1: procedure HCCP(G = (V, E),H,k,c,max)
2:     S_h ← ∅, 1 ≤ h ≤ H
3:     S_0 ← {0}
4:     mC_i ← ∞, 1 ≤ i ≤ |V|
5:     mC_0 ← 0
6:     f(i, h) ← ∞, 1 ≤ i ≤ |V|, 0 ≤ h ≤ H
7:     f(0, h) ← 0, 0 ≤ h ≤ H
8:     h_cheapest ← ∞
9:     p(i, h) ← ∞, 0 ≤ i ≤ |V|, 0 ≤ h ≤ H
10:    for h = 0 to H − 1 do
11:        for i ∈ S_h do
12:            if f(i, h) ≤ mC_k ∧ f(i, h) < mC_i then
13:                mC_i ← f(i, h)
14:            end if
15:        end for
16:        for i ∈ S_h do
17:            if f(i, h) ≤ mC_k then
18:                for {i, j} ∈ E do
19:                    cost ← c_ij + f(i, h)
20:                    if cost < mC_j ∧ cost < mC_k ∧ cost < f(j, h + 1) ∧ cost < max then
21:                        S_{h+1} ← S_{h+1} ∪ {j}
22:                        f(j, h + 1) ← cost
23:                        p(j, h + 1) ← i
24:                        if j == k then
25:                            mC_k ← cost
26:                            h_cheapest ← h + 1
27:                        end if
28:                    end if
29:                end for
30:            end if
31:        end for
32:    end for
33:    path p = ∅
34:    if mC_k ≤ max then
35:        i ← k
36:        h ← h_cheapest
37:        while i ≠ 0 do
38:            pre ← p(i, h)
39:            p ← p ∪ (pre, i)
40:            i ← pre
41:            h ← h − 1
42:        end while
43:    end if
44:    return p
45: end procedure
```

CHAPTER 5

# The Branch-and-Price Algorithm

In this chapter, the branch-and-price approach, which has been developed to solve instances of the STPRBH using the formulations presented in the previous chapter, is discussed. A focus lies on methods to accelerate the column generation/branch-and-price algorithm, since such an approach usually suffers from some well-known computational problems [3, 24], especially when solving the LPs is based on the simplex method [61, 75]. We will shortly discuss some of the problems according to the classification given by Vanderbeck in [75] in the following.

The *"tailing-off"* effect denotes a slow-convergence of the procedure. This means, a near-optimal solution is found very fast in most cases and then many iterations are needed to find the optimum solution and to prove optimality.

At the start of the procedure, many irrelevant columns are frequently added, called the *"heading-in"* effect. This happens, since the pricing subproblem uses the dual information to price in columns and in the beginning we do not have much useful information (i.e. columns, which are present in a good/optimal solution) in the RMP.

Also, the values of the dual variables may oscillate during the procedure and only get the dual-optimal value for (the dual of) the MP in the last iterations of the procedure. This behavior is denoted as *"bang-bang"* effect.

## 5.1 Discussion of the Dual Problem

In [61] it is pointed out that a careful analysis of the dual of the RMP can often give valuable insight for the column generation procedure of the primal. In this spirit, we will give a general discussion of the dual problem, before we move on to detailed presentations of the methods used in the branch-and-price framework. It turns out that the dual of $(\mathrm{DPFnt})^{(\mathrm{RMP})}$ has an interesting economical interpretation. We will denote the dual by $(\mathrm{DUAL})^{(\mathrm{DPFnt})}$:

$$(\mathrm{DUAL})^{(\mathrm{DPFnt})} \quad \min \sum_{t \in T} \zeta_t + \beta B \tag{5.1}$$

$$\text{s.t. } \mu_t + \zeta_t \geq r_t \qquad (y_t) \quad \forall t \in T \tag{5.2}$$

$$\sum_{(ij) \in p} \pi_{ij}^t \geq \mu_t \qquad (\lambda_p) \quad \forall t \in T,\ \forall p \in \tilde{P}_t \tag{5.3}$$

$$c_{ij}\beta \geq \sum_{t \in T} \pi_{ij}^t \qquad (x_{ij}) \quad \forall (i,j) \in A \tag{5.4}$$

$$\mu_t \geq 0 \qquad \forall t \in T \tag{5.5}$$

$$\pi_{ij}^t \geq 0 \qquad \forall t \in T,\ \forall (i,j) \in A \tag{5.6}$$

$$\beta \geq 0 \tag{5.7}$$

$$\zeta_t \geq 0 \qquad \forall t \in T \tag{5.8}$$

In $(\mathrm{DUAL})^{(\mathrm{DPFnt})}$, we just stated (5.2) and (5.3) in a more "natural" way than $-\mu_t + \sum_{(i,j) \in p} \pi_{ij}^t \geq 0$ and $-\sum_{t \in T} \pi_{ij}^t + c_{ij}\beta \geq 0$ respectively. Two things are instantly clear: First, the objective value is as least as big as the revenue of the root node 0, because there is no path $p$ from 0 to 0, therefore we have $0 \geq \mu_0$ in (5.3), which in turn means that $\zeta_0 \geq r_0$ and $\zeta_0$ is part of the objective function. Second, for every hop-constrained path $p$ to a terminal $t$ the sum of variables $\pi_{ij}^t$ corresponding to the arcs $(i,j)$ in the path has to be bigger than or equal to $\mu_t$. This of course corresponds to the pricing rule of the column generation for the primal. So both of these two easy observations do not provide us with any new insight for solving the primal.

Therefore, let us give some interpretation of $(\mathrm{DUAL})^{(\mathrm{DPFnt})}$, to get some "feeling" for the stated problem: Suppose the terminals $t$ are some goods you want to sell and every good production cost $r_t$. Depending on the price $\mu_t$ set for a good, you will incur a loss of $\zeta_t$ (see (5.2)), so one part of the objective function is minimizing this loss, i.e. the sum of the $\zeta_t$. The arcs $(i,j) \in A$ are potential buyers, where the lhs of constraints (5.4) gives the money a buyer is willing to spend on all goods combined. Variable $\pi_{ij}^t$ gives the price you propose to them for good $t$.

A hop-constrained path to a terminal $t$ corresponds to a group of buyers, who potentially buy a good $t$. You can only be sure to sell a good $t$ for the price $\mu_t$ if every group of potential buyers (i.e. every hop-constrained path) pays you at least $\mu_t$ (see (5.3)). The variable $\beta$ can be interpreted as some kind of "marketing-parameter", with which the money buyers are willing to spend on the goods can be increased (see (5.4)), so you can sell more goods/sell goods for a higher price. This decreases your loss from $\sum_{t \in T} \zeta_t$, however you have to pay the marketing department $\beta b$ and this is also part of the objective function.

Essentially the problem consists of finding a tradeoff between spending more money for marketing or lowering the prices. This problem can be solved with branch-and-cut by starting out without the constraints (5.3) (since there can be exponentially many) and adding cuts of the form (5.3) by finding HCCPs with the current $\pi_{ij}^t$ as edge weights. If the length of such a HCCP to a terminal $t$ is smaller than the value $\mu_t$, we have found a cut and need to resolve the problem. This should come as no surprise, as it is just the "row generation" equivalent to the column generation in the primal.

The dual program also gives a good intuition for the effect of the budget $B$: A high budget will result in a high objective value, because if we decrease the associated variable $\beta$ (to minimize the term $\beta B$), the lhs of constraints (5.4) decreases. This means because of the $\pi_{ij}^t$ variables, which are upper bound in (5.4) and needed in the cuts (5.3), we need to decrease $\mu_t$ and in turn have to increase $\zeta_t$ (due to (5.2)), to guarantee feasibility. However $\zeta_{ij}$ is part of the objective function with positive coefficients.

In the same spirit, one can also analyze the effect of having many paths through some arcs $(i, j)$ with small costs $c_{ij}$: If many different paths use some arc, the associated variable $\pi_{ij}^t$ will often occur in (5.3) for any terminal $t$. However, the $\pi_{ij}^t$ are upper bound in (5.4), and $c_{ij}$ is part of the upper bound. So if $c_{ij}$ is small, the associated $\pi_{ij}^t$ can not add very much to fulfilling (5.3), so we either have to lower $\mu_t$ or have to increase $\beta$, giving a bigger lhs of (5.4). Note all these three actions increase the objective value, which is not surprising, considering that having many paths through some arcs $(i, j)$ with small weights $c_{ij}$ means we likely can connect many terminals $t$ in the primal. Naturally, the same argumentation works for many paths through arcs with high costs, then we do not need to increase $\beta$ or decrease $\mu_t$ much, giving a smaller objective value.

We now present dual programs of other formulations to illustrate the impact of different formulations of the RMP. Consider the dual of $(\mathrm{UPF})^{(\mathrm{RMP})}$:

$$(\mathrm{DUAL})^{(\mathrm{UPF})} \quad \min \gamma + \sum_{v \in V} \zeta_v + \beta B \tag{5.9}$$

$$\text{s.t. } \mu_t - \sum_{e \in \{i,t\}} \alpha_e - \gamma + \zeta_t \geq r_t \qquad (y_t) \quad \forall t \in T \tag{5.10}$$

$$- \sum_{e \in \{i,v\}} \alpha_e - \gamma + \zeta_v \geq 0 \qquad (y_v) \quad \forall v \in V \setminus T \tag{5.11}$$

$$- \mu_t + \sum_{e \in p} \pi_e^t \geq 0 \qquad (\lambda_p) \quad \forall t \in T,\ \forall p \in \tilde{P} \tag{5.12}$$

$$- \sum_{t \in T} \pi_e^t - \alpha_e + \gamma + c_e \beta \geq 0 \qquad (x_e) \quad \forall e \in E \tag{5.13}$$

$$\mu_t \geq 0 \qquad \forall t \in T \tag{5.14}$$

$$\pi_e^t \geq 0 \qquad \forall t \in T,\ \forall \{i, j\} \in E \tag{5.15}$$

$$\alpha_j \geq 0 \qquad \forall j \in V \setminus T \tag{5.16}$$

$$\beta \geq 0 \tag{5.17}$$

$$\gamma \geq 0 \tag{5.18}$$

$$\zeta_v \geq 0 \qquad \forall v \in V \tag{5.19}$$

Note that the dual of $(\mathrm{DPFT})$ is almost the same as $(\mathrm{DUAL})^{(\mathrm{UPF})}$, since $(\mathrm{DPFT})$ is a directed version of $(\mathrm{UPF})$ and thus we do not present it.

The dual of $(\mathrm{DPF})^{(\mathrm{RMP})}$ is

$$(\mathrm{DUAL})^{(\mathrm{DPF})} \quad \min \sum_{j \in V \setminus T} \alpha_j + \sum_{t \in T} \zeta_t + \beta B \tag{5.20}$$

$$\text{s.t. } \mu_t - \alpha_t + \zeta_t \geq r_t \qquad (y_t) \quad \forall t \in T \tag{5.21}$$

$$-\mu_t + \sum_{(i,j) \in p} \pi_{ij}^t \geq 0 \qquad (\lambda_p) \quad \forall t \in T,\ \forall p \in \tilde{P} \tag{5.22}$$

$$-\sum_{t \in T} \pi_{ij}^t + \alpha_j + c_{ij}\beta \geq 0 \qquad (x_{ij}) \quad \forall (ij) \in A \tag{5.23}$$

$$\mu_t \geq 0 \qquad \forall t \in T \tag{5.24}$$

$$\pi_{ij}^t \geq 0 \qquad \forall t \in T,\ \forall (i,j) \in A \tag{5.25}$$

$$\alpha_j \geq 0 \qquad \forall j \in V \setminus T \tag{5.26}$$

$$\beta \geq 0 \tag{5.27}$$

$$\zeta_t \geq 0 \qquad \forall t \in T \tag{5.28}$$

and the dual of $(\mathrm{DPF2})^{(\mathrm{RMP})}$ is

$$(\mathrm{DUAL})^{(\mathrm{DPF2})} \quad \min \sum_{j \in V} \alpha_j + \sum_{(i,j) \in A} \eta_{ij} + \beta B + r_0 \tag{5.29}$$

$$\text{s.t. } -\mu_t + \sum_{(i,j) \in p} \pi_{ij}^t \geq 0 \qquad (\lambda_p) \quad \forall t \in T,\ \forall p \in \tilde{P} \tag{5.30}$$

$$-\sum_{t \in T} \pi_{ij}^t + \alpha_j + c_{ij}\beta + \eta_{ij} + \mu_t \geq r_t \qquad (x_{ij}) \quad \forall (i,j) \in A : j \in T \setminus \{0\} \tag{5.31}$$

$$-\sum_{t \in T} \pi_{ij}^t + \alpha_j + c_{ij}\beta + \eta_{ij} \geq 0 \qquad (x_{ij}) \quad \forall (i,j) \in A : j \in V \setminus T \tag{5.32}$$

$$-\sum_{t \in T} \pi_{i0}^t + \alpha_0 + c_{i0}\beta + \eta_{i0} + \mu_0 \geq 0 \qquad (x_{i0}) \quad \forall (i,0) \in A \tag{5.33}$$

$$\mu_t \geq 0 \qquad \forall t \in T \tag{5.34}$$

$$\pi_{ij}^t \geq 0 \qquad \forall t \in T,\ \forall (i,j) \in A \tag{5.35}$$

$$\alpha_j \geq 0 \qquad \forall j \in V \tag{5.36}$$

$$\beta \geq 0 \tag{5.37}$$

$$\eta_{ij} \geq 0 \qquad \forall (i,j) \in V \tag{5.38}$$

Some observations can be made about the dual programs:

**Proposition 5.1.** *The optimal solution of* $(\mathrm{DUAL})^{(\mathrm{DPFnt})}$ *is not bigger than* $\sum_{t \in T} r_t$.

*Proof.* It is always possible to construct a feasible solution for $(\mathrm{DUAL})^{(\mathrm{DPFnt})}$ in the following way: Set all the $\zeta_t$ to $r_t$ so all constraints (5.2) are fulfilled. The rest of the variables can then be set to 0. The objective function of this solution is $\sum_{t\in T}\zeta_t + 0B0 = \sum_{t\in T} r_t$ □

The same proof strategy also works for the other formulations, in $(\mathrm{DUAL})^{(\mathrm{DPF2})}$, we have to set the values of variables $x_{ij}$ equal to $r_j$ (which is zero, if $j \in V \setminus T$. Naturally, the proven fact is also a obvious consequence of duality.

**Proposition 5.2.** *There is an optimal solution of* $(\mathrm{DUAL})^{(\mathrm{DPFnt})}$*, where* $\beta \leq \frac{\sum_{t\in T} r_t}{B}$.

*Proof.* Follows immediately from the previous proposition: Since the optimal solution of $(\mathrm{DUAL})^{(\mathrm{DPFnt})}$ is bound from above by $\sum_{t\in T} r_t$, we have $\sum_{t\in T} r_t \geq \sum_{t\in T}\zeta_t + \beta B$, so clearly also $\sum_{t\in T} r_t \geq \beta B$. Dividing by $B$ gives us the proposition. □

This proposition also holds for the other formulations.

**Proposition 5.3.** *There is an optimal solution of* $(\mathrm{DUAL})^{(\mathrm{DPFnt})}$*, where* $\mu_t + \zeta_t = r_t, \forall t \in T$

*Proof.* If for some $t$, $\mu_t + \zeta_t > r_t$ and $\mu < r_t$, the solution cannot be optimal, since $\zeta_t$ can be decreased in (5.2) and therefore the value of the objective function gets also decreased. Now consider an optimal solution, where some $\mu_t > r_t$. Since $\zeta_t$ must be nonnegative, we can decrease $\mu_t$ to at least $r_t - \zeta_t$ and (5.2) is still feasible. This does not change the objective value. Moreover, every constraint (5.3) which is feasible for some $\mu'_t > \mu_t$ is also feasible for $\mu_t$, because $\sum_{(ij)\in p} \pi^t_{ij} \geq \mu'_t > \mu_t$. □

Note that this proposition also follows from the fact that in $(\mathrm{DPF})^{(\mathrm{RMP})}$, the $y$ variables can be left without lower bound.

**Proposition 5.4.** *There is an optimal solution of* $(\mathrm{DUAL})^{(\mathrm{DPF2})}$*, where* $\eta_{i,j} = 0, \forall (i,j) \in A$

*Proof.* Observer that both $\alpha_j$ and $\eta_{ij}, \forall j \in V, \forall (i,j) \in A$ are part of the objective function and the same constraints with the same coefficients. Thus, whenever $\eta_{ij} \neq 0$ for some $\forall (i,j) \in A$, $\alpha_j$ can be increased with the value of $\eta_{ij}$ and $\eta_{i,j}$ can be set to zero. □

This essentially means that we need no upper bound on the $x$ variables in the primal problem and can be seen as justification, why we dropped these upper bounds in some of the other RMPs.

## 5.2 Overview of the Branch-and-Price Algorithm and Branching

An overview, how the different techniques presented in the next sections are used in a branch-and-price framework to solve the STPRBH is given Algorithm 3, the function SOLVE used in this algorithm is given in Algorithm 4. Preprocessing of an STPRBH Instance will be discussed in the next section, heuristics to generate an initial solution in Section 5.4, stabilization techniques in Section 5.5 and pricing strategies in Section 5.6. Note that some of the stabilization techniques need to be implemented in different positions compared to where STABILIZATION is located in Algorithm 3.

**Algorithm 3** Branch-and-price-framework

1: **procedure** BRANCH-AND-PRICE($I_{orig} = (G_{orig} = (V_{orig}, E_{orig}), 0, c : E_{orig} \to \mathbb{Q}^+, r : V_{orig} \to \mathbb{Q}^+, B, H)$)
2: $I \leftarrow preprocess(I_{orig})$
3: $F \leftarrow$ formulation of $I$ as directed/undirected integer program
4: $\bar{F} \leftarrow$ RMP of $F$
5: $\bar{F} \leftarrow$ add columns $p$ found by INITIALHEURISTIC(I)
6: $x^* \leftarrow$BRANCH-AND-BOUND using the function SOLVE($\bar{F}$) to solve the LPs at the branch-and-bound nodes
7: **return** $x^*$
8: **end procedure**

**Algorithm 4** Column generation with acceleration methods

1: **function** SOLVE($\bar{F}$)
2: $(\bar{x}^*, \bar{\pi}^*) \leftarrow$ optimal primal and dual solution vector obtained by solving the LP $\bar{F}$
3: $\bar{\pi}^* \leftarrow$ STABILIZATION($\bar{\pi}^*$)
4: $\bar{F} \leftarrow$ add columns found by PRICING($\bar{F}, \bar{\pi}^*$)
5: **end function**

Since the formulations of the STPRBH are integer linear programs, we have to use branch-and-price to solve them. This means column generation to solve the RMP is embedded in a branch-and-bound framework and thus branching decisions have to be made once a RMP (which is a linear program) is solved to optimality. There are many possibilities for branching-schemes in branch-and-price [3, 61, 75].

In our case, we used all the variables, which have to be integral in the MP as branching candidates, i.e. both the arc-variables (edge-variables) $x$ and the terminal-variables $y$. Branching on an arc-variable $x_{ij}$ has the effect of creating two subproblems of the current problem, where in one subproblem the arc $(i, j)$ must be included in a solution and in the other subproblem the arc $(i, j)$ must not be included. When branching on a terminal variable $y_t$, we create one subproblem, where the terminal $t$ must be in any solution and another one, where terminal $t$ must not be in any solution.

## 5.3 Preprocessing of an STPRBH Instance

Given an instance $I_{orig} = (G_{orig} = (V_{orig}, E_{orig}), 0, c : E_{orig} \to \mathbb{Q}^+, r : V_{orig} \to \mathbb{Q}^+, B, H)$ of the STPRBH, some preprocessing can be done. The aim of preprocessing is simply to remove parts of the instance, which cannot be in an optimal solution anyway, thus reducing the number of variables in our ILP formulations, which is likely to result in a faster runtime of the branch-and-price algorithm. Naturally the methods used for preprocessing need to be fast in comparison with the branch-and-price algorithm, otherwise there would be no gain in using preprocessing.

There exist many preprocessing techniques (also called reduction tests) for the Steiner tree problem on graphs [27, 52], the PCSTP [73] and variants thereof [14]. However most of these

techniques can not be applied to the STPRBH because they do not work due to the budget or hop constraints.

We present three propositions, which can be used for preprocessing of an STPRBH instance. The first proposition is an adaption of the degree one test [14, 52, 73], while the next two just state the simple facts, that a node, which is not reachable due to the budget or hop limit cannot be part of any solution.

**Proposition 5.5.** *Given an instance $I$ of the STPRBH, all Steiner nodes (and their incident edge) with degree one can be removed.*

**Proposition 5.6.** *Given an instance $I$ of the STPRBH, all vertices $v \in V$ (and incident edges $\{i, v\} \in E$), where the cheapest path from the root node $0$ to $v$ has costs greater than $B$, can be removed.*

**Proposition 5.7.** *Given an instance $I$ of the STPRBH, all vertices $v \in V$ (and incident edges $\{i, v\} \in E$) can be removed, where the path from the root node $0$ to $v$ with the fewest number of edges has more than $H$ edges, can be removed.*

These vertices can be identified by a breadth-first-search starting from the root node.

The result of this preprocessing is a reduced instance $I = (G = (V, E), 0, c : E \to \mathbb{Q}_0^+, r : V \to \mathbb{Q}_0^+ B, H)$, $V \subseteq V_{orig}, E \subseteq E_{orig}$ with the same optimal solution as $I$.

## 5.4 Heuristics to Find an Initial Solution

At the beginning of the branch-and-price process, an initial feasible solution must be provided [3, 61]. Such a feasible solution can be generated using phase one of the simplex method. However, since artificial variables are used in phase one, we are likely to incur the "heading-in" effect. This happens, because the solution with artificial variables may not resemble the structure of an optimal solution and thus generates poor dual information [61]. Hence it is beneficial to start the process with "good" columns added to the initial RMP [3, 75]. Since we use branch-and-price, all the LPs in the branch-and-bound tree also need feasible solutions to start. Such a starting point is given by using the already generated columns, after deleting columns which have become infeasible due to branching [75].

We now describe simple heuristics to generate meaningful initial columns. All the following algorithms take the graph $G$, terminals $T$, hop limit $H$, budget limit $B$, the arc-cost vector $c$ and the revenue vector $r$ as input.

### A Basic Heuristic

Algorithm 5 is the most basic approach we pursued, it just adds a hop-constrained cheapest path for every terminal to the LP-relaxation, following the intuition that a cheapest path is normally not a bad choice. Note that it is possible that the set of all added paths does not form a feasible solution, however there is at least one subset, which is a feasible solution (in the worst case, this subset consists of a single path).

**Algorithm 5** A basic heuristic

**procedure** CHEAPEST($G = (V, A), T, H, B, c, r$)
  **for** $t \in T$ **do**
    path $p \leftarrow$ HCCP($G = (V, A), t, H, B, c$)
    **if** $p \neq \emptyset$ **then**
      add $\lambda_p$ corresponding to $p$ to the RMP
    **end if**
  **end for**
**end procedure**

## A Greedy Algorithm

The next approach, GREEDY (Algorithm 6) is a more sophisticated method. The terminals are sorted according to revenue and we try adding a hop-constrained cheapest path for every terminal, taking into account the budget and the cost of already added paths. Moreover, once a path has been added, all the arcs in it get cost zero for the calculation of the following hop-constrained cheapest paths, since further paths may use such an arc without increasing the used budget.

**Algorithm 6** A greedy algorithm

1: **procedure** GREEDY($G = (V, A), T, H, B, c, r$)
2: $T' \leftarrow T$ ordered non-increasing according to revenue $r_t$
3: $B' \leftarrow B$
4: $c' \leftarrow c$
5: **for** $x = 1, \ldots, |T|$ **do**
6: $t \leftarrow x$-th entry of $T'$
7: path $p \leftarrow$ HCCP($G = (V, A), t, H, B', c'$)
8: **if** $p \neq \emptyset$ **then**
9: $B' \leftarrow B' - \sum_{(i,j)\in p} c'_{ij}$
10: $c'_{ij} \leftarrow 0, \forall (i, j) \in p$
11: add $\lambda_p$ corresponding to $p$ to the RMP
12: **end if**
13: **end for**
14: **end procedure**

Algorithm 6 stores the terminals sorted nondecreasing according to revenue in $T'$ and sets $t$ to the first entry of $T'$ (i.e. $t$ is the terminal with the highest revenue). The variable $B'$ gets initialized with the budget and the vector $c'$ gets initialized with the arc-costs. Then the algorithm calls HCCP for terminal $t$ using $c'$ as edge-costs and $B'$ as upper-bound for the cost of the path. If such a path $p$ exists (i.e. $p \neq \emptyset$), we decrease the variable $B'$ (i.e. the remaining budget) by the current cost according to $c'$ of the sum of the arcs in $p$, update the cost vector $c'$ by setting the cost of the arcs in $p$ to zero and add the column corresponding to $p$ to the RMP. We repeat this process for every terminal.

### A Heuristic Using Different Hop Constraints

Heuristic HOPSHORTEST (Algorithm 7) solves the hop-constrained cheapest path problem for every terminal $t$ multiple times using different hop limits $h = 1, \ldots, H$. Every time we encounter a hop-constrained shortest path for a hop limit $h$, we add it to the RMP if its cost is cheaper than the cost of the previously found paths (i.e. every time we find a new hop-constrained cheapest path). This technique may have the effect to generate different paths with not many arcs in common to a terminal and therefore generates a broader pool of initial paths.

**Algorithm 7** Heuristic using different hop constraints

```
1:  procedure HOPSHORTEST(G = (V, A), T, H, B, c, r)
2:      for t ∈ T do
3:          mC ← ∞
4:          for h = 1, ..., H do
5:              path p ← HCCP(G = (V, A), t, H, B, c)
6:              if Σ_{(i,j)∈p} c_ij < mC then
7:                  add λ_p corresponding to p to the RMP
8:                  mC ← Σ_{(i,j)∈p} c_ij
9:              end if
10:         end for
11:     end for
12: end procedure
```

### A Heuristic Based on the Knapsack-like Structure of STPRBH

Finally, Algorithm 8, KNAPSACK, tries to exploit the knapsack-like structure of the STPRBH: We need to pack paths to different terminals in a knapsack of size $B$. The profit of a path is the revenue of the associated terminal and the weight of each path corresponds to its cost. In contrast to the classical knapsack problem, the weight of the paths is not fixed, but a monotonically non-increasing function which depends on the paths already in the knapsack, since arcs in an already added path can be used for free in other paths.

Algorithm 8 iteratively packs terminals in the knapsack according to the ratio $r_t/w_t$, where $w_t$ is the cost of the hop constraint cheapest path to terminal $t$ at the current iteration. If there is no path fitting in the knapsack, the algorithm stops. It uses the variable $B'$ for the currently available budget, the set $T'$ of terminals not added, vectors $p$ and $w$ to save the cheapest paths to the terminals and their cost for the current iteration. Moreover, vector $c'$ is used for the current arc-cost.

In every iteration, the hop-constrained cheapest path to every terminal $t$ not in the knapsack and its corresponding weight is calculated. All terminals, which fit in the knapsack are stored in the list $T^*$ and the terminal in $T^*$ with the highest ratio $r_t/w_t$ is chosen (denoted with $t^*$). The path $p_{t^*}$ corresponding to $t^*$ is added to the RMP and the cost of every arc in this path is set to zero (since these arcs are now part of the solution and therefore we incur no further costs, if one

**Algorithm 8** Heuristic based on the knapsack-like structure of the STPRBH

**procedure** KNAPSACK($G = (V, A), T, H, B, c, r$)
  $B' \leftarrow B$
  $T' \leftarrow T$
  $w_t \leftarrow 0, \forall t \in T$
  $p_t \leftarrow \emptyset, \forall t \in T$
  $c' \leftarrow c$
  **repeat**
    **for** $t \in T'$ **do**
      path $p_t \leftarrow$ HCCP($G = (V, A), t, H, B', c'$)
      $w_t \leftarrow \sum_{(i,j) \in p_t} c_{ij}$
    **end for**
    $T^* \leftarrow t \in T' : B' - w_t \geq 0$
    **if** $T^* \neq \emptyset$ **then**
      $t^* \leftarrow argmax_{t \in T^*}\{r_t / w_t\}$
      add $\lambda_{p_t^*}$ corresponding to $p_{t^*}$ to the RMP
      $c'_{ij} \leftarrow 0, \forall (i, j) \in p_{t^*}$
      $B' \leftarrow B' - w_{t^*}$
      $T' \leftarrow T' \setminus \{t^*\}$
    **end if**
  **until** $T^* = \emptyset$
**end procedure**

of them gets used in another path). The available budget gets decreased according to the cost of $p_{t^*}$ and then the next iteration starts. The algorithm stops, if $T^*$ is empty.

## 5.5 Stabilization Techniques

Stabilization techniques for column generation are one method to reduce the effect of the problems described in the beginning of this chapter. These techniques can be separated into general methods and problem-specific methods. The general methods can be further divided into two groups: One group of methods, which do modify the original RMP like the BOXSTEP method [63] and piecewise linear stabilization [5, 7, 26] and one group, where the methods do not change the original RMP like weighted Dantzig-Wolfe decomposition [78], interior-point stabilization [71] or the method of Neame [65]. An example for problem-specific methods are dual-optimal inequalities [6, 61]. The method of alternative dual-optimal solutions [55, 56] lies somewhere between the two groups of problem-specific and general methods. In the following, we will describe the adaption of some of this methods for the STPRBH.

### Alternative Dual-Optimal Solutions

At first, let us consider alternative dual-optimal solutions. The method was introduced in [55, 56] for a survivable network design problem and is further used in [57, 58] for the rooted delay-constrained minimum Steiner tree problem.

To illustrate this method, recall the dual program $(\mathrm{DUAL})^{(\mathrm{DPFnt})}$. We chose this formulation to explain alternative dual-optimal solutions, because it has the smallest number of dual variables and thus allows an easy explanation. The technique also works for the other formulations in a similar way.

Let $(\mu^*, \pi^*, \beta^*, \zeta^*)$ denote the current dual optimal solution computed by an LP solver for the RMP. Moreover, let $A' = \{(i,j) \in A | \nexists p \in \tilde{P} : (i,j) \in p\}$ be the set of arcs which are not part of any path $p$ corresponding to path-variables $\lambda_p$ included in the current RMP. Constraints (5.3) restrict the values of the dual variables of arcs in paths $p \in \tilde{P}$, so only Constraints (5.4) are relevant for arcs in $A'$. This means for arcs $(i,j) \in A'$, any value ${\pi_{ij}^t}^* \geq 0$ is optimal as long as $c_{ij}\beta^* \geq \sum_{t \in T} {\pi_{ij}^t}^*$ holds. Moreover, for any terminal $t$ and arc $(i,j) \in A \setminus A'$ the values of the associated dual variables $\pi_{ij}^t$ can also be increased until constraint (5.4) is tight.

These two facts can be exploited as follows [57, 58]: Let $\delta_{ij} = c_{ij}\beta^* + \sum_{t \in T} {\pi_{ij}^t}^*, \forall (i,j) \in A$, denote the slack of Constraints (5.4). Then any values ${\pi_{ij}^t}' \geq {\pi_{ij}^t}^*, \forall t \in T, \forall (i,j) \in A$, are dual optimal as long as $\sum_{t \in T} {\pi_{ij}^t}' \leq \sum_{t \in T} {\pi_{ij}^t}^* + \delta_{ij}$. However, state-of-the-art LP solvers usually give minimal dual optimal values, i.e. ${\pi_{ij}^t}^* = 0, \forall t \in T, \forall (i,j) \in A'$.

This has some undesirable effects for column generation of the STPRBH: Remember the pricing subproblem, where we have to solve a hop-constrained cheapest path problem with weights ${\pi_{ij}^t}^*$ for the arcs. Thus, if many of these weights are zero, solving the subproblem can yield many paths $p, p'$, where the number of arcs, which are in both paths (i.e. $|p \cap p'|$) is high, so both $p$ and $p'$ are nearly the same paths. In particular, it is possible that we get paths having the same reduced costs, but containing a vastly different number of arcs. Figure 5.1 illustrates this fact, where dotted lines indicate arcs with weight zero.

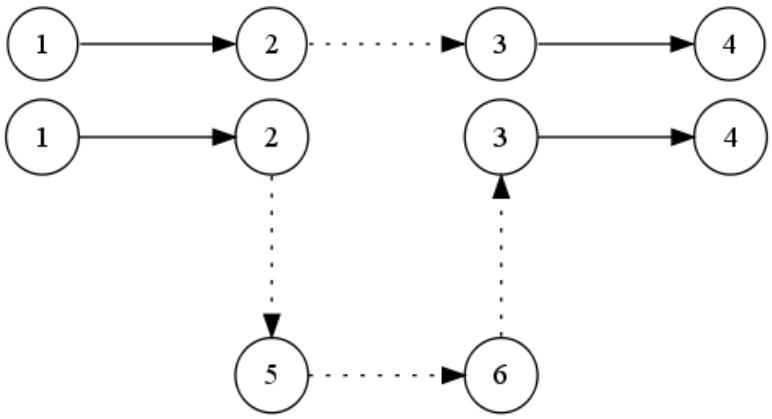



Figure 5.1: Two different paths with the same reduced cost

There is an intuitive explanation from both the primal point of view why such a result of

solving the pricing subproblem may have negative effects: In the primal problem, it is unrealistic that a path $p'$ containing (many) more arcs is better than a path with fewer arcs $p$, when $|p \cap p'|$ contains many arcs (relative to the number of arcs in the path containing fewer arcs), simply because any arc has an associated nonnegative cost $c_{ij}$ and hence a path with more edges has higher cost most of the time.

Therefore, it can be beneficial to use alternative dual-optimal solutions. The alternative dual-optimal solutions are obtained by distributing the slack $\delta_{ij}$ to the relevant dual variables ${\pi_{ij}^t}^*, \forall t \in T$. Two different strategies to do so are proposed in [57, 58]: The first simply distributes $\delta_{ij}$ equally among all relevant dual variables, i.e. dual-solutions $\bar{\pi}_{ij}^t = {\pi_{ij}^t}^* + \frac{\delta_{ij}}{|T|}, \forall (ij) \in A$ are used. The other strategy starts out with different dual-optimal solutions for each terminal $t$ and lets these solutions converge towards $\bar{\pi}_{ij}^t, \forall t \in T, \forall (ij) \in A$ using a parameter $Q \geq 2$. This parameter denotes the major iterations and is iterated with $q = 1, \ldots, Q$. The dual values for the terminal $t' \in T$ currently considered, are denoted as $\hat{\pi}_{ij}^t, \forall (i,j) \in A, \forall t \in T$ and are set as follows:

$$\hat{\pi}_{ij}^t = \begin{cases} {\pi_{ij}^t}^* + \frac{\delta_{ij}}{|T|} + \frac{Q-q}{Q-1}(\delta_{ij} - \frac{\delta_{ij}}{|T|}) & \text{if } t = t' \\ {\pi_{ij}^t}^* & \text{otherwise} \end{cases} \tag{5.39}$$

Equation 5.39 divides the interval $\left[\frac{\delta_{ij}}{|T|}, \delta_{ij}\right]$ into $Q-1$ equally sized intervals according to the actual major iteration $q$. A major iteration happens, if we do not find a path with positive reduced costs using the current dual variables $\hat{\pi}_{ij}^t$. For each terminal $t \in T$, the resulting vector $\hat{\pi}$ is a dual-optimal solution and when $Q = q$, we get $\hat{\pi}_{ij}^{t'} = \bar{\pi}_{ij}^{t'}$ for the currently considered terminal $t'$. Hence, we can stop the column generation at the current node, iff $q = Q$ and no path with positive reduced costs has been found by the pricing subproblem.

In addition to these two strategies, we propose two new variations: One consists of distributing the slack only over all dual variables with value zero. When variables ${\pi_{ij}^t}^*$ have the value zero, it is likely that the corresponding arc $(i,j)$ will not be in the solution (especially in later iterations), because otherwise it would be a good chance that $\pi_{ij}^t$ is in one or several cuts and hence there is a high probability that it has a positive dual value. The other variant simply randomly distributes the slack.

Note, that the distribution of the slack $\delta_{ij}$ of (5.4) in general is a very powerful tool (with the natural limitation, that not every Constraint (5.4) has slack in a given dual-optimal solution): We can guide the behavior of the pricing subproblem. For example, if we want to have an arc $(i,j) \in A$ in the solution, we can distribute the slack of Constraints (5.4) for all arcs $(i',j') \neq (i,j)$, giving all the other arcs a higher value and thus increasing the chance that $(i,j)$ is in the found hop-constrained cheapest path of the pricing subproblem. This could be used in combination with the fact we established in the previous section, that arcs with small costs are more likely to occur in a solution. If we really want to "punish" some arc, we can even use the idea of the second strategy above, i.e. using a terminal-specific dual-optimal solution for every terminal $t \in T$, then we can distribute the whole slack only over this $(i,j) \in A$ and leave all the other arcs untouched. Moreover, by adding all the slack to one terminal $t \in T$ (as is done for every terminal in the second strategy of alternative-dual optimal solutions method), we can make it very hard to find new paths to this terminal $t$, since the path has to be smaller than $\mu_t$.

### Piecewise Linear Stabilization

Piecewise linear stabilization techniques try to minimize the oscillation of the values of the dual variables during column generation, i.e. the bang-bang effect. This is achieved by adding a piecewise linear penalty function in the dual program. The function penalizes the derivation of the values of the dual variables from a current stability center (also called trust region).

Du Merle et al. [26] proposed to use a 3-piecewise linear stabilization function, while Amor et al. [5] recommend a 5-piecewise linear stabilization function. Moreover, Amor et al. [7] compared different linear and nonlinear stabilization functions and concluded, that a 5-piecewise linear stabilization function has the most practical value. Thus we used 5-piecewise linear stabilization.

To illustrate the effect of the stabilization function (following [5]), consider the primal problem $Pr = min\{d^T x | Fx = e, x \geq 0\}$ and its corresponding dual $Du = max\{e^T \kappa | F^T \kappa \leq d\}$. Moreover, let $\hat{\kappa}^l$ denote the dual center at the current major iteration $l \in \mathbb{N}_0^+$ and $[\delta_1^l, \delta_2^l]$ ($\delta_1^l < \hat{\kappa}^l < \delta_2^l$) be the hyperbox around this dual center, where the values of the dual variables of the next iteration $l+1$ can lie, without being penalized. Furthermore, let $\varepsilon_1, \varepsilon_2$ be the penalties if a dual value in iteration $l+1$ is outside $[\delta_1^l, \delta_2^l]$ but inside the hyperbox $[\gamma_1^l, \gamma_2^l]$ ($\gamma_1^l \leq \delta_1^l < \delta_2^l \leq \gamma_2^l$) and $\zeta_1^l, \zeta_2^l$ be the penalties, if a dual value also lies outside $[\gamma_1^l, \gamma_2^l]$.

We get the following stabilized primal program $Pr'$:

$$\begin{aligned} \min \quad & d^T x - \gamma_1^{lT} s_1 - \delta_1^{lT} t_1 + \gamma_2^{lT} s_2 + \delta_2^{lT} t_2 \\ \text{s.t.} \quad & Fx - z_1 - y_1 + y_2 + z_2 = e \\ & s_1 \leq \zeta_1^l \\ & s_2 \leq \zeta_2^l \\ & r_1 \leq \varepsilon_1^l \\ & r_2 \leq \varepsilon_2^l \\ & x, s_1, s_2, r_1, r_2 \geq 0 \end{aligned} \tag{5.40}$$

and corresponding stabilized dual program $Du'$:

$$\begin{aligned} \max \quad & e^T \kappa - \zeta_1^{lT} v_1 - \varepsilon_1^{lT} u_1 - \zeta_2^{lT} v_2 - \varepsilon_2^{lT} u_2 \\ \text{s.t.} \quad & F^T \kappa \leq d \\ & \delta_1^l - u_1 \leq \kappa \leq \delta_2^l + u_2 \\ & \gamma_1^l - v_1 \leq \kappa \leq \gamma_2^l + v_2 \\ & u_1, u_2, v_1, v_2 \geq 0 \end{aligned} \tag{5.41}$$

Now every feasible solution to $Du'^l$ is also a feasible solution to $Du^l$, however the corresponding primal solution is infeasible if $-s_1 - r_1 + s_2 + r_2 \neq 0$. Thus we need to obtain a solution, where $s_1 = r_1 = s_2 = r_2 = 0$. This can be achieved in the following way:

Start with an initial stability center $\kappa^0$ and initial settings $\varepsilon_1^0, \varepsilon_2^0, \zeta_1^0, \zeta_2$ and $\gamma_1^0, \gamma_2^0, \delta_1^0, \delta_2^0$ for the penalties and trust regions, respectively. Then solve $Pr'^0$ with column generation. This is a major iteration of the algorithm, i.e. $l = 0$ at the start. If the solution to $Pr'^l$ does not fulfill

$s_1 = r_1 = s_2 = r_2 = 0$, update the stability center to the current dual solution, i.e. $\hat{\kappa}^{l+1} = \kappa^l$. The penalties and trust regions can also be updated. Then increment $l$ and repeat the process. It can be shown that this process converges to an optimal solution of the original problem $Pr$, if the trust region $[\delta_1^l, \delta_2^l]$ never shrinks to a single point [5].

### Method of Neame

In [71] a stabilization technique proposed by Neame [65] is mentioned. It consists of taking a convex combination of the actual dual optimal solution and the dual prices of the previous iteration for solving the subproblem. In our case, this amounts to $\bar{\pi}_{ij}^t = \alpha\pi_{ij}^t + (1-\alpha)\tilde{\pi}_{ij}^t$ and $\bar{\mu}_t = \alpha\mu_t + (1-\alpha)\tilde{\mu}_t$, where $\tilde{\pi}_{ij}^t$, $\tilde{\mu}_t$ describe the dual prices of the previous iteration and $0 \leq \alpha \leq 1$. Using such a convex combination of the values, it is possible to get solutions with positive reduced costs for the modified values, but with negative reduced costs in respect to the original values. In such a case, the generated columns are rejected, a new convex combination more in favor of the actual dual optimal solutions is computed (i.e. $\alpha$ is increased) and the pricing problem is solved again. This is repeated until we either get a solution with positive reduced costs both for the modified and the original values, or $\alpha = 1$, i.e. we used the original values and did not get positive reduced costs, which is just the "normal" stopping criterion for column generation.

The weighted Dantzig-Wolfe decomposition introduced by Wentges in [78] is very similar to the method of Neame. Instead of using a convex combination of the current and previous dual prices, a convex combination of the current dual solution and the dual prices of the solution providing the best Lagrangian dual bound found so far, is taken.

## 5.6 Pricing Strategies

To guarantee that the optimal solution to the MP is found, we only need to find a column with positive reduced costs or prove that none exists when solving the pricing subproblem [3, 61]. This means that we do not need to find the column with the most positive reduced costs. For example, a dual strategy consisting of a fast approximation algorithm and a slow exact algorithm for the subproblem could be used. To guarantee optimality, the exact algorithm needs only to be called, if the approximation algorithm does not find a column with positive reduced costs [3]. This strategy is not helpful in our case, since the hop-constrained cheapest path subproblem can be solved efficiently to optimality.

The above fact has also other implications, which turn out to be more useful: We are not restricted to add only one column with positive reduced costs during an iteration of column generation [3, 24, 61, 75] . This means, more than one path with positive reduced costs can be added for a terminal or paths for every terminal instead of stopping the pricing subproblem when the first path with positive reduced costs is found. Such pricing strategies will be the topic of the next subsections. It should be noted, that there is a trade-off, when adding more columns during an iteration of column generation: On the one hand, the RMP gets larger and thus needs more time for solving, on the other hand the overall number of iterations may decrease [3].

### Finding More Paths with Positive Reduced Costs

A cornerstone of the following pricing strategies will be the ability, that given a hop-constrained cheapest path to a terminal, we can find more paths with positive reduced costs efficiently. This can be done with the following algorithm, a slight modification of an algorithm presented in [41].

**Algorithm 9** Algorithm to add more than one column for a terminal

```
1:  procedure MORECOLUMNS(G = (V, A),S, f, p, t, μ_t, max_h)
2:      for h = 1, ..., H do
3:          if t ∈ S_h then
4:              for (j, t) ∈ A do
5:                  if (f(j, h − 1) + c_jt ≤ μ_t) then
6:                      if ((h ≠ max_h) ∨ ((j, t) ≠ p(v, max_h)) then
7:                          path p' ← path found by backtracking from f(j, h − 1)
8:                          path p ← p' ∪ (j, t)
9:                          add λ_p corresponding to p to the RMP
10:                     end if
11:                 end if
12:             end for
13:         end if
14:     end for
15: end procedure
```

Aside from the graph, MORECOLUMNS takes the data-structures $S_h$,$f$,$p$ and variable $max_h$ from Algorithm 2, hop limit $H$, the current terminal $t$ and the value corresponding dual variable $\mu_t$ as input. The algorithm considers every $h$ for which there is a path form the root node to the terminal $t$ consisting of exactly $h$ hops (i.e. $t \in S_h, h = 1, \ldots, H$). For these values of $h$ and every incoming arc $(j, t)$, it is checked, if there is a hop-constrained shortest path with $h$ hops using this arc. This amounts to the check, if the sum of cost of the path to $j$ using $h - 1$ arcs (i.e. $(f(j, h - 1))$ and the cost of the arc $(j, t)$ is not larger than $\mu_t$ (line). Line 6 ensures that the path found by HCCP is not added again.

MORECOLUMNS can be used to augment HCCP, i.e. if HCCP found a path with positive reduced costs for a given terminal, MORECOLUMS is called and tries to find more paths with positive reduced costs. The following pricing strategies can be used with or without MORECOLUMNS as augmentation to HCCP.

### Typical Pricing Strategies

Algorithm 10 is the basic pricing strategy. It simply adds a single column with positive reduced costs, if one exists. Since we need to resolve the RMP again after adding this single column, this strategy is likely to be not too efficient.

Algorithm 11 tries to find one or more paths with positive reduced costs for every terminal $t$. The only difference to Algorithm 10 is, that there is no $break$ after line 5.

**Algorithm 10** Basic pricing strategy

```
1: procedure BASICPRICING(T,H)
2:     for t ∈ T do
3:         path p ← HCCP(G = (V, A), t, H, B, c)
4:         if p ≠ ∅ then
5:             add λ_p corresponding to p to the RMP
6:             break
7:         end if
8:     end for
9: end procedure
```

**Algorithm 11** Another simple pricing strategy

```
1: procedure NORMALPRICING(T, H)
2:     for t ∈ T do
3:         path p ← HCCP(G = (V, A), t, H, B, c)
4:         if p ≠ ∅ then
5:             add λ_p corresponding to p to the RMP
6:         end if
7:     end for
8: end procedure
```

## A Pricing Strategy Based on a Tabu List

The next pricing strategy arises from the following idea: After some initial iterations of the pricing procedure, often only a few terminals still have paths with positive reduced costs, but when using NORMALPRICING, we call HCCP (and MORECOLUMNS) for every terminal in each iteration. Trying to avoid such unnecessary calls of HCCP should therefore result in a speed-up. Algorithm 12 gives a tabu search like procedure to achieve this. The strategy is therefore called TABUPRICING. It works by keeping an active-list $T_A \subseteq T$ including all terminal, for which a path with positive reduced costs has been found in at least one of the previous $k$ iterations of the pricing procedure. In each iteration, terminals $T \setminus T_A$ are only considered if no path with positive reduced costs for terminals on the active list has been found.

The pricing strategy uses the global variable $tabuList$, which is an integer vector, as tabu list. Moreover, it consists of two different procedures, INITTABUPRICING initializes the $tabuList$ to zero at the start of the branch-and-price procedure and TABUPRICING is the pricing procedure. In addition to the set of terminals $T$ and hop limit $H$, TABUPRICING also gets the parameter $k$ as input. This parameter indicates, how many pricing iterations ago a path with positive reduced costs to a terminal $t$ has had to be found, so that terminal $t$ is considered in the current iteration. These terminals form the set $T_A$. In the algorithm, HCCP gets called for every terminal $t \in T_A$. If find a path with positive reduced costs for a terminal $t$, aside from adding the path to the RMP, we also set $tabuList_t$ to zero. If we do not find a path, we increase $tabuList_t$. Thus a terminal gets removed from $T_A$ if no path with positive reduced costs to it has been found

**Algorithm 12** Pricing strategy based on a tabu list

```
1: procedure INITTABUPRICING(T)
2:     tabuList_t ← 0, ∀t ∈ T
3: end procedure
4: procedure TABUPRICING(T,H,k)
5:     found ← false
6:     notConsidered ← ∅
7:     for t ∈ T do
8:         if tabuList_t < k then
9:             path p ← HCCP(G = (V, A), t, H, B, c)
10:            if p ≠ ∅ then
11:                found ← true
12:                tabuList_t ← 0
13:                add λ_p corresponding to p to the RMP
14:            else
15:                tabuList_t ← tabuList_t + 1
16:            end if
17:        else
18:            notConsidered ← notConsidered ∪ {i}
19:        end if
20:    end for
21:    if ¬found then
22:        for t ∈ notConsidered do
23:            path p ← HCCP(G = (V, A), t, H, B, c)
24:            if p ≠ ∅ then
25:                tabuList_t ← 0
26:                add λ_p corresponding to p to the RMP
27:            end if
28:        end for
29:    end if
30: end procedure
```

in the last $k$ pricing iterations.

Only if we do not find a path with positive reduced costs for any terminal in $T_A$, we call HCCP for the terminals $T \setminus T_A$ (which are saved in the set $notConsidered$). Naturally, if HCCP finds a path with positive reduced costs for a terminal $t \in T \setminus T_A$, we add it to the RMP, moreover, we set $tabuList_t$ to zero, i.e. terminal $t$ gets added to $T_A$ again.

### A Pricing Strategy Based on the Hop Constraints

Another pricing strategy was implemented with the runtime of HCCP in mind: Since the runtime of HCCP is $O(H|E|)$, it depends on the hop limit. We therefore start the pricing procedure with a small hop limit for every terminal and just use the original hop limit (and increase the hop

limit for the next pricing iteration), if we do not find a path with the smaller hop limit. Aside from having a shorter runtime of HCCP, we can possibly avoid the generation of unnecessary columns, since it is not realistic that every terminal has a path with a high hop limit in the best solution (at least if the hop limit $H$ is not too strict). The algorithm used for this pricing strategy is given in Algorithm 13 and is denoted by HOPPRICING.

**Algorithm 13** Pricing strategy based on the hop constraints

```
1:  procedure INITHOPPRICING(T)
2:      hvals_t ← 1, ∀t ∈ T
3:  end procedure
4:  procedure HOPPRICING(T, H)
5:      for t ∈ T do
6:          path p ← HCCP(G = (V, A), t, hvals_t, B, c)
7:          if p = ∅ ∧ hvals_t <= H then
8:              hvals_t ← hvals_t + 1
9:              path p ← HCCP(G = (V, A), t, H, B, c)
10:             if p ≠ ∅ then
11:                 add λ_p corresponding to p to the RMP
12:             end if
13:         else
14:             add λ_p corresponding to p to the RMP
15:         end if
16:     end for
17: end procedure
```

The procedure INITHOPPRICING is used to initialize the global variable $hvals$, which is a vector containing the currently used hop limit for every terminal $T$, to one at the start of the branch-and-price procedure. HOPPRICING carries out the pricing as described above, i.e. if HCCP does not find a path with positive reduced costs for a terminal $T$ and its corresponding current hop limit $hvals_t$, $hvals_t$ gets increased and HCCP is called again for this terminal $T$ using the original hop limit $H$ of the instance. This is done to guarantee an optimal solution.

### A Little Trick to Accelerate All Pricing Strategies

The previous branching decisions of the branch-and-bound tree can be used to accelerate all presented pricing strategies. Since we branch on terminal-variables $y_t$ and arc-variables $x_{ij}$, at any node of the branch-and-bound tree, some terminal-variables $y_t$ may be restricted to value zero. The corresponding terminals $t$ cannot be in the tree caused by the feasible solution at the current node and thus there can be no path with positive reduced costs to these terminals. Therefore we do not need to solve the pricing subproblem for these terminal.

CHAPTER 6

# Computational Results

Different combinations of the previously described formulations, stabilization methods, pricing strategies and heuristics to generate initial solutions have been implemented using ZIB SCIP 2.0.0 [1] with IBM CPLEX 12.2 as LP solver. The computations have been performed on a single core of an Intel Xeon E5540 processor with 2.53 GHz and 3GB RAM. A time-limit of 10000 CPU-seconds has been used for all experiments.

We used the test instances proposed by Costa et al. in [17], which are based on graphs B (denoted as MSteinb) and C (denoted as Steinc) from the OR-Library [4]. Since these instances originally have no revenues associated with the terminal nodes, Costa et al. adapted them for the STPRBH by assigning revenues generated by a discrete uniform distribution to the terminals. For the MSteinb graphs, the range of the distribution was $[1, 100]$ and for the Steinc graphs, they used the ranges $[1, 10]$ and $[1, 100]$ to generate two different problems out of every Steinc graph. Therefore, the MSteinb graphs will be referred to as B1 to B18 for the rest of the thesis and the Steinc graphs as C1-10 to C5-10 and C1-100 to C5-100, respectively. Sometimes, we will also use CX when referring to both of the Steinc graphs, i.e. CX-10 and CX-100 (for X between one and five)

We use the hop limits and budget proposed in [17], i.e. $H = 3, 6, 9, 12$ and $B = s, s/5, s/10$ for the MSteinb instances and $H = 5, 15, 25$ and $B = s, s/10, s/30$ for the Steinc instances. Here $s$ denotes the sum of all edge-costs, i.e. $\sum_{e \in E} c_e$. Moreover, the terminal with the smallest index is chosen as root node [17]. In the following tables the budget will be marked with $b$, which stands for the divisor, i.e. $B = s/1$ is denoted with $b = 1$, $B = s/10$ with $b = 10$ and so on. In the text, we will use GRAPH/H=X/b=Y to fully specify an instance (H or b are dropped, when it is clear from context).

Since SCIP allows the configuration of some parameters, e.g. the algorithm used for solving LPs, we did some initial experiments to find the best configuration. The dual-simplex algorithm clearly outperforms other possible choices like primal-simplex or barrier. Using steepest edge pricing as LP pricing strategy as mentioned in [61] nearly halves the runtime for some instances compared with the default setting. The other recommendation from [61], devex pricing, does not work so well. Hence, all results in the following have been obtained using the dual-simplex

algorithm and steepest edge pricing.

We use MORECOLUMNS (see Section 5.6) to add more than one column with positive reduced costs for a terminal during a pricing iteration (if such columns exist) in all our experiments, since initial test showed it clearly outperformed adding single columns for each terminal in preliminary tests.

## 6.1 Preprocessing

Tables 6.1 and 6.2 give the resulting reduced graphs when using the preprocessing steps described in Section 5.3 on the instances based on the MSteinb and Steinc graphs. The runtime of preprocessing was below one second for all instances.

Interestingly, a hop limit larger than 3 for MSteinb and 5 for Steinc does not significantly change the resulting reduced instance. Moreover, different budget also do hardly influence the outcome. The latter is, however, no surprise, because the budget does only have effects, if the cheapest path to some node is more expensive than the budget. Note that the pairs of graphs CX-10 and CX-100 give the same preprocessing results, since they only differ in revenue. All further computational experiments have been performed on these preprocessed instances.

Table 6.1: Result of the preprocessing on the instances based on the MSteinb graphs. The number of nodes, edges and terminals of the original graph is given in columns $|V_{orig}|$, $|E_{orig}|$ and $|T_{orig}|$, respectively.

| Instance | | | | | $H=3$ | | | $H=6$ | | | $H=9$ | | | $H=12$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Graph | $\|V_{orig}\|$ | $\|E_{orig}\|$ | $\|T_{orig}\|$ | b | $\|V\|$ | $\|E\|$ | $\|T\|$ | $\|V\|$ | $\|E\|$ | $\|T\|$ | $\|V\|$ | $\|E\|$ | $\|T\|$ | $\|V\|$ | $\|E\|$ | $\|T\|$ |
| B1 | 50 | 63 | 9 | 1 | 15 | 16 | 2 | 37 | 50 | 8 | 38 | 51 | 9 | 38 | 51 | 9 |
| | | | | 5 | 15 | 16 | 2 | 37 | 50 | 8 | 38 | 51 | 9 | 38 | 51 | 9 |
| | | | | 10 | 15 | 16 | 2 | 37 | 50 | 8 | 38 | 51 | 9 | 38 | 51 | 9 |
| B2 | 50 | 63 | 13 | 1 | 11 | 11 | 3 | 39 | 50 | 12 | 41 | 54 | 13 | 41 | 54 | 13 |
| | | | | 5 | 11 | 11 | 3 | 39 | 50 | 12 | 41 | 54 | 13 | 41 | 54 | 13 |
| | | | | 10 | 11 | 11 | 3 | 35 | 43 | 11 | 35 | 43 | 11 | 35 | 43 | 11 |
| B3 | 50 | 63 | 25 | 1 | 5 | 4 | 5 | 35 | 42 | 21 | 44 | 57 | 25 | 44 | 57 | 25 |
| | | | | 5 | 5 | 4 | 5 | 35 | 42 | 21 | 44 | 57 | 25 | 44 | 57 | 25 |
| | | | | 10 | 5 | 4 | 5 | 35 | 42 | 21 | 41 | 53 | 22 | 41 | 53 | 22 |
| B4 | 50 | 100 | 9 | 1 | 35 | 55 | 5 | 46 | 96 | 9 | 46 | 96 | 9 | 46 | 96 | 9 |
| | | | | 5 | 35 | 55 | 5 | 46 | 96 | 9 | 46 | 96 | 9 | 46 | 96 | 9 |
| | | | | 10 | 35 | 55 | 5 | 46 | 96 | 9 | 46 | 96 | 9 | 46 | 96 | 9 |
| B5 | 50 | 100 | 13 | 1 | 42 | 63 | 12 | 45 | 95 | 13 | 45 | 95 | 13 | 45 | 95 | 13 |
| | | | | 5 | 42 | 63 | 12 | 45 | 95 | 13 | 45 | 95 | 13 | 45 | 95 | 13 |
| | | | | 10 | 42 | 63 | 12 | 45 | 95 | 13 | 45 | 95 | 13 | 45 | 95 | 13 |
| B6 | 50 | 100 | 25 | 1 | 40 | 65 | 18 | 49 | 99 | 25 | 49 | 99 | 25 | 49 | 99 | 25 |
| | | | | 5 | 40 | 65 | 18 | 49 | 99 | 25 | 49 | 99 | 25 | 49 | 99 | 25 |
| | | | | 10 | 40 | 65 | 18 | 49 | 99 | 25 | 49 | 99 | 25 | 49 | 99 | 25 |
| B7 | 75 | 94 | 13 | 1 | 7 | 6 | 2 | 45 | 52 | 11 | 57 | 76 | 13 | 57 | 76 | 13 |
| | | | | 5 | 7 | 6 | 2 | 45 | 52 | 11 | 57 | 76 | 13 | 57 | 76 | 13 |
| | | | | 10 | 7 | 6 | 2 | 45 | 52 | 11 | 57 | 76 | 13 | 57 | 76 | 13 |
| B8 | 75 | 94 | 19 | 1 | 7 | 6 | 2 | 39 | 43 | 13 | 54 | 73 | 19 | 54 | 73 | 19 |
| | | | | 5 | 7 | 6 | 2 | 39 | 43 | 13 | 54 | 73 | 19 | 54 | 73 | 19 |
| | | | | 10 | 7 | 6 | 2 | 39 | 43 | 13 | 54 | 73 | 19 | 54 | 73 | 19 |
| B9 | 75 | 94 | 38 | 1 | 25 | 24 | 13 | 63 | 82 | 37 | 65 | 84 | 38 | 65 | 84 | 38 |
| | | | | 5 | 25 | 24 | 13 | 63 | 82 | 37 | 65 | 84 | 38 | 65 | 84 | 38 |
| | | | | 10 | 25 | 24 | 13 | 63 | 82 | 37 | 65 | 84 | 38 | 65 | 84 | 38 |
| B10 | 75 | 150 | 13 | 1 | 47 | 66 | 7 | 72 | 147 | 13 | 72 | 147 | 13 | 72 | 147 | 13 |
| | | | | 5 | 47 | 66 | 7 | 72 | 147 | 13 | 72 | 147 | 13 | 72 | 147 | 13 |
| | | | | 10 | 47 | 66 | 7 | 72 | 147 | 13 | 72 | 147 | 13 | 72 | 147 | 13 |
| B11 | 75 | 150 | 19 | 1 | 40 | 45 | 8 | 72 | 147 | 19 | 72 | 147 | 19 | 72 | 147 | 19 |
| | | | | 5 | 40 | 45 | 8 | 72 | 147 | 19 | 72 | 147 | 19 | 72 | 147 | 19 |
| | | | | 10 | 40 | 45 | 8 | 72 | 147 | 19 | 72 | 147 | 19 | 72 | 147 | 19 |
| B12 | 75 | 150 | 38 | 1 | 46 | 57 | 25 | 75 | 150 | 38 | 75 | 150 | 38 | 75 | 150 | 38 |
| | | | | 5 | 46 | 57 | 25 | 75 | 150 | 38 | 75 | 150 | 38 | 75 | 150 | 38 |
| | | | | 10 | 46 | 57 | 25 | 75 | 150 | 38 | 75 | 150 | 38 | 75 | 150 | 38 |
| B13 | 100 | 125 | 17 | 1 | 19 | 18 | 3 | 73 | 97 | 17 | 74 | 99 | 17 | 74 | 99 | 17 |
| | | | | 5 | 19 | 18 | 3 | 73 | 97 | 17 | 74 | 99 | 17 | 74 | 99 | 17 |
| | | | | 10 | 19 | 18 | 3 | 73 | 97 | 17 | 74 | 99 | 17 | 74 | 99 | 17 |
| B14 | 100 | 125 | 25 | 1 | 10 | 9 | 4 | 70 | 84 | 23 | 79 | 104 | 25 | 79 | 104 | 25 |
| | | | | 5 | 10 | 9 | 4 | 70 | 84 | 23 | 79 | 104 | 25 | 79 | 104 | 25 |
| | | | | 10 | 10 | 9 | 4 | 70 | 84 | 23 | 79 | 104 | 25 | 79 | 104 | 25 |
| B15 | 100 | 125 | 50 | 1 | 31 | 32 | 19 | 83 | 105 | 50 | 83 | 108 | 50 | 83 | 108 | 50 |
| | | | | 5 | 31 | 32 | 19 | 83 | 105 | 50 | 83 | 108 | 50 | 83 | 108 | 50 |
| | | | | 10 | 31 | 32 | 19 | 83 | 105 | 50 | 83 | 108 | 50 | 83 | 108 | 50 |
| B16 | 100 | 200 | 17 | 1 | 54 | 70 | 10 | 95 | 195 | 17 | 95 | 195 | 17 | 95 | 195 | 17 |
| | | | | 5 | 54 | 70 | 10 | 95 | 195 | 17 | 95 | 195 | 17 | 95 | 195 | 17 |
| | | | | 10 | 54 | 70 | 10 | 95 | 195 | 17 | 95 | 195 | 17 | 95 | 195 | 17 |
| B17 | 100 | 200 | 25 | 1 | 16 | 15 | 6 | 95 | 194 | 25 | 95 | 195 | 25 | 95 | 195 | 25 |
| | | | | 5 | 16 | 15 | 6 | 95 | 194 | 25 | 95 | 195 | 25 | 95 | 195 | 25 |
| | | | | 10 | 16 | 15 | 6 | 95 | 194 | 25 | 95 | 195 | 25 | 95 | 195 | 25 |
| B18 | 100 | 200 | 50 | 1 | 58 | 72 | 26 | 96 | 196 | 50 | 96 | 196 | 50 | 96 | 196 | 50 |
| | | | | 5 | 58 | 72 | 26 | 96 | 196 | 50 | 96 | 196 | 50 | 96 | 196 | 50 |
| | | | | 10 | 58 | 72 | 26 | 96 | 196 | 50 | 96 | 196 | 50 | 96 | 196 | 50 |

Table 6.2: Result of the preprocessing on the instances based on the Steinc graphs. The number of nodes, edges and terminals of the original graph is given in columns $|V_{orig}|$, $|E_{orig}|$ and $|T_{orig}|$, respectively.

| Instance | | | | | $H = 5$ | | | $H = 15$ | | | $H = 25$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Graph | $\|V_{orig}\|$ | $\|E_{orig}\|$ | $\|T_{orig}\|$ | b | $\|V\|$ | $\|E\|$ | $\|T\|$ | $\|V\|$ | $\|E\|$ | $\|T\|$ | $\|V\|$ | $\|E\|$ | $\|T\|$ |
| C1 | 500 | 625 | 5 | 1 | 79 | 82 | 2 | 357 | 482 | 5 | 357 | 482 | 5 |
| | | | | 10 | 79 | 82 | 2 | 357 | 482 | 5 | 357 | 482 | 5 |
| | | | | 30 | 79 | 82 | 2 | 357 | 482 | 5 | 357 | 482 | 5 |
| C2 | 500 | 625 | 10 | 1 | 94 | 99 | 5 | 353 | 478 | 10 | 353 | 478 | 10 |
| | | | | 10 | 94 | 99 | 5 | 353 | 478 | 10 | 353 | 478 | 10 |
| | | | | 30 | 94 | 99 | 5 | 353 | 478 | 10 | 353 | 478 | 10 |
| C3 | 500 | 625 | 83 | 1 | 152 | 163 | 30 | 366 | 491 | 83 | 366 | 491 | 83 |
| | | | | 10 | 152 | 163 | 30 | 366 | 491 | 83 | 366 | 491 | 83 |
| | | | | 30 | 152 | 163 | 30 | 366 | 491 | 83 | 366 | 491 | 83 |
| C4 | 500 | 625 | 125 | 1 | 75 | 76 | 22 | 391 | 516 | 125 | 391 | 516 | 125 |
| | | | | 10 | 75 | 76 | 22 | 391 | 516 | 125 | 391 | 516 | 125 |
| | | | | 30 | 75 | 76 | 22 | 391 | 516 | 125 | 391 | 516 | 125 |
| C5 | 500 | 625 | 250 | 1 | 87 | 94 | 49 | 421 | 546 | 250 | 421 | 546 | 250 |
| | | | | 10 | 87 | 94 | 49 | 421 | 546 | 250 | 421 | 546 | 250 |
| | | | | 30 | 87 | 94 | 49 | 421 | 546 | 250 | 421 | 546 | 250 |

## 6.2 Comparison of the Branch-and-Price Approaches

Table 6.3 gives the comparison of the branch-and-price approaches based on the different formulations discussed in Chapter 4 on the Steinc graphs with H=15 and $b = 1, 5, 10$. We chose NORMALPRICING as pricing strategy, i.e. we try to add columns with positive reduced costs for every terminal during a pricing iteration. Moreover, we used no stabilization or heuristic.

Formulation $(\mathrm{DPF})$ has the fastest runtime for 20 of the 30 instances, followed by $(\mathrm{DPFnt})$ with 16 out of 30 and $(\mathrm{DPF})$ with (4.37) with 15 out of 30 (remember that constraints (4.37) ensure that at most one arc of each oppositely directed pair of arcs is used). The instances based on graphs C1 and C2 are solved almost instantly by all our approaches, even the one based on $(\mathrm{UPF})$. This is consistent with the results of Costa et al. [17], where these instances also turned out to be easy to solve.

Moreover, all instances with $b = 1$ are solved within few seconds regardless of the used formulation. Taking into account the structure of the STPRBH, when $b = 1$, this is no surprise: Since with $b = 1$, the budget is equal to the sum of all the edges in the instance, it does not matter and the problem reduces to the STPRH, which is polynomially solvable [69]. Any tree, which connects all terminals reachable with respect to the hop limit of the instance, is an optimal solution. Thus, we only need to check for every terminal, if there is a hop-constrained path to it, take the terminals, where such a paths exists and the corresponding paths, and ensure that the graph formed by them is a tree. Our branch-and-price approach behaves similar to this strategy. The pricing subproblem provides different hop-constrained paths for every terminal and the ILP ensures that the paths picked in the solution form a tree (when $b \neq 1$, the ILP also ensures, that the paths do not exceed the budget). Therefore, when $b = 1$ all proposed branch-and-price approaches should yield good performance even for quite large graphs and hop limits. This is in contrast to the branch-and-cut algorithms presented in [17], which do not scale so well when

$b = 1$.

The most difficult instances for our approaches are the ones based on graphs C3 and C5 with $b = 10$. Using the undirected formulation $(\mathrm{UPF})$, all four of these instances (i.e. C3-10/b=10, C3-100/b=10, C5-10/b=10 and C5-100/b=10) could not be solved within the time limit. Formulations $(\mathrm{DPF})$ with (4.37) and $(\mathrm{DPF2})$ are not able to solve C3-100/b=10 within the time limit and the runtime using $(\mathrm{DPFT})$ is also relatively high for C3-100/b=10 compared to other instances.

Aside from the performance on these difficult instances, there is no formulation, which dominates the rest. Even on the same underlying graphs, different budgets can result in vastly different performance of the formulations. For example, $(\mathrm{DPFT})$ is nearly three times as fast as $(\mathrm{DPFnt})$ for C4-10/b=10, while for C4-10/b=30 the picture is reversed. Concerning average runtime, $(\mathrm{DPFnt})$ is the best, followed by $(\mathrm{DPF})$. The undirected formulation $(\mathrm{UPF})$ has the worst performance by far with an average runtime more than five times as high as $(\mathrm{DPFnt})$. Formulation $(\mathrm{DPF2})$ also performed relatively poor in comparison to the other directed formulations. This may be explained from the fact, that there are no terminal-variables to branch on.

For all further experiments, formulation $(\mathrm{DPFnt})$ is used. Moreover, instances based on graphs C1, C2 and $b = 1$ are dropped from our testsetting, since they are too easy to solve. Instead, we include instances with hop limit H=25.

Table 6.3: CPU-time in seconds for branch-and-price based on the different formulations given in Chapter 4 using NORMALPRICING and no stabilization and no heuristic for instances based on Steinc graphs and $H = 15$

| Instance | | Setting | | | | | |
|---|---|---|---|---|---|---|---|
| Graph | b | (UPF) | (DPFnt) | (DPF) | (4.37) | (DPFT) | (DPF2) |
| C1-10 | 1 | **0** | **0** | **0** | **0** | **0** | **0** |
| | 10 | **0** | **0** | **0** | **0** | **0** | **0** |
| | 30 | **0** | **0** | **0** | **0** | **0** | **0** |
| C2-10 | 1 | **0** | **0** | **0** | **0** | **0** | **0** |
| | 10 | **0** | **0** | **0** | **0** | **0** | **0** |
| | 30 | 65 | **7** | 15 | 13 | 11 | 32 |
| C3-10 | 1 | 1 | **0** | **0** | **0** | 1 | 1 |
| | 10 | 10000 | 226 | 495 | 719 | 416 | **201** |
| | 30 | 83 | **50** | 213 | 107 | 74 | 190 |
| C4-10 | 1 | 3 | 1 | **0** | **0** | 10 | 3 |
| | 10 | 999 | 222 | 338 | 443 | **66** | 109 |
| | 30 | 286 | **256** | 377 | 581 | 535 | 637 |
| C5-10 | 1 | 14 | 11 | **1** | **1** | 24 | 8 |
| | 10 | 10000 | 247 | 176 | 192 | 193 | **120** |
| | 30 | 821 | 175 | **125** | 190 | 165 | 396 |
| C1-100 | 1 | **0** | **0** | **0** | **0** | **0** | **0** |
| | 10 | **0** | **0** | **0** | **0** | **0** | **0** |
| | 30 | **0** | **0** | **0** | **0** | **0** | **0** |
| C2-100 | 1 | **0** | **0** | **0** | **0** | **0** | **0** |
| | 10 | **0** | **0** | **0** | **0** | **0** | **0** |
| | 30 | 95 | **13** | **13** | 14 | 162 | 71 |
| C3-100 | 1 | 1 | 2 | **0** | **0** | 1 | 1 |
| | 10 | 10000 | **1721** | 1986 | 10000 | 5584 | 10000 |
| | 30 | 14 | 18 | **11** | 15 | 17 | 12 |
| C4-100 | 1 | 3 | 1 | **0** | 1 | 2 | 3 |
| | 10 | 3705 | 2080 | 1555 | 1627 | **1167** | 2938 |
| | 30 | **416** | 437 | 753 | 709 | 546 | 813 |
| C5-100 | 1 | 13 | 6 | **1** | **1** | 31 | 7 |
| | 10 | 10000 | 678 | **396** | 622 | 512 | 425 |
| | 30 | 1191 | 1603 | 3397 | 1609 | 2423 | **756** |
| #-best | | 11 | 16 | 20 | 15 | 12 | 13 |
| average | | 1590 | 258 | 328 | 561 | 398 | 557 |

## 6.3 Influence of the Heuristics

Table 6.4 detail our computational results on experiments using different heuristics on the Steinc graphs C3, C4, C5 with H=15 and $b = 5, 10$. The results in Table 6.4 were obtained using formulation (DPFnt), no stabilization and NORMALPRICING.

We have tested the following heuristics described in Section 5.4: SHORTEST, which just adds a hop-constrained cheapest path for every terminal, GREEDY, which adds a hop-constrained paths for every terminal taking into account already added paths, HOPSHORTEST, which is SHORTEST with gradually increasing hop limit and KNAPSACK, which is inspired by a simple heuristic for the knapsack problem. Moreover, variations of SHORTEST and GREEDY, where more than one path for every terminal is added with the help of MORECOLUMNS, have also been tested.

We conclude that SHORTEST, which yields the fastest runtime in four out of twelve instances seems to perform best for these instances. Furthermore, using no heuristic as well was using SHORTEST with MORECOLUMNS performs quite well. HOPSHORTEST has the second best average runtime, but has the fastest runtime only for one instance. The other heuristics have poor performance, with KNAPSACK being the worst by far.

On an individual instance level, some interesting things can be observed: While SHORTEST is the best for C5-100 both when $b = 5$ and $b = 10$, it gives pretty bad results on the same graph, when the revenues only range from one to ten (i.e. C5-10). Interestingly, for C5-10 SHORTEST with MORECOLUMNS is best both for $b = 5$ and $b = 10$, needing 1/3 of the time of only SHORTEST for C5-10/b=20.

For C4-10/b=10, SHORTEST and HOPSHORTEST are three to six times faster than the rest and for C5-100/b=30, SHORTEST, SHORTEST with MORECOLUMNS and HOPSHORTEST are two times as fast as the other tested heuristics and no heuristic.

Table 6.4: CPU-times in seconds with different heuristics using formulation $(\mathrm{DPFnt})$ for instances based on Steinc graphs and $H = 15$. NO denotes no heuristic, SH heuristic SHORTEST, GR heuristic GREEDY, SHM heuristic SHORTEST with MORECOLUMNS, GRM heuristic GREEDY with MORECOLUMS, HS heuristic HOPSHORTEST and KS heuristic KNAPSACK.

| Instance | | Setting | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Graph | b | NO | SH | GR | SHM | GRM | HS | KS |
| C3-10 | 10 | **226** | 266 | 282 | 264 | 324 | 252 | 367 |
| | 30 | 50 | 52 | 69 | 137 | **36** | 75 | 60 |
| C4-10 | 10 | 222 | **76** | 203 | 380 | 451 | 85 | 277 |
| | 30 | 256 | **253** | 357 | **253** | 405 | 259 | 469 |
| C5-10 | 10 | 247 | 291 | 243 | **184** | 187 | 247 | 190 |
| | 30 | 175 | 391 | 172 | **133** | 518 | 513 | 186 |
| C3-100 | 10 | **1721** | 2129 | 2789 | 2474 | 2504 | 2537 | 1982 |
| | 30 | 18 | 15 | 14 | 15 | 15 | **13** | 20 |
| C4-100 | 10 | 2080 | 2026 | 1858 | 1954 | **1482** | 1694 | 2751 |
| | 30 | **437** | 559 | **437** | 680 | 598 | 558 | 598 |
| C5-100 | 10 | 678 | **497** | 580 | 637 | 560 | 530 | 543 |
| | 30 | 1603 | **766** | 1587 | 867 | 1379 | 867 | 1795 |
| #-best | | 3 | 4 | 1 | 3 | 2 | 1 | 0 |
| average | | 642 | 610 | 715 | 664 | 704 | 635 | 769 |

Table 6.5 presents the results obtained when using a looser hop limit of H=25. Now we get a different picture, where using no heuristic gives best average runtime and together with KNAPSACK has the fastest runtime most often. The two best heuristics when H=15, SHORTEST and HOPSHORTEST, now have the highest average runtime. Two instances (C3-10/b=10 and C5-100/b=30) are not solvable within the time limit no matter what heuristic is used.

This striking dependence of the performance on H is also visible on the individual instance level. For C5-10/H=15/b=10 SHORTEST with MORECOLUMS is best, closely followed by GREEDY with MORECOLUMS and KNAPSACK, while the other heuristics results in considerably worse runtime. For H=25 (i.e. instance C5-10/H=25/b=10), GREEDY with MORECOLUMNS gives an acceptable performance (third best), while SHORTEST with

MORECOLUMNS and KNAPSACK result in the worst runtimes by far.

Table 6.5: CPU-times in seconds using different heuristics and formulation $(\mathrm{DPFnt})$, NORMALPRICING and no stabilization for instances based on Steinc graphs and $H = 25$. NO denotes no heuristic, SH heuristic SHORTEST, GR heuristic GREEDY, SHM heuristic SHORTEST with MORECOLUMNS, GRM heuristic GREEDY with MORECOLUMS, HS heuristic HOPSHORTEST and KS heuristic KNAPSACK.

| Instance | | Setting | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Graph | b | NO | SH | GR | SHM | GRM | HS | KS |
| C3-10 | 10 | 2126 | 2605 | 2799 | 2680 | 2451 | 2782 | **752** |
| | 30 | 297 | 862 | 555 | **175** | 317 | 953 | 699 |
| C4-10 | 10 | 267 | 177 | 172 | 141 | 213 | 200 | **122** |
| | 30 | 704 | 470 | 492 | 530 | 534 | 431 | **414** |
| C5-10 | 10 | 2862 | 3974 | **2814** | 4209 | 3019 | 4060 | 4172 |
| | 30 | **1342** | 1967 | 1634 | 1639 | 1777 | 1968 | 1733 |
| C3-100 | 10 | **10000** | **10000** | **10000** | **10000** | **10000** | **10000** | **10000** |
| | 30 | **28** | 52 | 39 | 60 | 47 | 50 | 41 |
| C4-100 | 10 | 148 | 188 | 187 | 140 | **122** | 184 | 154 |
| | 30 | **632** | **632** | 734 | 675 | 661 | 650 | 714 |
| C5-100 | 10 | 3775 | 3890 | 3992 | **3761** | 4595 | 3982 | 5122 |
| | 30 | **10000** | **10000** | **10000** | **10000** | **10000** | **10000** | **10000** |
| #-best | | 5 | 3 | 3 | 4 | 3 | 2 | 5 |
| average | | 2681 | 2901 | 2784 | 2834 | 2811 | 2938 | 2826 |

For all further experiments, heuristic $SHORTEST$ is used, since the combination of good performance for instances with $H = 15$ and rather bad performance for instances with $H = 15$ provides an interesting basis.

## 6.4 Comparison of the Stabilization Techniques

In this section, we give a comparison of the performance of the different variants of stabilization methods described in Section 5.5.

We tested three different strategies based on alternative-dual optimal solutions. In the first strategy, the slack of Constraints (5.4) is distributed equally over all dual variables occurring in the respective constraints. The second strategy uses parameter Q to construct an unique solution for every terminal. Q has been set to ten in our experiments. The third strategy distributes the slack of Constraints (5.4) only over the dual variables with value zero.

Moreover, the method of Neame, which tries to smoothen the progress of the dual variables by taking a convex combination of the dual variables of the current and previous pricing iteration, was tested. At each iteration, parameter $\alpha$ for the weight of the current dual solution was set to $0.5$ and increased in steps of $0.1$ until a path with positive reduced costs is found or $\alpha = 1$, which means the current dual solution for pricing is used. A combination of the method of Neame with the first strategy of alternative dual-optimal solutions has also been tested. In this combination, the slack of Constraints (5.4) is distributed equally over the current dual values and then a convex combination with the dual values from the previous iteration is taken.

Furthermore, piecewise linear stabilization was tested. We used stabilization only in the column generation of the root node in the branch-and-price tree. Both the $\mu_t$ and $\pi_{ij}^t$ variables were stabilized, the initial stability center was set to the respective dual values of the first LP iteration. In the following, let $\mu_t^l$ and $\pi_{ij}^{t\ l}$ denote the dual solutions at major iteration $l$.

The inner hyperbox at major iteration $l$ was set to variables was $[\mu_t^{l-1} - 0.3r_t, \mu_t^{l-1} + 0.3r_t]$ for the $\mu_t$, the outer hyberbox was set to $[\mu_t^{l-1} - 0.9r_t, \mu_t^{l-1} + 0.9r_t]$ . For the $\pi_{ij}^t$ the hyperboxes were $[\pi_{ij}^{t\ l-1} - 0.3c_{ij}, \pi_{ij}^{t\ l-1} + 0.3c_{ij}]$ and $[\pi_{ij}^{t\ l-1} - c_{ij}, \pi_{ij}^{t\ l-1} + c_{ij}]$, respectively. The penalty parameters for all variables were set to $\epsilon_1 = \epsilon_2 = 0.3$ and $\zeta_1 = \zeta_2 = 1.0$. Both the penalties and the size of the hyperboxes were not changed during the run of the algorithm. We also did some testruns with other settings for both the hyperboxes and the penalties, but the runtimes did not change significantly.

Table 6.6 summarizes the results for the different stabilization techniques for formulation $(\mathrm{DPFnt})$ on the Steinc graphs C3, C4, C5 with H=15 and $b = 5, 10$ using NORMALPRICING and heuristic SHORTEST.

Both strategies based on the method of Neame as well as piecewise linear stabilization have a disastrous effect on the runtime. The average runtime using the method of Neame is more than four times higher as no stabilization and the combination with alternative dual-optimal solutions gives a roughly three times as high average runtime as using no stabilization. This is consistent with the results in [71], where the method of Neame also performed poorly. Piecewise linear stabilization yields even worse results, most of the instances have not been solved within the time limit. A closer look at the run of the algorithms revealed, that many updates of the stability center were necessary and thus many LPs needed to be solved. The same behavior has also been observed for the rooted delay constrained Steiner tree problem in [57, 58].

On the other hand, all three strategies based on alternative dual-optimal solutions give a significantly lower average runtime than using no stabilization. The first and third strategy result in a slightly lower average runtime than the second, moreover they results in the shortest runtime for instances most often (six times for the first strategy and four times for the third strategy against one time for no stabilization or the third strategy).

Taking a closer look at individual instances, we can again observe that the revenues associated with the terminals play a role in the performance of the stabilization. For the instances based on graph C5, with the revenues between one and ten (i.e. C5-10), the first strategy is clearly the best and even five times faster than the third, when $b = 30$. But for C5-100 (i.e. revenues between one and hundred), the third strategy is the best

Table 6.6: CPU-times in seconds with different stabilization techniques using formulation $(\mathrm{DPFnt})$, NORMALPRICING and heuristic SHORTEST for instances based on Steinc graphs and $H = 15$. AD 1 denotes the first strategy based on alternative-dual optimal solutions (slack over all dual variables), AD2 the second (with parameter Q), and AD3 the third (slack only over dual variables with value zero). N1 denotes the method of Neame, N2 the combination of the method of Neame and the first strategy based on alternative-dual optimal solutions, PL piecewise linear stabilization.

| Instance | | Setting | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Graph | $b$ | NO | AD1 | AD2 | AD3 | N1 | N2 | PL |
| C3-10 | 10 | 266 | 163 | 211 | **149** | 1337 | 841 | 694 |
| | 30 | **52** | 109 | 57 | 55 | 1024 | 295 | 255 |
| C4-10 | 10 | **76** | 296 | 228 | 259 | 350 | 261 | 1789 |
| | 30 | 253 | **243** | 438 | 272 | 1851 | 2095 | 1418 |
| C5-10 | 10 | 291 | **154** | 207 | 166 | 1103 | 943 | 10000 |
| | 30 | 391 | **84** | 198 | 371 | 882 | 967 | 10000 |
| C3-100 | 10 | 2129 | 1640 | 1734 | **1353** | 8960 | 4334 | 10000 |
| | 30 | 15 | **13** | 24 | 15 | 146 | 61 | 945 |
| C4-100 | 10 | 2026 | 1719 | **1483** | 1576 | 7059 | 7000 | 10000 |
| | 30 | 559 | **447** | 520 | 691 | 2921 | 2764 | 10000 |
| C5-100 | 10 | 497 | 435 | 402 | **340** | 2905 | 2024 | 10000 |
| | 30 | 766 | 638 | 677 | **605** | 4534 | 3108 | 10000 |
| #-best | | 1 | 6 | 1 | 4 | 0 | 0 | 0 |
| average | | 610 | 495 | 514 | 487 | 2756 | 2057 | 6258 |

In Table 6.7, the results obtained for H=25 are presented. The picture is similar to H=15, however the combination of the method of Neame and alternative dual-optimal solutions now performs better and is much faster than no stabilization for the instance C1-10/b=100.

However, although the overall performance trend is nearly the same as for H=15, the performance for instances based on the same graph and budget is different in some cases. For example, for C5-10/H=15/b=10, the performance of the performance second alternative dual-optimal solutions strategy is slightly worse than the other two alternative dual-optimal solutions strategies, but for C5-10/H=25/b=10 using the second strategy takes five times longer than using the other two.

Table 6.7: CPU-times in seconds with different stabilization techniques using formulation (DPFnt), NORMALPRICING and heuristic SHORTEST for instances based on Steinc graphs and $H = 25$. AD 1 denotes the first strategy based on alternative-dual optimal solutions (slack over all dual variables), AD2 the second (with parameter Q), and AD3 the third (slack only over dual variables with value zero). N1 denotes the method of Neame, N2 the combination of the method of Neame and the first strategy based on alternative-dual optimal solutions, PL piecewise linear stabilization.

| Instance | | Setting | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Graph | $b$ | NO | AD1 | AD2 | AD3 | N1 | N2 | PL |
| C3-10 | 10 | 2605 | **260** | 458 | 306 | 8113 | 1860 | 4778 |
| | 30 | 862 | 247 | **81** | 192 | 1102 | 339 | 577 |
| C4-10 | 10 | 177 | **53** | 153 | **53** | 1070 | 520 | 2588 |
| | 30 | 470 | 289 | 341 | **255** | 2960 | 2280 | 10000 |
| C5-10 | 10 | 3974 | **832** | 4475 | 833 | 10000 | 4397 | 10000 |
| | 30 | 1967 | 1069 | 717 | **577** | 9550 | 3633 | 10000 |
| C3-100 | 10 | **10000** | **10000** | **10000** | **10000** | **10000** | **10000** | **10000** |
| | 30 | 52 | 21 | 39 | **18** | 477 | 97 | 1448 |
| C4-100 | 10 | 188 | **42** | 78 | 48 | 1300 | 439 | 10000 |
| | 30 | 632 | **378** | 411 | 444 | 4258 | 2312 | 10000 |
| C5-100 | 10 | 3890 | 967 | **857** | 922 | 10000 | 4052 | 10000 |
| | 30 | **10000** | **10000** | **10000** | **10000** | **10000** | **10000** | **10000** |
| #-best | | 2 | 7 | 4 | 6 | 2 | 2 | 2 |
| average | | 2901 | 2013 | 2300 | 1970 | 5735 | 3327 | 7449 |

## 6.5 Comparison of Pricing Strategies

In the following, we will present the results of our experiments with different pricing strategies introduced in Section 5.6. Note that all pricing strategies are used together with MORECOLUMNS, so it is possible that more than one column with positive reduced costs is added for every terminal during a pricing iteration.

TYPICALPRICING denotes a pricing strategy, where pricing is stopped after the first terminal having a path to it with positive reduced costs has been found. NORMALPRICING solves the pricing subproblem for every terminal, thus multiple columns for every terminal can be added during a pricing iteration. TABUPRICING denotes a pricing strategy, where we keep a list of terminals for which paths with positive reduced costs have been found in one of the previous $k$ iterations. The pricing subproblem gets solved for terminals on this list, and only if no column with positive reduced costs is found, the pricing subproblem is also solved for the other terminals. Parameter $k$ has been set to three in our experiments. Pricing strategy HOPPRICING solves the pricing subproblem for smaller hop limits than the original hop limit and increases the hop limit and resolves the pricing subproblem in case no column with positive reduced cost has been found. This is done for every terminal until either we find a path with positive reduced costs or we reach the original hop limit, in this case we can stop solving, if no path with positive reduced costs has been found. Furthermore, NORMALPRICING and TABUPRICING have been tested in an extended version, where no pricing is performed on terminals, which have to be zero due to previous branching decisions.

Table 6.8 details the computational results when these pricing strategies are used in combination with formulation $(\mathrm{DPFnt})$, heuristic SHORTEST and no stabilization on instances based on the Steinc graphs C3, C4, C5 with $b = 10, 30$ and H=15

TYPICALPRICING has an average runtime, which is two to four times worse than the other strategies. This is no surprise, since columns are only added for one terminal during a pricing iteration and thus many more LPs need to be solved. TABUPRICING and its extended version are clearly the dominating strategies, they have the smallest average runtimes and aside from them, no other strategy gives a fastest runtime for an instance.

It is surprising, that for some instances, the extended strategies have longer runtime than the respective standard strategies, since the extended strategies do exactly the same as their standard counterparts, aside from not solving pricing subproblems, which do not return paths with positive reduced costs anyway. A closer examination showed, that obtaining the upper bounds on the $y$ variables took longer than solving unnecessary pricing subproblems. Thus, a more careful implementation (i.e. storing the branching decisions explicitly) of the extended strategies will likely result in better runtime (i.e. at least as good as their standard counterparts).

Table 6.8: CPU-times in seconds with different pricing techniques using formulation $(\mathrm{DPFnt})$, heuristic SHORTEST and no stabilization for instances based on Steinc graphs and $H = 15$. TY denotes TYPICALPRICING, N denotes NORMALPRICING, TA denotes TABUPRICING, HP denotes HOPPRICING and NE and TAE, respectively the extended versions of NORMALPRICING and TABUPRICING, respectively.

| Instance | | Setting | | | | | |
|---|---|---|---|---|---|---|---|
| Graph | $b$ | TY | N | TA | HP | NE | TAE |
| C3-10 | 10 | 772 | 266 | 249 | 350 | 286 | **226** |
| | 30 | 677 | 52 | 45 | 189 | 52 | **43** |
| C4-10 | 10 | 809 | 76 | **68** | 1326 | 81 | **68** |
| | 30 | 1043 | 253 | **185** | 356 | 318 | 196 |
| C5-10 | 10 | 2117 | 291 | 197 | 254 | 252 | **181** |
| | 30 | 1803 | 391 | **161** | 191 | 460 | 175 |
| C3-100 | 10 | 4223 | 2129 | **1789** | 1995 | 2657 | 1853 |
| | 30 | 268 | 15 | 12 | 22 | 13 | **11** |
| C4-100 | 10 | 3846 | 2026 | **1265** | 1677 | 1688 | 1400 |
| | 30 | 1596 | 559 | **431** | 539 | 553 | 508 |
| C5-100 | 10 | 2462 | 497 | 399 | 580 | 523 | **334** |
| | 30 | 3198 | 766 | 669 | 930 | 841 | **592** |
| #-best | | 0 | 0 | 6 | 0 | 0 | 7 |
| average | | 1901 | 610 | 455 | 700 | 643 | 465 |

Turning our attention to H=25 instead of H=15 (see Table 6.9), we encounter nearly the same trends, only the extended version of TABUPRICING performs a little bit worse now and thus TABUPRICING is the best for eleven out of twelve instances.

Table 6.9: CPU-times in seconds with different pricing techniques using formulation $(\mathrm{DPFnt})$, heuristic SHORTEST and no stabilization for instances based on Steinc graphs and $H = 25$. TY denotes TYPICALPRICING, N denotes NORMALPRICING, TA denotes TABUPRICING, HP denotes HOPPRICING and NE and TAE, respectively the extended versions of NORMALPRICING and TABUPRICING, respectively.

| Instance | | Setting | | | | | |
|---|---|---|---|---|---|---|---|
| Graph | b | TY | N | TA | HP | NE | TE |
| C3-10 | 10 | 5765 | 2605 | 2003 | 2041 | 2810 | **1999** |
| | 30 | 1680 | 862 | **254** | 899 | 1012 | 477 |
| C4-10 | 10 | 2573 | 177 | **149** | 160 | 174 | 160 |
| | 30 | 2350 | 470 | **249** | 488 | 481 | 334 |
| C5-10 | 10 | 9074 | 3974 | **3765** | 4013 | 4195 | 4623 |
| | 30 | 10000 | 1967 | **651** | 1744 | 2043 | 818 |
| C3-100 | 10 | **10000** | **10000** | **10000** | **10000** | **10000** | **10000** |
| | 30 | 814 | 52 | **33** | 75 | 47 | 40 |
| C4-100 | 10 | 2234 | 188 | **120** | 193 | 168 | 152 |
| | 30 | 3436 | 632 | **325** | 708 | 652 | 448 |
| C5-100 | 10 | 10000 | 3890 | **1664** | 3940 | 3701 | 2479 |
| | 30 | **10000** | **10000** | **10000** | **10000** | **10000** | **10000** |
| #-best | | 2 | 2 | 11 | 2 | 2 | 3 |
| average | | 5660 | 2901 | 2434 | 2855 | 2940 | 2627 |

## 6.6 In-depth Comparison of Three Settings

To end this chapter on results, we give an in-depth comparison of three different settings in combination with formulation $(\mathrm{DPFnt})$. The tests were run on the same instances as in [17]. Thus they can give a rough comparison of our branch-and-price approach to the branch-and-cut approaches presented in [17]. One must, however, be aware of the different environments (i.e. computers, CPLEX versions, ...) used to obtain the results. Aside from the runtime, we will also give the number of pricing iterations and the number of paths added (i.e. columns generated) to the RMP. Moreover, the runtime of three branch-and-cut approaches Costa has tested in [17] will be stated in the tables. Two of Costa's approaches are based on Garcia-Gouveia hop constraints (denoted by S1 and S3) and the other one is based on a directed Dantzig-Fulkerson-Johnson formulation (denoted by S5).

For our branch-and-price approach, we used the following settings: BP1 denotes a standard branch-and-price setting, i.e. NORMALPRICING and no heuristic and no stabilization. Setting BP2 consists of TABUPRICING, the third alternative dual-optimal solution strategy and no heuristic. BP3 uses heuristic SHORTEST, TABUPRICING and the first strategy based on alternative dual-optimal solutions.

The results will be grouped in tables according to the hop limits, i.e. H=3,6,9,12 for the instances based on MSteinb graphs and H=5,15,25 for the instances based on Steinc graphs. We begin the discussion with the hop limits which turned out to be easily solvable, meaning that the average runtime for solving the instances having this hop limit has been under five seconds. These hop limits are H=3,6,9,12 (i.e. all instances based on Msteinb graphs) and H=5. The results for these instances are presented in Tables 6.10, 6.11, 6.12, 6.13 and 6.14, respectively.

We conclude that our branch-and-price approach is competitive when the hop limit is small.

Regarding the budget for the instances with these hop limits, $b = 10$ seems to be more difficult to solve than $b = 5$ (and of course $b = 1$). A notable exception is the underlying graph B-6, where $b = 5$ results in a longer runtime than $b = 10$ for all three settings.

B-18/H=12/b=10 is the most difficult of the instances with these hop limits, with all three settings taking roughly three times as much to solve it than the second most difficult (which is B-12/H=12/b=10). For these two instances, settings BP2 and BP3 are approximately two times as fast as BP1 and for B-10/H=12/b=10 settings BP2 and BP3 deliver results instantly, while BP1 takes 20 seconds.

When $b = 1$ all settings need only very few pricing iterations, setting BP3 only needs one. This is not unexpected, since setting BP3 uses heuristic SHORTEST to add columns at the start and the other two settings start with an empty column set. Due to SHORTEST, the branch-and-price approach BP3 starts out for every terminal with a column representing the hop-constrained cheapest path to the terminal. As hinted above (see Section 6.2), this is already the optimal solution, since formulation $(\mathrm{DPFnt})$ does not care whether the solution is a tree or not. For $b = 5$ and $b = 10$, more pricing iterations are needed in general, however there are instances, where the numbers for $b = 1$ and $b = 10$ are nearly the same. Most of the time, BP3 needs the fewest number of pricing iterations, this is also apparent in the average number of pricing iterations.

When H=3 and H=5 the number of paths added to the RMP is nearly the same for all three approaches, but for higher H, BP2 and BP3 add fewer paths on average than BP1. This difference gets more pronounced as H grows, hinting that for larger hop limits, we may see a performance difference between BP1 and BP2, BP3.

Table 6.10: Detailed computational results for three settings using formulation $(\mathrm{DPFnt})$ for instances based on Msteinb graphs and $H = 3$. BP1 denotes a standard branch-and-price setting consisting of NORMALPRICING, no heuristic and no stabilization. BP2 uses TABUPRICING, no heuristic and the third strategy based on alternative dual-optimal solutions. BP3 consists of TABUPRICING, heuristic SHORTEST and the first strategy based on alternative dual-optimal solutions. Columns S1, S3 and S5 state the CPU-times reported by Costa et al. [17] for their branch-and-cut approaches.

| Instance | | CPU-time [s] | | | Iterations | | | Paths | | | Costa | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Graph | b | BP1 | BP2 | BP3 | BP1 | BP2 | BP3 | BP1 | BP2 | BP3 | S1 | S3 | S5 |
| B1 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **1** | **1** | 2 | 0 | 0 | 1 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | **1** | **1** | 2 | 0 | 0 | 2 |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **1** | **1** | 2 | 0 | 0 | 0 |
| B2 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **2** | **2** | 3 | 0 | 0 | 5 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | **2** | **2** | 3 | 0 | 0 | 2 |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **2** | **2** | 3 | 0 | 0 | 0 |
| B3 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 0 | 0 | 0 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 0 | 0 | 0 |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 0 | 0 | 0 |
| B4 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 0 | 0 | [inf] |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 0 | 0 | [inf] |
| | 10 | **0** | **0** | **0** | 3 | 3 | **1** | 7 | 7 | **5** | 0 | 0 | [inf] |
| B5 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **12** | **12** | **12** | 0 | 0 | [inf] |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | **12** | **12** | **12** | 0 | 0 | [inf] |
| | 10 | **0** | **0** | **0** | 4 | 4 | **3** | 18 | **17** | **17** | 0 | 0 | 4416 |
| B6 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **17** | **17** | 18 | 0 | 0 | [inf] |
| | 5 | **0** | **0** | **0** | 5 | 5 | **4** | **33** | **33** | 34 | 0 | 0 | [inf] |
| | 10 | **0** | **0** | **0** | 19 | 18 | **17** | 36 | 36 | **35** | 0 | 0 | 4432 |
| B7 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **1** | **1** | 2 | 0 | 0 | 2 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | **1** | **1** | 2 | 0 | 0 | 0 |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **1** | **1** | 2 | 0 | 0 | 0 |
| B8 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **1** | **1** | 2 | 0 | 0 | 3 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | **1** | **1** | 2 | 0 | 0 | 2 |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **1** | **1** | 2 | 0 | 0 | 0 |
| B9 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **12** | **12** | 13 | 0 | 0 | 4228 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | **12** | **12** | 13 | 0 | 0 | 1087 |
| | 10 | **0** | **0** | **0** | 14 | 14 | **13** | **12** | **12** | 13 | 0 | 0 | 19 |
| B10 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **6** | **6** | 7 | 0 | 0 | [inf] |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | **6** | **6** | 7 | 0 | 2 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **6** | **6** | 7 | 0 | 1 | [inf] |
| B11 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **7** | **7** | 8 | 0 | 0 | [inf] |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | **7** | **7** | 8 | 0 | 0 | [inf] |
| | 10 | **0** | **0** | **0** | 7 | 7 | **6** | **8** | **8** | 9 | 0 | 3 | [inf] |
| B12 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **24** | **24** | 25 | 0 | 0 | [inf] |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | **24** | **24** | 25 | 0 | 0 | [inf] |
| | 10 | **0** | **0** | **0** | 42 | 49 | **29** | **35** | **35** | 36 | 0 | 0 | [inf] |
| B13 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **2** | **2** | 3 | 0 | 0 | [inf] |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | **2** | **2** | 3 | 0 | 0 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **2** | **2** | 3 | 0 | 0 | 162 |
| B14 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **3** | **3** | 4 | 0 | 0 | 54 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | **3** | **3** | 4 | 0 | 0 | 95 |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **3** | **3** | 4 | 0 | 0 | 15 |
| B15 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **18** | **18** | 19 | 0 | 0 | [inf] |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | **18** | **18** | 19 | 0 | 0 | [inf] |
| | 10 | **0** | **0** | **0** | 5 | 5 | **4** | **19** | **19** | 20 | 0 | 0 | 28 |
| B16 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **9** | **9** | 10 | 0 | 0 | [inf] |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | **9** | **9** | 10 | 0 | 0 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **9** | **9** | 10 | 0 | 7 | [inf] |
| B17 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **5** | **5** | 6 | 0 | 0 | [inf] |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | **5** | **5** | 6 | 0 | 0 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **5** | **5** | 6 | 0 | 0 | [inf] |
| B18 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **25** | **25** | 26 | 0 | 0 | [inf] |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | **25** | **25** | 26 | 0 | 0 | [inf] |
| | 10 | **0** | **0** | **0** | 33 | 33 | **31** | **37** | **37** | **37** | 0 | 3 | [inf] |
| #-best | | 54 | 54 | 54 | 0 | 0 | 54 | 51 | 52 | 6 | | | |
| average | | 0.00 | 0.00 | 0.00 | 4 | 4 | 2 | 9 | 9 | 10 | | | |

Table 6.11: Detailed computational results for three settings using formulation $(\mathrm{DPFnt})$ for instances based on Msteinb graphs and $H = 6$. BP1 denotes a standard branch-and-price setting consisting of NORMALPRICING, no heuristic and no stabilization. BP2 uses TABUPRICING, no heuristic and the third strategy based on alternative dual-optimal solutions. BP3 consists of TABUPRICING, heuristic SHORTEST and the first strategy based on alternative dual-optimal solutions. Columns S1, S3 and S5 state the CPU-times reported by Costa et al. [17] for their branch-and-cut approaches.

| Instance | | CPU-time [s] | | | Iterations | | | Paths | | | Costa | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Graph | b | BP1 | BP2 | BP3 | BP1 | BP2 | BP3 | BP1 | BP2 | BP3 | S1 | S3 | S5 |
| B1 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **7** | **7** | 8 | 0 | 0 | 8 |
| | 5 | **0** | **0** | **0** | 3 | 3 | **1** | 16 | 15 | **8** | 0 | 0 | 0 |
| | 10 | **0** | **0** | **0** | **9** | 10 | **9** | 26 | **25** | 27 | 0 | 0 | 0 |
| B2 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **11** | **11** | 12 | 0 | 0 | 53 |
| | 5 | **0** | **0** | **0** | **10** | **10** | **10** | **22** | **22** | 24 | 0 | 0 | 1 |
| | 10 | **0** | **0** | **0** | **25** | 29 | **25** | **23** | 25 | 24 | 0 | 0 | 0 |
| B3 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **20** | **20** | 21 | 0 | 0 | 18 |
| | 5 | **0** | **0** | **0** | **22** | **22** | 89 | **25** | **25** | 27 | 0 | 0 | 1 |
| | 10 | **0** | **0** | **0** | 47 | 47 | **22** | **26** | **26** | 27 | 0 | 0 | 0 |
| B4 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 12 | 12 | **9** | 0 | 0 | 0 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | 12 | 12 | **9** | 0 | 0 | 6 |
| | 10 | **0** | **0** | **0** | 50 | **34** | 36 | 219 | **155** | 160 | 0 | 1 | 11 |
| B5 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 15 | 15 | **13** | 0 | 0 | 0 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | 15 | 15 | **13** | 0 | 1 | 0 |
| | 10 | **0** | **0** | **0** | 68 | 50 | **35** | 217 | 176 | **154** | 0 | 0 | 0 |
| B6 | 1 | **0** | **0** | **0** | 9 | 9 | **1** | 37 | 37 | **25** | 0 | 0 | 0 |
| | 5 | 1 | 1 | **0** | 60 | 84 | **36** | 434 | 426 | **376** | 0 | 0 | 740 |
| | 10 | 2 | **1** | **1** | 121 | 103 | **97** | 452 | 439 | **430** | 0 | 1 | 14 |
| B7 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **10** | **10** | 11 | 0 | 0 | [inf] |
| | 5 | **0** | **0** | **0** | **2** | **2** | **2** | **10** | **10** | 13 | 0 | 0 | 1 |
| | 10 | **0** | **0** | **0** | 28 | 28 | **27** | **12** | **12** | 13 | 0 | 0 | 0 |
| B8 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **12** | **12** | 13 | 0 | 0 | 6612 |
| | 5 | **0** | **0** | **0** | 12 | 12 | **11** | **15** | **15** | 16 | 0 | 1 | 2307 |
| | 10 | **0** | **0** | **0** | 27 | 26 | **21** | **15** | **15** | 16 | 0 | 2 | 4 |
| B9 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 40 | 40 | **37** | 0 | 0 | [inf] |
| | 5 | **1** | **1** | 2 | 94 | **81** | 165 | 139 | **132** | 142 | 0 | 0 | 17 |
| | 10 | **1** | **1** | 2 | 96 | 101 | **93** | **140** | **140** | 141 | 0 | 0 | 0 |
| B10 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 18 | 18 | **13** | 0 | 0 | [inf] |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | 18 | 18 | **13** | 0 | 1 | 9 |
| | 10 | 6 | 2 | **1** | 392 | 149 | **121** | 552 | 348 | **308** | 1 | 2 | 34 |
| B11 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 26 | 26 | **19** | 0 | 0 | [inf] |
| | 5 | **0** | **0** | **0** | 3 | 3 | **1** | 61 | 76 | **19** | 0 | 9 | [inf] |
| | 10 | **0** | **0** | **0** | 24 | 26 | **21** | 265 | **242** | 245 | 0 | 7 | 80 |
| B12 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 53 | 53 | **38** | 0 | 0 | [inf] |
| | 5 | **1** | 5 | **1** | **46** | 220 | 65 | **445** | 578 | 459 | 0 | 3 | [inf] |
| | 10 | **1** | **1** | 2 | **59** | 73 | 65 | 429 | 428 | **410** | 0 | 0 | 14 |
| B13 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **16** | **16** | 17 | 0 | 0 | 1415 |
| | 5 | **0** | **0** | **0** | 57 | **52** | 55 | **39** | **39** | 41 | 0 | 1 | 696 |
| | 10 | **0** | **0** | **0** | 27 | 28 | **26** | **41** | **41** | 42 | 0 | 1 | 1 |
| B14 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **22** | **22** | 23 | 0 | 1 | [inf] |
| | 5 | **0** | **0** | **0** | 60 | 60 | **59** | **28** | **28** | 29 | 0 | 2 | 30 |
| | 10 | **0** | **0** | **0** | 33 | 33 | **32** | **28** | **28** | 29 | 0 | 0 | 0 |
| B15 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **50** | **50** | **50** | 0 | 0 | [inf] |
| | 5 | **2** | **2** | **2** | 88 | 85 | **80** | 142 | 138 | **136** | 0 | 1 | 4 |
| | 10 | 5 | **4** | **4** | 144 | 159 | **142** | **145** | 146 | 147 | 0 | 1 | 0 |
| B16 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 21 | 21 | **17** | 0 | 11 | [inf] |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | 21 | 21 | **17** | 0 | 13 | [inf] |
| | 10 | **0** | **0** | **0** | **56** | 60 | **56** | 312 | 316 | **281** | 0 | 8 | [inf] |
| B17 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 28 | 28 | **25** | 2 | 6 | [inf] |
| | 5 | **0** | **0** | **0** | 3 | 3 | **1** | 49 | 54 | **25** | 0 | 18 | [inf] |
| | 10 | **0** | **0** | **0** | 17 | 14 | **12** | 144 | 136 | **128** | 0 | 9 | [inf] |
| B18 | 1 | **0** | **0** | **0** | 7 | 7 | **1** | 75 | 75 | **50** | 0 | 0 | [inf] |
| | 5 | 2 | **1** | 2 | 27 | 38 | **26** | 526 | 543 | **520** | 0 | 1 | [inf] |
| | 10 | 1 | **0** | **0** | 24 | 16 | **14** | 560 | 482 | **461** | 0 | 2 | [inf] |
| #-best | | 48 | 51 | 50 | 8 | 6 | 48 | 23 | 24 | 28 | | | |
| average | | 0.43 | 0.35 | 0.31 | 33 | 31 | 27 | 112 | 107 | 98 | | | |

Table 6.12: Detailed computational results for three settings using formulation $(\mathrm{DPFnt})$ for instances based on Msteinb graphs and $H = 9$. BP1 denotes a standard branch-and-price setting consisting of NORMALPRICING, no heuristic and no stabilization. BP2 uses TABUPRICING, no heuristic and the third strategy based on alternative dual-optimal solutions. BP3 consists of TABUPRICING, heuristic SHORTEST and the first strategy based on alternative dual-optimal solutions. Columns S1, S3 and S5 state the CPU-times reported by Costa et al. [17] for their branch-and-cut approaches.

| Instance | | CPU-time [s] | | | Iterations | | | Paths | | | Costa | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Graph | b | BP1 | BP2 | BP3 | BP1 | BP2 | BP3 | BP1 | BP2 | BP3 | S1 | S3 | S5 |
| B1 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **8** | **8** | 9 | 0 | 0 | 0 |
| | 5 | **0** | **0** | **0** | 30 | 29 | **22** | 67 | 54 | **50** | 0 | 0 | 0 |
| | 10 | **0** | **0** | **0** | 26 | **22** | **22** | 69 | **59** | 60 | 0 | 0 | 0 |
| B2 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **12** | **12** | 13 | 0 | 0 | 0 |
| | 5 | **0** | **0** | **0** | 32 | 28 | **26** | 112 | 101 | **88** | 0 | 0 | 0 |
| | 10 | **0** | **0** | **0** | 48 | 48 | **45** | 66 | 76 | **62** | 0 | 0 | 0 |
| B3 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **25** | **25** | **25** | 0 | 3 | 0 |
| | 5 | **0** | **0** | **0** | 51 | **47** | 111 | 90 | **85** | 103 | 0 | 0 | 0 |
| | 10 | **0** | **0** | **0** | 12 | 8 | **5** | 74 | 74 | **67** | 0 | 0 | 0 |
| B4 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 12 | 12 | **9** | 0 | 6 | 0 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | 12 | 12 | **9** | 0 | 0 | 0 |
| | 10 | **0** | **0** | **0** | 42 | **23** | 31 | 527 | 329 | **275** | 0 | 0 | 0 |
| B5 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 15 | 15 | **13** | 0 | 0 | 0 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | 15 | 15 | **13** | 1 | 0 | 0 |
| | 10 | **0** | **0** | **0** | 29 | 28 | **18** | 267 | 231 | **202** | 0 | 0 | 0 |
| B6 | 1 | **0** | **0** | **0** | 9 | 9 | **1** | 37 | 37 | **25** | 0 | 0 | 0 |
| | 5 | 6 | **5** | 7 | **226** | 321 | 305 | **1507** | 1874 | 1956 | 2 | 2 | 10 |
| | 10 | **4** | 5 | **4** | 251 | 226 | **172** | 1688 | 1402 | **1212** | 0 | 0 | 0 |
| B7 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **13** | **13** | **13** | 0 | 0 | 2 |
| | 5 | **0** | **0** | **0** | **10** | **10** | **10** | 45 | **43** | 46 | 0 | 0 | 0 |
| | 10 | **0** | **0** | **0** | 64 | 66 | **62** | 59 | 56 | **54** | 0 | 0 | 0 |
| B8 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 24 | 24 | **19** | 0 | 0 | 0112 |
| | 5 | **0** | **0** | **0** | 97 | **90** | 98 | 128 | **127** | 138 | 0 | 5 | 28 |
| | 10 | **0** | **0** | **0** | **25** | 26 | 29 | **104** | 107 | 105 | 0 | 1 | 0 |
| B9 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 41 | 41 | **38** | 0 | 3 | 3138 |
| | 5 | 1 | **0** | **0** | 48 | 46 | **44** | 274 | **260** | 283 | 0 | 0 | 0 |
| | 10 | 8 | **4** | 6 | 281 | 270 | **259** | **466** | 485 | 525 | 0 | 0 | 0 |
| B10 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 18 | 18 | **13** | 2 | 0 | 1 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | 18 | 18 | **13** | 1 | 1 | 1 |
| | 10 | 1 | **0** | **0** | 121 | **66** | 69 | 834 | 618 | **547** | 0 | 2 | 0 |
| B11 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 26 | 26 | **19** | 3 | 15 | [inf] |
| | 5 | **0** | **0** | **0** | 3 | 3 | **1** | 61 | 86 | **19** | 6 | 2 | 0 |
| | 10 | 2 | 2 | **1** | 70 | 115 | **66** | 1536 | 1444 | **1184** | 1 | 3 | 0 |
| B12 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 53 | 53 | **38** | 4 | 4 | 1 |
| | 5 | **1** | **1** | **1** | 48 | 48 | **45** | 801 | **690** | 747 | 0 | 1 | 5 |
| | 10 | 14 | 11 | **6** | 362 | 346 | **233** | 2235 | 1840 | **1299** | 1 | 0 | 1 |
| B13 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **16** | **16** | 17 | 0 | 0 | 12 |
| | 5 | **0** | **0** | **0** | 63 | 74 | **54** | 206 | 202 | **195** | 0 | 0 | 0 |
| | 10 | 5 | **1** | 4 | 382 | **177** | 370 | 267 | **243** | 268 | 0 | 1 | 0 |
| B14 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **25** | **25** | **25** | 0 | 10 | 1 |
| | 5 | **1** | **1** | **1** | 90 | 90 | **82** | 127 | 122 | **121** | 0 | 0 | 2 |
| | 10 | **1** | **1** | **1** | **73** | 81 | 78 | **119** | 122 | 124 | 0 | 0 | 0 |
| B15 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 51 | 51 | **50** | 0 | 6 | 26 |
| | 5 | **0** | **0** | **0** | 15 | 24 | **11** | 263 | **253** | 266 | 0 | 0 | 0 |
| | 10 | 8 | **6** | 7 | 259 | **218** | 223 | 533 | **471** | 484 | 2 | 1 | 0 |
| B16 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 21 | 21 | **17** | 3 | 36 | 167 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | 21 | 21 | **17** | 3 | 2 | 1 |
| | 10 | 2 | **1** | **1** | 89 | **43** | 55 | 1262 | 1063 | **1044** | 7 | 0 | 118 |
| B17 | 1 | **0** | **0** | **0** | 14 | 8 | **1** | 110 | 98 | **25** | 4 | 0 | [inf] |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | 33 | 33 | **25** | 4 | 21 | [inf] |
| | 10 | 3 | **2** | **2** | 109 | 100 | **93** | 976 | 774 | **763** | 2 | 23 | 5320 |
| B18 | 1 | **0** | **0** | **0** | 6 | 6 | **1** | 76 | 76 | **50** | 7 | 29 | 75 |
| | 5 | 2 | **1** | **1** | 26 | 13 | **9** | 1005 | **746** | 832 | 0 | 0 | 1613 |
| | 10 | 9 | 5 | **4** | 170 | 137 | **129** | 1442 | 1299 | **1287** | 0 | 0 | 35 |
| #-best | | 42 | 50 | 50 | 4 | 9 | 44 | 10 | 16 | 37 | | | |
| average | | 1.26 | 0.85 | 0.85 | 59 | 53 | 51 | 331 | 296 | 275 | | | |

Table 6.13: Detailed computational results for three settings using formulation $(\mathrm{DPFnt})$ for instances based on Msteinb graphs and $H = 12$. BP1 denotes a standard branch-and-price setting consisting of NORMALPRICING, no heuristic and no stabilization. BP2 uses TABUPRICING, no heuristic and the third strategy based on alternative dual-optimal solutions. BP3 consists of TABUPRICING, heuristic SHORTEST and the first strategy based on alternative dual-optimal solutions. Columns S1, S3 and S5 state the CPU-times reported by Costa et al. [17] for their branch-and-cut approaches.

| Instance | | CPU-time [s] | | | Iterations | | | Paths | | | Costa | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Graph | b | BP1 | BP2 | BP3 | BP1 | BP2 | BP3 | BP1 | BP2 | BP3 | S1 | S3 | S5 |
| B1 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **8** | **8** | 9 | 0 | 0 | 0 |
| | 5 | **0** | **0** | **0** | 34 | 31 | **18** | 84 | 58 | **50** | 0 | 0 | 0 |
| | 10 | **0** | **0** | **0** | 28 | **22** | 23 | 75 | **66** | 70 | 0 | 0 | 0 |
| B2 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **12** | **12** | 13 | 0 | 0 | 0 |
| | 5 | **0** | **0** | **0** | 37 | 40 | **31** | 141 | 132 | **127** | 0 | 0 | 0 |
| | 10 | **0** | **0** | **0** | 56 | 50 | **27** | 99 | 93 | **77** | 0 | 0 | 0 |
| B3 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **25** | **25** | **25** | 0 | 0 | 0 |
| | 5 | **0** | **0** | **0** | 69 | 84 | **54** | 144 | 172 | **137** | 0 | 0 | 0 |
| | 10 | **0** | **0** | **0** | 12 | 10 | **6** | 99 | 95 | **84** | 0 | 0 | 0 |
| B4 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 12 | 12 | **9** | 2 | 0 | 0 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | 12 | 12 | **9** | 6 | 0 | 0 |
| | 10 | **0** | **0** | **0** | 86 | 9 | **7** | 779 | 236 | **165** | 1 | 0 | 0 |
| B5 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 15 | 15 | **13** | 2 | 0 | 0 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | 15 | 15 | **13** | 5 | 0 | 0 |
| | 10 | **0** | **0** | **0** | 45 | 28 | **17** | 291 | 233 | **208** | 0 | 0 | 0 |
| B6 | 1 | **0** | **0** | **0** | 9 | 9 | **1** | 37 | 37 | **25** | 2 | 0 | 0 |
| | 5 | 11 | 6 | **5** | 499 | **278** | 338 | 3270 | **1997** | 2590 | 21 | 2 | 0 |
| | 10 | 7 | **4** | **4** | 451 | 274 | **261** | 2647 | 2156 | **2048** | 2 | 0 | 0 |
| B7 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **13** | **13** | **13** | 0 | 0 | 0 |
| | 5 | **0** | **0** | **0** | **10** | **10** | **10** | 93 | **82** | 87 | 0 | 0 | 0 |
| | 10 | **0** | **0** | **0** | 109 | 111 | **98** | 151 | 159 | **150** | 0 | 0 | 0 |
| B8 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 24 | 24 | **19** | 0 | 0 | 1 |
| | 5 | **0** | **0** | **0** | **33** | **33** | 47 | 274 | **256** | 266 | 0 | 2 | 0 |
| | 10 | **0** | **0** | **0** | 55 | 45 | **38** | 246 | 214 | **206** | 0 | 5 | 0 |
| B9 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 41 | 41 | **38** | 0 | 0 | 0 |
| | 5 | **1** | **1** | **1** | 57 | **55** | 59 | 390 | **372** | 391 | 0 | 0 | 0 |
| | 10 | **7** | **7** | **7** | **318** | 348 | 347 | **729** | 825 | 900 | 0 | 0 | 0 |
| B10 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 18 | 18 | **13** | 10 | 0 | 1 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | 18 | 18 | **13** | 21 | 0 | 1 |
| | 10 | 20 | 2 | **1** | 1128 | 189 | **97** | 5480 | 1550 | **963** | 3 | 0 | 0 |
| B11 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 26 | 26 | **19** | 8 | 0 | 1 |
| | 5 | **0** | **0** | **0** | 3 | 3 | **1** | 61 | 86 | **19** | 16 | 3 | 0 |
| | 10 | 6 | **2** | **2** | 175 | **92** | 106 | 2926 | **1663** | 1828 | 5 | 9 | 0 |
| B12 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 53 | 53 | **38** | 10 | 0 | 1 |
| | 5 | 4 | **1** | 3 | 142 | 115 | **101** | 1214 | **1005** | 1162 | 0 | 0 | 0 |
| | 10 | 31 | **17** | **17** | 833 | 580 | **579** | 4194 | **3490** | 3636 | 1 | 0 | 0 |
| B13 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **16** | **16** | 17 | 0 | 0 | 0 |
| | 5 | **1** | **1** | **1** | **127** | 161 | 148 | 544 | 533 | **478** | 1 | 0 | 0 |
| | 10 | 8 | **2** | **2** | 565 | **187** | 190 | 726 | **414** | 423 | 1 | 3 | 0 |
| B14 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **25** | **25** | **25** | 0 | 0 | 1 |
| | 5 | 3 | 3 | **2** | 153 | 157 | **139** | 365 | **327** | 350 | 0 | 0 | 0 |
| | 10 | **2** | **2** | 3 | 137 | 140 | **128** | 355 | 353 | **337** | 0 | 0 | 0 |
| B15 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 51 | 51 | **50** | 0 | 0 | 1 |
| | 5 | **0** | **0** | **0** | **19** | 32 | **19** | 347 | 374 | **328** | 0 | 0 | 0 |
| | 10 | 4 | 4 | **3** | 113 | **84** | 87 | 570 | 531 | **510** | 0 | 0 | 0 |
| B16 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 21 | 21 | **17** | 10 | 1 | 9 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | 21 | 21 | **17** | 28 | 4 | 2 |
| | 10 | 4 | 6 | **1** | 126 | 310 | **46** | 2392 | 3427 | **1407** | 16 | 3 | 4 |
| B17 | 1 | **0** | **0** | **0** | 14 | 8 | **1** | 110 | 98 | **25** | 10 | 38 | 9 |
| | 5 | **0** | **0** | **0** | 2 | 2 | **1** | 33 | 33 | **25** | 19 | 6 | 0 |
| | 10 | 16 | 9 | **4** | 332 | 355 | **171** | 3115 | 2955 | **1560** | 0 | 0 | 0 |
| B18 | 1 | **0** | **0** | **0** | 6 | 6 | **1** | 76 | 76 | **50** | 16 | 0 | 4 |
| | 5 | 2 | **1** | **1** | 24 | 13 | **10** | 990 | **765** | 845 | 20 | 3 | 013 |
| | 10 | 113 | **48** | 49 | 2008 | 979 | **942** | 6742 | 5411 | **4994** | 14 | 5 | 5 |
| #-best | | 41 | 48 | 51 | 5 | 8 | 45 | 7 | 17 | 39 | | | |
| average | | 4.44 | 2.15 | 1.96 | 145 | 90 | 77 | 744 | 568 | 497 | | | |

Table 6.14: Detailed computational results for three settings using formulation $(\mathrm{DPFnt})$ for instances based on Steinc graphs and $H = 5$. BP1 denotes a standard branch-and-price setting consisting of NORMALPRICING, no heuristic and no stabilization. BP2 uses TABUPRICING, no heuristic and the third strategy based on alternative dual-optimal solutions. BP3 consists of TABUPRICING, heuristic SHORTEST and the first strategy based on alternative dual-optimal solutions. Columns S1, S3 and S5 state the CPU-times reported by Costa et al. [17] for their branch-and-cut approaches.

| Instance | | CPU-time [s] | | | Iterations | | | Paths | | | Costa | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Graph | b | BP1 | BP2 | BP3 | BP1 | BP2 | BP3 | BP1 | BP2 | BP3 | S1 | S3 | S5 |
| C1-10 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **1** | **1** | 2 | 0 | 34 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **1** | **1** | 2 | 0 | 186 | [inf] |
| | 30 | **0** | **0** | **0** | 2 | 2 | **1** | **1** | **1** | 2 | 0 | 161 | [inf] |
| C2-10 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 0 | 10 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 0 | 308 | [inf] |
| | 30 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 0 | 105 | [inf] |
| C3-10 | 1 | **0** | **0** | **0** | 22 | 22 | **1** | **29** | **29** | 30 | 0 | 4 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **29** | **29** | 30 | 0 | 66 | [inf] |
| | 30 | **2** | **2** | **2** | **102** | 104 | 103 | **47** | **47** | **47** | 0 | 48 | [inf] |
| C4-10 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **21** | **21** | 22 | 0 | 2 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **21** | **21** | 22 | 0 | 12 | [inf] |
| | 30 | **0** | **0** | 1 | **53** | **53** | 246 | **22** | **22** | 23 | 0 | 102 | [inf] |
| C5-10 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **48** | **48** | 49 | 0 | 1 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **48** | **48** | 49 | 0 | 14 | [inf] |
| | 30 | **0** | **0** | **0** | 6 | 6 | **5** | **58** | **58** | 59 | 0 | 138 | [inf] |
| C1-100 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **1** | **1** | 2 | 0 | 34 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **1** | **1** | 2 | 0 | 536 | [inf] |
| | 30 | **0** | **0** | **0** | 2 | 2 | **1** | **1** | **1** | 2 | 0 | 416 | [inf] |
| C2-100 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 0 | 10 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 0 | 202 | [inf] |
| | 30 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 0 | 124 | [inf] |
| C3-100 | 1 | **0** | **0** | **0** | 21 | 21 | **1** | **29** | **29** | 30 | 0 | 5 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **29** | **29** | 30 | 0 | 33 | [inf] |
| | 30 | 2 | **1** | **1** | 62 | 65 | **60** | **45** | 49 | 47 | 0 | 65 | [inf] |
| C4-100 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **21** | **21** | 22 | 0 | 3 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **21** | **21** | 22 | 0 | 30 | [inf] |
| | 30 | **0** | **0** | 1 | **86** | **86** | 149 | **22** | **22** | 23 | 0 | 109 | [inf] |
| C5-100 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **48** | **48** | 49 | 0 | 1 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **48** | **48** | 49 | 0 | 24 | [inf] |
| | 30 | **0** | **0** | **0** | 5 | 5 | **4** | **58** | **58** | 59 | 0 | 71 | [inf] |
| #-best | | 29 | 30 | 28 | 3 | 2 | 27 | 30 | 29 | 1 | | | |
| average | | 0.13 | 0.10 | 0.17 | 13 | 13 | 19 | 22 | 22 | 23 | | | |

We now turn our attention to the instances, which proved to be more difficult (when $b \neq 1$). These are the instances based on Steinc graphs with H=15 and H=25. The results can be seen in Tables 6.15 and 6.16. To be more precise, only graphs C3, C4 and C5 presented real challenges to our approaches, while instances based on C1 and C2 are solved relatively fast, in particular, when $b \neq 30$. Since both C1 and C2 have a small number of terminals in contrast to the three other graphs, this leads to the conclusion, that our branch-and-price approach is competitive, when the number of terminals is small. Observing that for instances with smaller hop limit, the most difficult instances have a high number of terminals further strengthens this conclusion. However, there are also two instances C3-100/b=30/H=15 and C3-100/b=30/H=25, which do

not fit into this scheme and still are solved really fast by all three approaches.

For instances with H=15, BP2 and BP3 give half the average runtime than BP1, showing that acceleration strategies pay off, when the hop limit gets larger. While for most of these instances BP2 and BP3 perform equally good, there are notable exceptions like C4-10/b=30, C5-10/b=30 and C5-100/b=30, where the runtime of BP3 is approximately half the runtime of BP2. Thus, for the instances with H=15 and $b = 30$ BP3 seems to be a little bit better than BP2.

When H=25, all three average runtimes are about five times as high as for H=15. Now BP3 has a clear better average runtime than BP2, for instance C5-10/b=10, using BP2 takes six times as long as using BP3. BP2 is even worse than BP3 for this particular instance. Interestingly, instance C5-100/b=30/H=25 is solvable within time limit using BP2 or BP3. This has not been possible using only heuristics, stabilization techniques or pricing strategies on their own or even a combination of SHORTEST with stabilization techniques or pricing strategies (see the previous sections in this chapter). On the other hand, instance C3-100/b=10/H=25 is not solved within time limit by all three of our approaches. However, also the two approaches by Costa et al. based on Garcia-Gouveia hop constraints also do not manage to solve this instance. Although the instances with H=25 are harder to solve in general, instances based on graph C4-100 are a notable exception, especially for $b = 10$: For instance C4-100/b=10/H=15 all three settings needed about 15 times as long as for C4-100/b=10/H=25.

Concerning the number of iterations and added paths, the picture already apparent for smaller hop limits also holds for larger hop limits. Setting BP1 has both the highest average number of iterations and added paths, especially for the instances with H=25. In contrast to the instances with smaller hop limit, this higher average now has a clear impact on the runtime. This effect is not unexpected, since the number of pricing iterations is proportional to the number of LPs solved and the number of added paths is proportional to the size of the LPs.

It is interesting to notice, that the number of added paths grows faster as the number of iterations when comparing the numbers for H=15 and H=25. The number of iterations not even doubles for BP2 and BP3, but the number of added paths for instances with H=25 is almost three times as high for for instances with H=15. This means, more paths per iteration are added for H=25, which is not too surprising, since a higher hop limit also gives a higher number of possible paths.

Thus, we conclude that when the underlying graphs and hop limits get larger, acceleration techniques reduce the runtimes of our branch-and-price approach. Still, this acceleration is not enough to make branch-and-price competitive with the branch-and-cut approaches presented by Costa et al. for instances with large hop limit or a large number of terminals.

The observed different performance depending on the number of terminals and the hop limit of our approaches based on formulation $(\mathrm{DPFnt})$ can be explained theoretically as follows: Essentially the approaches search hop-constrained paths to a subset of all terminals, in such a way, that the sum of edge-costs in this paths is lower or equal to the budget and the revenue of the terminals connected with this paths is maximized. Thus, when the number of terminals is low, the number of different subsets of terminals is also low and the approaches are likely to perform better. Likewise, when the hop limit gets larger, the set of possible paths also gets bigger and thus our approaches have to try out more paths.

Table 6.15: Detailed computational results for three settings using formulation $(\mathrm{DPFnt})$ for instances based on Steinc graphs and $H = 15$. BP1 denotes a standard branch-and-price setting consisting of NORMALPRICING, no heuristic and no stabilization. BP2 uses TABUPRICING, no heuristic and the third strategy based on alternative dual-optimal solutions. BP3 consists of TABUPRICING, heuristic SHORTEST and the first strategy based on alternative dual-optimal solutions. Columns S1, S3 and S5 state the CPU-times reported by Costa et al. [17] for their branch-and-cut approaches.

| Instance | | CPU-time [s] | | | Iterations | | | Paths | | | Costa | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Graph | b | BP1 | BP2 | BP3 | BP1 | BP2 | BP3 | BP1 | BP2 | BP3 | S1 | S3 | S5 |
| C1-10 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 0 | 43 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 0 | 70 | [inf] |
| | 30 | **0** | **0** | **0** | 7 | 3 | **1** | 19 | 16 | **5** | 0 | 49 | 1 |
| C2-10 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **10** | **10** | **10** | 2 | 520 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **10** | **10** | **10** | 2 | 79 | [inf] |
| | 30 | **7** | 10 | 8 | 119 | 122 | **113** | 924 | 734 | **634** | 58 | 2 | 7 |
| C3-10 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 93 | 93 | **83** | 49 | 900 | 7105 |
| | 10 | 226 | 142 | **135** | 490 | 462 | **400** | 5136 | 4710 | **4129** | 114 | 397 | 28 |
| | 30 | 50 | 79 | **33** | 164 | 335 | **134** | 2781 | 3553 | **2084** | 1 | 35 | 0 |
| C4-10 | 1 | 1 | 1 | **0** | 2 | 2 | **1** | 136 | 136 | **125** | 19 | 368 | [inf] |
| | 10 | 222 | **144** | 199 | 423 | **325** | 432 | 4022 | **3308** | 3710 | 15 | 163 | 2740 |
| | 30 | 256 | 209 | **145** | 535 | 587 | **403** | 4820 | 5117 | **3962** | 13 | 37 | 348 |
| C5-10 | 1 | 11 | 11 | **1** | 2 | 2 | **1** | 289 | 289 | **250** | 45 | 871 | [inf] |
| | 10 | 247 | **126** | 129 | 152 | 154 | **122** | 3539 | 3455 | **3082** | 5 | 251 | 1234 |
| | 30 | 175 | 224 | **95** | 139 | 308 | **103** | 3685 | 4616 | **2918** | 8 | 82 | 93 |
| C1-100 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 0 | 43 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 0 | 347 | [inf] |
| | 30 | **0** | **0** | **0** | 11 | 3 | **1** | 54 | 16 | **5** | 0 | 29 | 1 |
| C2-100 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **10** | **10** | **10** | 2 | 521 | [inf] |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **10** | **10** | **10** | 2 | 450 | 229 |
| | 30 | 13 | 9 | **8** | 156 | 124 | **123** | 1050 | 727 | **672** | 224 | 11 | 3 |
| C3-100 | 1 | 2 | 2 | **0** | 12 | 12 | **1** | 93 | 93 | **83** | 50 | 96 | [inf] |
| | 10 | 1721 | 1227 | **1200** | 3234 | **2119** | 2147 | 14829 | **11758** | 12060 | 190 | 663 | 85 |
| | 30 | 18 | **8** | 9 | 60 | **41** | 50 | 1885 | **1600** | 1687 | 1 | 35 | 0 |
| C4-100 | 1 | 1 | 1 | **0** | 2 | 2 | **1** | 136 | 136 | **125** | 19 | 306 | [inf] |
| | 10 | 2080 | **820** | 874 | 2354 | 1707 | **1662** | 12284 | 9287 | **8415** | 77 | 509 | [inf] |
| | 30 | 437 | **256** | 325 | 864 | **660** | 782 | 6116 | **5404** | 5745 | 9 | 9 | 23 |
| C5-100 | 1 | 6 | 6 | **1** | 2 | 2 | **1** | 289 | 289 | **250** | 7 | 129 | 1465 |
| | 10 | 678 | 282 | **238** | 705 | 384 | **276** | 4760 | 4175 | **3938** | 8 | 82 | 93 |
| | 30 | 1603 | 708 | **429** | 1298 | 747 | **555** | 8679 | 6729 | **5344** | 22 | 50 | 9 |
| #-best | | 12 | 16 | 24 | 0 | 4 | 26 | 8 | 12 | 22 | | | |
| average | | 258.47 | 142.17 | 127.63 | 358 | 270 | 243 | 2522 | 2209 | 1978 | | | |

Table 6.16: Detailed computational results for three settings using formulation $(\mathrm{DPFnt})$ for instances based on Steinc graphs and $H = 25$. BP1 denotes a standard branch-and-price setting consisting of NORMALPRICING, no heuristic and no stabilization. BP2 uses TABUPRICING, no heuristic and the third strategy based on alternative dual-optimal solutions. BP3 consists of TABUPRICING, heuristic SHORTEST and the first strategy based on alternative dual-optimal solutions. Columns S1, S3 and S5 state the CPU-times reported by Costa et al. [17] for their branch-and-cut approaches.

| Instance | | CPU-time [s] | | | Iterations | | | Paths | | | Costa | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Graph | b | BP1 | BP2 | BP3 | BP1 | BP2 | BP3 | BP1 | BP2 | BP3 | S1 | S3 | S5 |
| C1-10 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 8 | 14 | 222 |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 8 | 878 | 43 |
| | 30 | **0** | **0** | **0** | 7 | 3 | **1** | 19 | 16 | **5** | 9 | 230 | 2 |
| C2-10 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **10** | **10** | **10** | 5 | 2 | 7 |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **10** | **10** | **10** | 5 | 57 | 7 |
| | 30 | 47 | **7** | 28 | 503 | **159** | 377 | 6534 | **1531** | 2182 | 431 | 24 | 8 |
| C3-10 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | 93 | 93 | **83** | 223 | 257 | 2 |
| | 10 | 2126 | 904 | **137** | 1863 | 947 | **658** | 25716 | 17830 | **8607** | 4293 | 3309 | 18 |
| | 30 | 297 | **102** | 114 | 574 | 512 | **465** | 10920 | 9108 | **8606** | 153 | 44 | 0 |
| C4-10 | 1 | 1 | 1 | **0** | 2 | 2 | **1** | 136 | 136 | **125** | 539 | [inf] | 112 |
| | 10 | 267 | **50** | 62 | 440 | 224 | **200** | 5380 | **3925** | 4120 | 14 | 141 | 2 |
| | 30 | 704 | 207 | **166** | 1015 | 423 | **404** | 11012 | 8316 | **8215** | 149 | 26 | 2 |
| C5-10 | 1 | 14 | 9 | **1** | 2 | 2 | **1** | 289 | 289 | **250** | 1009 | 150 | 98 |
| | 10 | 2862 | 3063 | **563** | 2269 | 2105 | **617** | 15463 | 25936 | **9395** | 139 | 550 | 21 |
| | 30 | 1342 | **342** | 407 | 1407 | 429 | **349** | 13194 | 9445 | **7647** | 1568 | 18 | 13 |
| C1-100 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 8 | 14 | 222 |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **4** | **4** | 5 | 8 | 682 | 43 |
| | 30 | **0** | **0** | **0** | 12 | 3 | **1** | 79 | 16 | **5** | 8 | 56 | 2 |
| C2-100 | 1 | **0** | **0** | **0** | 2 | 2 | **1** | **10** | **10** | **10** | 5 | 2 | 7 |
| | 10 | **0** | **0** | **0** | 2 | 2 | **1** | **10** | **10** | **10** | 5 | 183 | 7 |
| | 30 | 17 | **8** | 24 | 204 | **171** | 266 | 3115 | **1512** | 1763 | 362 | 24 | 4 |
| C3-100 | 1 | 2 | 2 | **0** | 12 | 12 | **1** | 93 | 93 | **83** | 184 | 120 | 2 |
| | 10 | **10000** | **10000** | **10000** | 3832 | 3184 | **2569** | **36319** | 42205 | 39086 | [inf] | [inf] | 36 |
| | 30 | 28 | 17 | **10** | 99 | 64 | **47** | 3113 | 2728 | **2084** | 8 | 13 | 0 |
| C4-100 | 1 | 1 | 1 | **0** | 2 | 2 | **1** | 136 | 136 | **125** | 542 | [inf] | 112 |
| | 10 | 148 | **47** | 66 | 284 | **156** | 208 | 6037 | **3815** | 4163 | 30 | 70 | 0 |
| | 30 | 632 | **211** | 214 | 1336 | 604 | **526** | 13228 | 10905 | **9615** | 115 | 57 | 8 |
| C5-100 | 1 | 5 | 5 | **1** | 2 | 2 | **1** | 289 | 289 | **250** | 1308 | 150 | 98 |
| | 10 | 3775 | 970 | **598** | 3676 | 793 | **614** | 20015 | 11519 | **10014** | 305 | 878 | 10 |
| | 30 | 10000 | **2938** | 3430 | 10331 | 2957 | **2824** | 48469 | **32514** | 34563 | 89 | 29 | 7 |
| #-best | | 12 | 20 | 22 | 0 | 3 | 27 | 9 | 13 | 20 | | | |
| average | | 1075.60 | 629.47 | 527.37 | 929 | 425 | 338 | 7323 | 6080 | 5034 | | | |

CHAPTER 7

# Conclusion and Outlook

In this thesis branch-and-price approaches based on several path-based ILP formulations for the STPRBH were presented. Different methods to accelerate the approaches were tried: These methods were heuristics to generate initial solutions, stabilization techniques for column generation and pricing strategies.

For the implemented heuristics to generate initial solutions, it turned out that using no heuristic gives the best results compared to other heuristics. A heuristic, which for every terminal adds the column corresponding to the hop-constrained cheapest path from the root to this terminal, also performed quite well. However, the effectiveness of the heuristics heavily depends on the instances, especially the hop limit.

Three different column generation stabilization methods based on alternative dual-optimal solutions were tested, as well as two different stabilization methods using the method of Neame and piecewise linear stabilization. Using any of the three stabilization methods based on alternative dual-optimal solutions resulted in a significantly lower runtime than using no stabilization. The other techniques performed rather poor and yielded longer runtimes than using no stabilization. With piecewise linear stabilization, most of the tested instances were not even solvable within a time limit of 10000 CPU-seconds.

Regarding pricing strategies, the best strategy consisted of keeping an active list of terminals and solving the pricing subproblem for terminals on this active list. Only if during a pricing iteration no column with positive reduced costs were found for any terminal on this active list, the pricing subproblem was also solved for the other terminals. For each tested instance, this strategy outperformed the adaption of typical pricing strategy used in branch-and-price, which solves the pricing subproblem for each terminal during every branch-and-price iteration.

Combinations of the above acceleration methods were also tested. For instances based on large graphs and large hop limits, branch-and-price using a combination of acceleration methods was roughly two times as fast as a standard branch-and-price approach.

For instances with small hop limits or a small number of terminals, the proposed branch-and-price approaches are competitive most of the time with the state of the art exact-methods [17] for solving the STPRBH. When the budget is big enough, that it is not restricting (i.e. when

the problem reduces to the STPRH), our best combination even outperforms these state of the art techniques from [17].

It should be easy to modify the introduced branch-and-price formulation for use in problems related to the STPRBH like the STPH or the PCSTP with added hop limit, a problem which to the author's knowledge, has not been described in the literature yet. Moreover, one can also replace the hop limit with delay constraints.

We also want to further investigate the dual programs to see if we can extract more helpful information for solving the primal and to see if we can find meaningful interpretations for some other dual program than $(\mathrm{DUAL})^{(\mathrm{DPF})}$.

Another goal for future work is the implementation of some other stabilization methods like weighted Dantzig-Wolfe decomposition [78] or interior-point stabilization [71] for this problem. The combination of alternative dual-optimal solutions with other stabilization methods also presents an interesting challenge.

# Bibliography


[1] T. Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

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

[4] J. Beasley. OR-library: Distributing test problems by electronic mail. *The Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[5] H. Ben-Amor and J. Desrosiers. A proximal-trust region alogirthm for column generation stabilization. *Computers & Operations Research*, 33:910–927, 2006.

[6] H. Ben-Amor, J. Desrosiers, and J. M. Valerio de Carvalho. Dual-optimal inequalites for stabilized column generation. *Operations Research*, 54(3):454–463, 2006.

[7] H. Ben-Armor, J. Desrosiers, and A. Frangioni. On the choice of explicit stabilizing terms in column generation. *Discrete Applied Mathematic*, 157(3):454–463, 2009.

[8] D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. Williamson. A note on the prize collecting traveling salesman problem. *Mathematical Programming*, 59(1):413–420, 1993.

[9] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[10] O. Chapovska and A. P. Punnen. Variations of the prize-collecting Steiner tree problem. *Networks*, 47(4):199–205, 2006.

[11] S. Chopra and M. R. Rao. The Steiner tree problem I: Formulations, compositions and extension of facets. *Mathematical Programming*, (64):209–229, 1994.

[12] S. Chopra and M. R. Rao. The Steiner tree problem II: Properties and classes of facets. *Mathematical Programming*, (64):231–246, 1994.

[13] E. Coffman and J. Bruno. *Computer and job-shop scheduling theory*. Wiley, 1976.

[14] A. Costa, J.-F. Cordeau, and G. F. Laporte. Steiner tree problems with profits. *INFOR*, 44:99–115, 2006.

[15] A. M. Costa. *Models and algorithms for two network design problems*. PhD thesis, Ecole des Hautes Etudes Commerciales - Montreal, 2006.

[16] A. M. Costa, J.-F. Cordeau, and G. Laporte. Fast heuristics for the Steiner tree problem with revenues, budget and hop constraints. *European Journal of Operational Research*, 190:68–78, 2008.

[17] A. M. Costa, J.-F. Cordeau, and G. Laporte. Models and branch-and-cut-algorithms for the Steiner tree problem with revenues, budget and hop constraints. *Networks*, 53(2):141–159, 2009.

[18] G. Dahl, L. Gouveia, and C. Requejo. On formulations and methods for the hop-constrained minimum spanning tree problem. *Handbook of Optimization in Telecommunications*, II:493–515, 2006.

[19] G. Dantzig. Programming of interdependent activities, II, mathematical model. *Econometrics*, 17(3,4):200–211, 1949.

[20] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

[21] G. Dantzig and M. N. Thapa. *Linear Programming - 1 : Introduction*. Springer, 2003.

[22] G. Dantzig and M. N. Thapa. *Linear Programming - 2 : Theory and Extensions*. Springer, 2003.

[23] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.

[24] J. Desrosiers and M. E. Luebbecke. A primer in column generation. In G. Desaulniers, J. Desrosiers, and M. Solomon, editors, *Column Generation*, pages 1–32. Springer, 2005.

[25] M. Dorigo, G. D. Di Caro, and L. Gambardella. Ant colony optimization: A new meta-heuristic. In P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1470–1477, Mayflower Hotel, Washington D.C., USA, 1999. IEEE Press.

[26] O. Du Merle, D. Villeneuve, , D. J., and P. Hansen. A proximal trust region for column generation stabilization. *Discrete Mathematics*, 194:229–237, 1999.

[27] C. W. Duin and A. Volgenant. Reduction tests for the Steiner problem in graphs. *Networks*, 19(5):549–567, 1989.

[28] I. Dumitrescu and N. Boland. Improved preprocessing, labeling and scaling algorithms for the weight constrained shortest path problem. *Networks*, (42):135–153, 2004.

[29] L. R. Ford and F. D. R. *Flows in Networks*. Princeton University Press, 1962.

[30] M. Garcia. Arvores com restricoes de diametro. Master's thesis, University of Lisbon, 1994.

[31] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences. W.H. Freeman & Company, 1979.

[32] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859, 1961.

[33] F. W. Glover and G. A. Kochenberger. *Handbook of Metaheuristics*, volume 114 of *International Series in Operations Research & Management Science*. Springer, 2003.

[34] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, (24):296–317, 1995.

[35] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989.

[36] R. E. Gomory. Outline of an alogrithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.

[37] L. Gouveia. Using the Miller-Tucker-Zemlin constraints to formulate a minimal spanning tree problem with hop constraints. *Computers & Operations Research*, 22(9):959–970, 1995.

[38] L. Gouveia. Multicommdity flow models for spanning trees with hop constraints. *European Journal of Operational Research*, 95(1):178–190, 1996.

[39] L. Gouveia. Using variable redefinition for computing lower bounds for minimum spanning and Steiner trees with hop constraints. *INFORMS Journal on Computing*, 10(2):180–188, 1998.

[40] L. Gouveia. Using hop-indexed models for constrained spanning and Steiner tree models. In B. Sanso and P. Soriano, editors, *Telecommunications network planning*, pages 21–32. Kluwer, 1999.

[41] L. Gouveia, A. Paias, and D. Sharma. Modeling and solving the rooted distance-constrained minimum spanning tree problem. *Computers and Operations Research*, 35:600–613, 2008.

[42] L. Gouveia and C. Requejo. A new Lagrangean relaxation approach for the hop-constrained minimum spanning tree problem. *European Journal of Operational Research*, 132(3):539–552, 2001.

[43] L. Gouveia, L. Simonetti, and E. Uchoa. Modelling the hop-constrained minimum spanning tree problem over a layered graph. In *International Network Optimization Conference (INOC) 2007*, pages 1–6, Spa, Belgium.

[44] L. Gouveia, L. Simonetti, and E. Uchoa. Modeling hop-constrained and diameter-constrained minimum spanning tree problems as Steiner tree problems over layered graphs. *Mathematical Programming*, pages 1–26, 2010.

[45] S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. In G. Desaulniers, J. Desrosiers, and M. Solomon, editors, *Column Generation*, pages 33–67. Springer, 2005.

[46] D. Johnson, M. Minkoff, and S. Phillips. The prize collecting Steiner tree problem: Theory and practice. In *SODA ́00 Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms* , pages 760–769, 2000.

[47] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[48] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[49] L. G. Khachian. A polynomial algorithm for linear programming. *Dokolady Academica Nauk USSR*, 244:1093–1096, 1979.

[50] G. W. Klau, I. Ljubic, A. Moser, P. Mutzel, P. Neuner, U. Pferschy, G. Raidl, and R. Weiskircher. Combining a memetic algorithm with integer programming to solve the prize-collecting Steiner tree problem. *Springer Lecture Notes in Computer Science*, (3102):1304–1315, 2004.

[51] P. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted Steiner trees. *Journal of Algorithms*, (19):104–115, 1995.

[52] T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32:207–232, 1998.

[53] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1,2):83–97, 1955.

[54] A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[55] M. Leitner and G. R. Raidl. Strong lower bounds for a survivable network design problem. *Electronic Notes in Discrete Mathematics*, 36:295–306, 2010.

[56] M. Leitner, G. R. Raidl, and U. Pferschy. Accelerating column generation for a survivable network design problem. In *Proceedings of the International Network Optimization Conference 2009*, Pisa, Italy, 2009.

[57] M. Leitner, M. Ruthmair, and G. R. Raidl. Stabilized branch-and-price for the rooted delay-constrained Steiner tree problem. In J. Pahl, T. Reiners, and S. Voß, editors, *Network Optimization: 5th International Conference, INOC 2011*, volume 6701 of *LNCS*, pages 124–138, Hamburg, Germany, 2011. Springer.

[58] M. Leitner, M. Ruthmair, and G. R. Raidl. Stabilized column generation for the rooted delay-constrained Steiner tree problem. In *Proceedings of the VII ALIO/EURO – Workshop on Applied Combinatorial Optimization*, pages 250–253, Porto, Portugal, 2011.

[59] I. Ljubic, R. Weiskircher, U. Pferschey, P. Mutzel, G. W. Klau, and M. Fischetti. An algorithmic framework for the exact solution of the prize-collecting Steiner tree problem. *Mathematical Programming*, 105(2-3):427–449, 2006.

[60] A. Lucena and M. G. C. Resende. Strong lower bounds for the prize collecting Steiner problem in graphs. *Discrete Applied Mathematics*, 141:227–294, 2004.

[61] M. E. Luebbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

[62] T. L. Magnanti and W. L. Optimal trees. *Handbooks in Operations Research and Mangement Science*, 7:530–615, 1995.

[63] R. E. Marsten, W. W. Hogan, and J. Blankenship. The BOXSTEP method for large-scale optimization. *Operations Research*, 23(3):389–405, 1975.

[64] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24:1097–1100, 1997.

[65] P. J. Neame. *Nonsmooth dual methods in integer programming*. PhD thesis, University of Melbourne, 1999.

[66] M. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6:1–7, 1987.

[67] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991.

[68] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization : Algorithms and Complexity*. Dover Publications, 1998.

[69] L. L. Pinto and G. Laporte. An efficient algorithm for the Steiner tree problem with revenue, bottleneck and hop objective function. *European Journal of Operational Research*, 207:45–49, 2010.

[70] T. Polzin and S. V. Daneshmand. A comparison of Steiner tree relaxations. *Discrete Applied Mathematics*, (112):241–261, 2001.

[71] L. M. Rousseau, M. Gendreau, and D. Feillet. Interior point stabilization for column generation. *Operations Research Letters*, 35:660–668, 2007.

[72] A. Segev. The node-weighted Steiner tree problem. *Networks*, 27:169–190, 1987.

[73] E. Uchoa. Reduction tests for the prize-collecting Steiner problem. *Operations Research Letters*, 34(4):437–444, 2006.

[74] L. Vandenberghe and S. Boyd. *Convex Optimization*. Cambridge University Press, 2004.

[75] F. Vanderbeck. Implementing mixed integer column generation. In G. Desaulniers, J. Desrosiers, and M. Solomon, editors, *Column Generation*, pages 331–358. Springer, 2005.

[76] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[77] S. Voß. The Steiner tree problem with hop constraints. *Annals of Operations Research*, 86:321–345, 1999.

[78] P. Wengtes. Weighted Dantzig-Wolfe decomposition for mixed-integer programming. *International Transactions in Operational Research*, 4(2):151–162, 1997.

[79] L. A. Wolsey. *Integer Programming*. Wiley, 1998.