Exploring Advanced Techniques in Mixed-Integer Linear Programming: A Study of the Branch-and-Cut Framework

Bachelor Thesis

by
Thanh-Van Huynh

at the

Universität Konstanz

Written and submitted at the
Department of Mathematics and Statistics
Faculty of Sciences

Supervisor
Prof. Dr. Stefan Volkwein

Konstanz, 2023

Thanh-Van Huynh: Bachelor Thesis
in Mathematics.

Acknowledgements

I would like to thank my supervisor Prof. Dr. Stefan Volkwein for all his help and advice, and also for giving me a lot of freedom and flexibility with my thesis.

This endeavor would not have been possible without Carl Eggen and Moritz Link, who generously provided knowledge and expertise – I highly appreciated their open door policy and that our meetings could include immensly helpful professional advice as well as having fun together with gamified internet applications about my topic. Their enthusiasm really sparked my own excitement about my thesis and I cannot imagine better mentors.

I would like to express my deepest gratitude to my friends, especially Erik, Loïse, and the Bib-Crew. Thesis writing period is inevitably linked to spending the day in the library together, taking a swim in the lake before, during, and after, and enjoying summer evenings (and nights) in Konstanz (and other cities). We were kind of having an amazing time despite the academic stress, which is an impressive achievement and only speaks to their character and how much I appreciate these extraordinary people.

Lastly, I am extremely grateful to my familiy, especially my parents, for their unwavering love and support.

Thanh-Van Huynh
Konstanz, 2023

Zusammenfassung

Diese Arbeit behandelt die theoretischen Grundlagen des Branch-and-Cut Frameworks, das ein Standardwerkzeug zum Lösen gemischt-ganzzahliger linearer Optimierungsprobleme ist.

Als Kombination von Branch-and-Bound und Schnittebenenverfahren enthält Branch-and-Cut wesentliche Bestandteile beider Strategien. Die Idee, welche dem Branch-and-Bound Algorithmus zugrunde liegt, ist das Teile-und-Herrsche-Verfahren, auch *divide-and-conquer* genannt. Bei der iterativen Teilung des Problems in kleinere Subprobleme und dem Lösen von Relaxierungen dieser Subprobleme, welche keine Ganzzahligkeitsbedingungen mehr enthalten, stellen sich drei Fragen: Wie wird der Lösungsraum partitioniert, um neue Subprobleme zu generieren? In welcher Reihenfolge werden diese gelöst? Wie verhindern wir, dass suboptimale Bereiche des Lösungsraums überhaupt erst näher untersucht werden? Zur Beantwortung der ersten beiden Fragen geben wir einen Überblick über Verzweigungsregeln und Suchverfahren.

Der Fokus liegt auf den Schnittebenenverfahren, welche eine Lösung für die drittgenannte Frage darstellen. Die Idee ist es, Ungleichungen einzuführen, die für den Lösungsraum des ursprünglichen Problems gelten, aber nicht für die aktuelle Lösung des relaxierten Subproblems – dies führt zu einer schrittweisen Verschärfung der Relaxierung bis (im Idealfall) eine ganzzahlige Lösung gefunden wird. Wir stellen die *Mixed-Integer Rounding Cuts* sowie die *Gomory Mixed-Integer Cuts* vor und beweisen sie. Diese beiden Schnitte sind äquivalent, was wir ebenfalls zeigen werden. Zum Schluss demonstrieren wir diese Äquivalenz und visualisieren die Schnitte anhand eines Beispiels.

Abstract

In this thesis, we will study the theoretical foundations of the branch-and-cut framework, which is a state-of-the-art paradigm for solving linear optimization problems involving integer variables.

As a hybrid of the branch-and-bound algorithm and the cutting planes method, branch-and-cut contains key ingredients of both strategies. The underlying idea of branch-and-bound is divide-and-conquer. When iteratively dividing into smaller subproblems and solving these subproblems' relaxations, which do not contain any integrality conditions anymore, three questions arise: How is the solution space partioned to produce new subproblems, in which order does the algorithm explore the subproblems, and how do we prevent exploration of suboptimal regions of the solution space? We will give an overview over various branching rules and node selection methods, which address the first two questions.

The focus will be on the plane separation, which answers the last question: The idea is to introduce inequalities, which are valid for the solution space of the root problem, but not for the current solution of the relaxed subproblem, tightening the relaxation iteratively until, ideally, an integer solution is found. We will introduce and prove both the mixed-integer rounding cut and the Gomory mixed-integer cut. These two cuts are equivalent, for which we will provide a proof as well. Lastly, we will demonstrate this equivalence and visualize the cuts in an example.

Contents

List of Figures

1 Introduction

In an era characterized by an unprecedented volume of data and complex decision-making scenarios, optimization techniques play a pivotal role in addressing a wide spectrum of real-world challenges. The warehouse location problem or the traveling salesman problem are just two of the most commonly known examples that can be mathematically modeled using mixed-integer linear programs. The branch-and-cut framework has proved to be a successful tool in solving these complex combinatorial problems involving integer variables. By blending techniques from linear programming, cutting plane methods, and branch-and-bound frameworks, the branch-and-cut paradigm provides a systematic and efficient means to tackle linear optimization problems that encompass both continuous and discrete variables.

The primary objective of this thesis is to provide a comprehensive exploration of the branch-and-cut framework's theoretical foundations.

Chapter 2 introduces the mixed-integer linear programming (MILP) problem and also the resulting linear programming (LP) problem that arises when the integrality conditions are removed. Chapter 3 gives commonly known examples of MILP problems and how to model them mathematically. Chapter 4 presents two methods commonly used to solve MILP problem: branch-and-bound and cutting planes. In Chapter 5 we investigate the key ingredients of the branch-and-cut framwork, which is a hybrid of these two strategies, and discuss a number of different approaches and algorithms for each component – namely, various branching rules, node selection choices, and cutting plane options. Chapter 6 concludes this thesis and suggests a few ideas how this topic could be explored further.

2 Mixed-Integer Linear Programming

This section introduces the mixed-integer linear programming (MILP) problem and the linear programming (LP) problem resulting from relaxing the MILP problem by removing the integrality conditions.

The notation and also the structure of this chapter are based on [Wol98; Ach07, Section 1.3].

A mixed-integer linear program (MILP) is defined as follows.

Definition 2.1. Given matrices $A \in \mathbb{R}^{m \times n}$, $G \in \mathbb{R}^{m \times p}$, vectors $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^{n+p}$, the *mixed-integer linear program* $\text{MILP} = (A, G, b, c)$ is to solve

$$c^* = \min_{z \in \mathcal{F}_{\text{MILP}}} c^\top z \tag{MILP}$$

on the *feasible set*

$$\mathcal{F}_{\text{MILP}} := \{(x, y) \in \mathbb{Z}^n \times \mathbb{R}^p : \; Ax + Gy \leq b\}.$$

The vectors in the set $\mathcal{F}_{\text{MILP}}$ are called *feasible solutions*. A feasible solution $z^* \in \mathcal{F}_{\text{MILP}}$ is called *optimal* if its objective value satisfies $c^\top z^* = c^*$.

Remark 2.2. There are two common special cases of mixed-integer linear programs: *integer programs* (IP) and *binary programs* (BP). IPs are MILPs with $p = 0$, hence the feasible set is

$$\mathcal{F}_{\text{IP}} := \{x \in \mathbb{Z}^n : \; Ax \leq b\}.$$

BPs are MILPs with $p = 0$ and only allow 0 or 1 as values for the integer variables, hence the feasible set is

$$\mathcal{F}_{\text{BP}} := \{0, 1\}^n.$$

MILP is $\mathcal{NP}$-hard [Kar72] as the traveling salesman problem, which can be formulated as an integer programming problem (see Section 3.2), is $\mathcal{NP}$-complete[1]. Nevertheless, linear programs (LPs) are solvable in polynomial time [Kha79], which is exploited when it comes to also solving MILP problems.

In mixed-integer linear programming we are restricted to

- linear constraints,
- a linear objective function, and
- integer or real-valued domains.

Despite these restrictions in modeling, practical applications prove that MILP, IP, and BP can effectively address numerous real-world challenges. However, successfully applying the MILP framework to real-world problems often demands expertise to formulate models compatible with existing MILP solvers.

Most modern MILP solvers recursively split the problem into smaller subproblems, thereby generating a so-called branching tree (see Section 4.1). Nevertheless, the approach to handling the subproblems visualized by nodes varies. For each node of the tree, the

[1] See Appendix A.1 for an introduction to complexity theory.

LP relaxation is solved, which can be derived from the MILP by removing the integrality conditions[2].

Definition 2.3. Given a mixed-integer linear program MILP $= (A, G, b, c)$, its *LP relaxation* is defined as

$$\check{c} = \min_{z \in \mathcal{F}_{\text{LP}}} c^\top z. \tag{LP}$$

The set of *feasible solutions* of the LP relaxation is

$$\mathcal{F}_{\text{LP}} := \left\{(x, y) \in \mathbb{R}^{n+p} : \; Ax + Gy \leq b\right\}.$$

An LP-feasible solution $\check{z} \in \mathcal{F}_{\text{LP}}$ is called *LP-optimal* if $c^\top \check{z} = \check{c}$.

Remark 2.4. The solution set of the LP relaxation defines a polyhedron $P = \mathcal{F}_{\text{LP}}$. This *LP polyhedron* is a superset of its integer hull $P_I \subseteq P$, which is the convex hull

$$P_I = \text{conv}\{P \cap (\mathbb{Z}^n \times \mathbb{R}^p)\} = \text{conv}\{\mathcal{F}_{\text{MILP}}\}$$

of the MILP feasible solutions.

Strengthening the LP relaxation is a critical strategy in MILP to enhance the performance of our solving algorithm. A stronger LP relaxation often results in tighter bounds and can lead to more effective branch-and-bound procedures. There are several techniques to strengthen the LP relaxation, e.g., tightening variable bounds, incorporating feasibility heuristics, or introducing valid inequalities, which is called cutting planes. This strategy uses the LP information and the integrality restrictions to derive inequalities that are valid for the MILP problem but may not be satisfied by the LP relaxation, therefore cutting off the solution of the current LP relaxation without removing integral solutions (see Section 4.2). The objective value $\check{c}$ of the LP relaxation provides a lower bound for the whole subtree, and if this bound is not smaller than the value $\hat{c} = c^\top \hat{z}$ of the current best primal solution $\hat{z}$, the node and its subtree can be discarded.

To conclude this section, we define the floor and ceiling function which we will need later on.

Definition 2.5. Given a real number $x \in \mathbb{R}$, the floor function $\lfloor x \rfloor$ and the ceiling function $\lceil x \rceil$ are defined by the following equations, respectively,

$$\begin{aligned} \lfloor x \rfloor &:= \max\{z \in \mathbb{Z} : \; z \leq x\}, \\ \lceil x \rceil &:= \min\{z \in \mathbb{Z} : \; z \geq x\}. \end{aligned}$$

In other words, the floor function $\lfloor x \rfloor$ rounds x down to the closest integer and, analogously, the ceiling function $\lceil x \rceil$ rounds x up to the closest integer. For integers $x \in \mathbb{Z}$ the equation

$$\lfloor x \rfloor = \lceil x \rceil = x$$

holds.

[2]See Appendix A.2 for a brief sketch of the idea behind the simplex algorithm, which is used for solving LPs.

3 Motivational Examples

In this section, we present two commonly known examples of mixed-integer linear programming problems and how to model them mathematically, i.e., we derive the cost function under consideration of variables, parameters, and constraints.

3.1 Warehouse Location Problem

This section is based on the notation of [Füh11, Section 3.2].

The warehouse location problem (WLP) is an integer problem and determines the optimal number and corresponding best locations of warehouses to be placed based on geographical demands, facility costs, and transportation distances. Assuming that there is no maximum capacity, i.e., the problem is uncapacitated, the WLP can be described as follows:

$$\begin{aligned} c_{ij} &= \text{parameter: transportation costs from warehouse } i \text{ to customer } j \\ f_i &= \text{parameter: fix costs to open warehouse } i \\ x_{ij} &= \text{binary decision variable: customer } j \text{ is being serviced from warehouse } i \\ y_i &= \text{binary decision variable: warehouse } i \text{ is being opened} \end{aligned}$$

Given a set I of warehouse locations and a set J of customers, we obtain the cost function

$$F(x,y) = \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \sum_{i \in I} f_i y_i \tag{3.1}$$

with constraints

$$\sum_{i \in I} x_{ij} = 1 \qquad \text{for all } j \in J, \tag{3.2}$$

$$x_{ij} \leq y_i \qquad \text{for all } i \in I, j \in J, \tag{3.3}$$

$$x_{ij}, y_i \in \{0,1\} \qquad \text{for all } i \in I, j \in J. \tag{3.4}$$

The objective function (3.1) of the WLP sums up the fixed costs f_i, which arise as a result of a warehouse being opened, with the variable costs c_{ij}, which are incurred by a transport from i to customer j, and aims to minimize them. The constraints of the WLP ensure that each customer's demand is fully satisfied by exactly one warehouse (3.2) as well as that a customer can only be supplied from a location i, if the establishment of a depot is also planned there (3.3).

Warehouse location problems are utilized in many industries to find the optimal placement of various facilities, including power plants, public transportation terminals, polling locations, and cell towers, to maximize efficiency, impact, and profit. In more unique applications, extensive research has been done in applying WLPs to humanitarian efforts, such as identifying disaster management sites to maximize accessibility to healthcare and treatment [ASS17].

Make it Mixed: Adding Continuous Decision Variables

We can relax constraint (3.2), which states that each customer's demands is satisfied by exactly one warehouse, and allow for multiple warehouses to service the same customer. Hence, we can add a continuous variable,

z_{ij} = continuous decision variable: fraction of customer j's demand serviced by warehouse i

This leads to a modification of the cost function

$$F^*(x,y) = \sum_{i\in I}\sum_{j\in J} c_{ij}z_{ij} + \sum_{i\in I} f_i y_i \tag{3.1*}$$

and to the modified/additional constraints

$$\sum_{i\in I} z_{ij} = 1 \qquad \text{for all } j \in J, \tag{3.2*}$$

$$z_{ij} \leq x_{ij} \qquad \text{for all } i \in I, j \in J. \tag{3.5}$$

These constraints ensure that each customer's demand is fully satisfied (3.2*) and that a fraction of a customer's demand can only be satisfied by a warehouse if this customer is also serviced by this warehouse (3.5), hence, this is some kind of consistency constraint.

3.2 Traveling Salesman Problem

This section is based on [Law+85, Chapter 2] and [KD15].

The traveling salesman problem (TSP) is an integer problem and determines the minimal total tour duration of a salesman who starts his tour in one city, visits $n-1$ other cities exactly once, and the returns home, i.e., we consider a circuit tour with n cities. The TSP can be described as follows:

t_{ij} = parameter: travel time from city i and city j
u_i = parameter: arbitrary real numbers
x_{ij} = binary decision variable: salesman travels from city i to city j

We obtain the cost function

$$F(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} t_{ij}x_{ij} \tag{3.6}$$

with constraints

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \text{for all } j \in \{1,\dots,n\}, \tag{3.7}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \text{for all } i \in \{1,\dots,n\}, \tag{3.8}$$

$$u_i - u_j - nx_{ij} \leq n-1 \qquad \text{for all } i,j \in \{2,\dots,n\}. \tag{3.9}$$

The objective function (3.6) calculates the duration of the total tour by summing up the separate tour durations from one city to another. The constraint (3.7) insures that the

salesman enters each city once from one of the other cities. Similarly, constraint (3.8) insures that the salesman leaves each city once for one of the other cities. Constraint (3.9) restricts the problem by prohibiting subtours, e.g., two circuit tours which cover all cities in total: Arbitrarily designate city 1 to be the home base. Then the last constraint blocks all tours not containing city 1. In conjunction with (3.7) and (3.8), all subtours are blocked.

The traveling salesman problem is one of the most intensely studied problems in combinatorial optimization and has several applications, such as planning, logistics, and manufacturing. Slightly modified, it appears to be a subproblem in many areas, such as DNA sequencing [Lee+04] or astronomy, as astronomers observing many sources will want to minimize the time spent moving the telescope between the sources. In such problems, the TSP can be embedded inside an optimal control problem [RPK20].

Make it Mixed: Adding Continuous Variables

A variation of the TSP, the Traveling Salesman Problem with Time Windows (TSPTW), is typically considered as a mixed-integer problem. In TSPTW, the traveling salesman needs to visit the set of cities within specified time windows. Hence, we modify the problem and model the time windows using two additional parameters and a continuous decision variable,

$$\begin{aligned} a_i &= \text{parameter: earliest time the salesman can arrive at city } i \\ b_i &= \text{parameter: latest time the salesman can arrive at city } i \\ t_i &= \text{continuous decision variable: time passed until visiting city } i \end{aligned}$$

This leads to the additional constraints

$$\begin{aligned} t_i - t_j &\geq t_{ij} - Mx_{ij} && \text{for all } i,j \in \{1,\dots,n\}, && (3.10)\\ t_j - t_i &\geq t_{ij} - M(1-x_{ij}) && \text{for all } i,j \in \{1,\dots,n\}, && (3.11)\\ t_i - t_1 &\geq t_{1i} && \text{for all } i \in \{2,\dots,n\}, && (3.12)\\ t_n - t_i &\geq t_{i1} && \text{for all } i \in \{2,\dots,n\}, && (3.13)\\ a_i \leq t_i &\leq b_i && \text{for all } i \in \{1,\dots,n\}, && (3.14)\\ t_{ij} &\geq 0 && \text{for all } i,j \in \{1,\dots,n\}. && (3.15) \end{aligned}$$

$M > 0$ from constraints (3.10) and (3.11) is a sufficiently large positive constant that ensures that the difference between the earliest arrival time of two cities is larger than the travel time between them, i.e., these constraints are the linearization of the inequality

$$|t_i - t_j| \geq t_{ij} \quad \text{for all } i \in \{3,4,\dots,n\},\ j \in \{2,\dots,i\},$$

as proven in [KD15, Proposition 1]. Similarly, constraints (3.12) and (3.13) ensure that the arrival time t_i at an arbitrary city i aligns with the arrival time at the first and last city, and also the travel time in between. Constraint (3.14) ensures that the salesman visits a city in the corresponding time window.

4 Algorithms

This section presents two important algorithms that are often employed when solving mixed-integer linear programs. This problem class is commonly solved by *branch-and-bound*, which is explained in Section 4.1. State-of-the-art MILP solvers heavily rely on the linear programming (LP) relaxation to guide the decision how to divide the given problem instance into smaller subproblems and calculate lower bounds for these subproblems. The LP relaxation can be tightened by *cutting planes* to improve the lower bounds, see Section 4.2.

This chapter is based on [Ach07, Chapter 2].

4.1 Branch-and-Bound

The branch-and-bound procedure is a very general and widely used method to solve (mixed-)integer optimization problems, also known as *implicit enumeration, divide-and-conquer, backtracking*, or *decomposition*. It systematically explores the search space of a problem by dividing it into smaller subproblems until the individual subproblems are easy to solve. The best of the subproblems' solutions is the global solution. Algorithm 1 summarizes this procedure.

The iterative splitting of a subproblem into two or smaller subproblems in Step 7 is

Algorithm 1: Branch-and-bound

Input : Minimization problem instance R.
Output: Optimal solution z^* with value c^*, or conclusion that R has no solution, indicated by $c^* = \infty$.

1. Initialize $\mathcal{L} := \{R\}$, $\hat{c} := \infty$. `# init`
2. If $\mathcal{L} = \emptyset$, then return $z^* = \hat{z}$ and $c^* = \hat{c}$. `# abort`
3. Choose $Q \in \mathcal{L}$, and set $\mathcal{L} := \mathcal{L} \setminus Q$. `# select`
4. Solve a relaxation Q_{relax} of Q. If Q_{relax} is empty, set $\check{c} := \infty$. Otherwise, let $\check{z}$ be an optimal solution to Q_{relax} and $\check{c}$ its objective value. `# solve`
5. If $\check{c} \geq \hat{c}$, go to step 2. `# bound`
6. If $\check{z}$ is feasible R, set $\hat{z} := \check{z}$, $\hat{c} := \check{c}$, and go to step 2. `# check`
7. Split Q into subproblems $Q = Q_1 \cup \ldots \cup Q_k$, set $\mathcal{L} := \mathcal{L} \cup \{Q_1 \cup \ldots \cup Q_k\}$, and go to step 2. `# branch`

called *branching* and creates a *branching tree* where each node corresponds to a specific subproblem (see Figure 4.1). The root of the tree represents the original problem R, and its children represent the subproblems obtained by dividing the original problem into smaller parts – the children are either leaves which are "easy" to solve, or subproblems in $\mathcal{L}$ that still have to be processed and potentially result in more subproblems via branching.

The purpose of the *bounding* in Step 5 is to prevent exhaustive examination of all potential solutions if R, which are usually exponentially many. In order for bounding to be effective, we need good lower (dual) bounds $\check{c}$ and upper (primal) bounds $\hat{c}$. Lower bounds are derived by solving a relaxation Q_{relax} which should be computationally inexpensive. Upper bounds can be found during the branch-and-bound process outlined in Step 6, or alternatively formulated using primal heuristics.

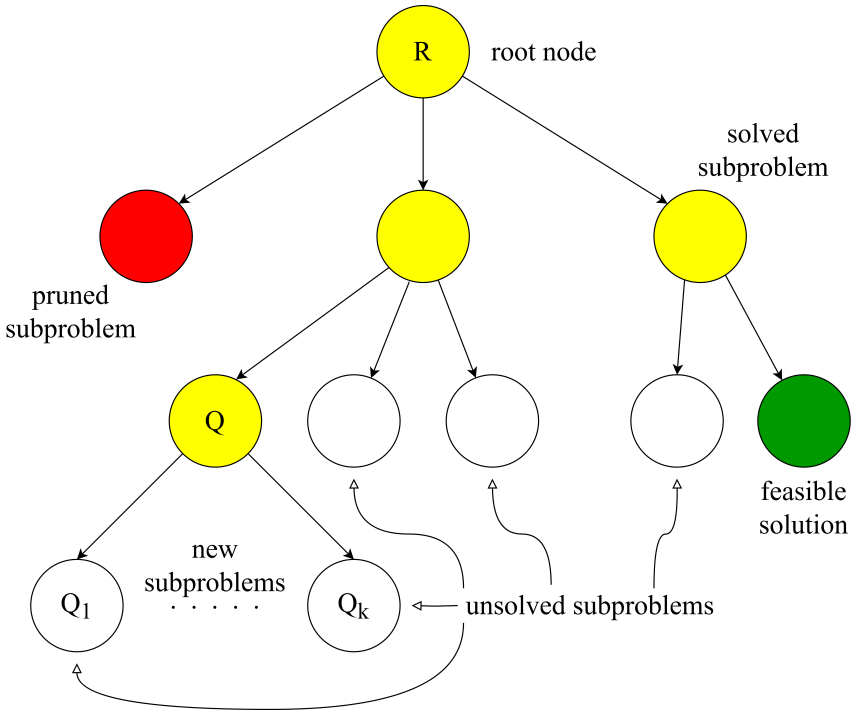


Figure 4.1: Branch-and-bound search tree. Based on: [Ach07, Figure 2.1].

The node selection detailed in Step 3 and the branching strategy described in Step 7 constitute pivotal choices within the branch-and-bound algorithm. These selections must be tailored to the specific problem class at hand. They profoundly impact the early identification of optimal primal solutions in Step 6 and the rate at which the lower bounds of the open subproblems in $\mathcal{L}$ increase. They also influence the bounding in Step 5, which should cut off subproblems as early as possible and thereby prune large parts of the search tree. Even more important for a branch-and-bound algorithm to be effective is the type of relaxation that is solved in Step 4. The chosen relaxation must reconcile two typically contradictory requirements: it should be easy to solve while also yielding strong dual bounds.

In mixed-integer programming, the most commonly used relaxation is the LP relaxation (see Definition 2.3), which proved to be very successful in practice. Besides supplying a dual bound that can be exploited for the bounding in Step 5, the LP relaxation can also be used to guide the branching decision in Step 7. The most popular branching strategy in MILP solving is to split the domain of an integer variable $x_j \in \mathbb{Z}$, with fractional LP value $\check{x}_j \notin \mathbb{Z}$ into two parts, thus creating the two subproblems $Q_1 = Q \cap \{x_j \leq \lfloor \check{x}_j \rfloor\}$ and $Q_2 = Q \cap \{x_j \geq \lceil \check{x}_j \rceil\}$ (see Figure 4.2). Methods to select a fractional variable as branching variable are discussed in Section 5.1.

To summarize the main features and calculation steps of the branch-and-bound framework in other words: this procedure implicitly enumerates all possible solutions to the problem by storing partial solutions called subproblems in a tree structure. Unexplored nodes in the tree generate children by partitioning the solution space into smaller regions that can be solved recursively (i.e., branching), and rules are used to prune off regions of the search space that are provably suboptimal (i.e., bounding). Once the entire tree has

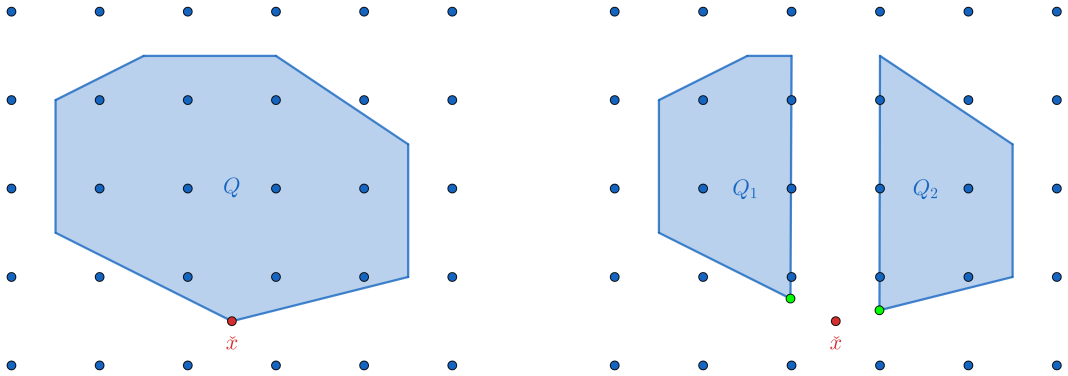


Figure 4.2: LP based branching on a single fractional variable. Based on: [Ach07, Figure 2.2].

been explored, the best solution found in the search is returned.

However, in the above framework there are three components which are left unspecified, but which can have significant impacts on the performance of the algorithm. These components are the branching strategy (i.e., how the solution space is partitioned to produce new subproblems in the tree, see Section 5.1), the search strategy (i.e., the order in which subproblems in the tree are explored, see Section 5.2), and the pruning rules (i.e., rules that prevent exploration of suboptimal regions of the tree, see Section 5.3).

4.2 Cutting Planes

Besides splitting the current subproblem Q into two or more easier subproblems by branching, one can also try to tighten the subproblem's relaxation in order to rule out the current fractional solution $\check{x}$ and to obtain a different one that may be integral or guides us closer to an integral one.

The LP relaxation can be tightened by introducing additional linear inequalities $a^\top x \leq b$ that are violated by the current LP solution $\check{x}$ but do not cut off feasible solutions from the original MILP problem (see Figure 4.3). Thus, the current solution $\check{x}$ is *separated* from the convex hull of integer solutions Q_I by the *cutting plane* $a^\top x \leq b$, i.e.,

$$\check{x} \notin \{x \in \mathbb{R}^n : a^\top x \leq b\} \supseteq Q_I.$$

To benefit from the stronger relaxations obtained by cutting planes without hampering the solvability of the LP relaxations, today's most successful MILP solvers combine branching and cutting plane separation in one of the following fashions:

- **Cut-and-branch:** The LP relaxation R_{LP} of the initial (root) problem R is strengthened by cutting planes as long as it seems to be reasonable and does not reduce numerical stability too much. Afterwards, the problem is solved with branch-and-bound.

- **Branch-and-cut:** The problem is solved with branch-and-bound, but the LP relaxation Q_{LP} of *all* subproblems Q (including the initial problem R) might be strengthened by cutting planes. Here one has to distinguish between globally valid cuts and

cuts that are only valid in a local part of the branch-and-bound search tree, i.e., cuts can be used for all the subproblems during the course of the algorithm, but local cuts have to be removed from the LP relaxation after the search leaves the subtree for which they are valid.

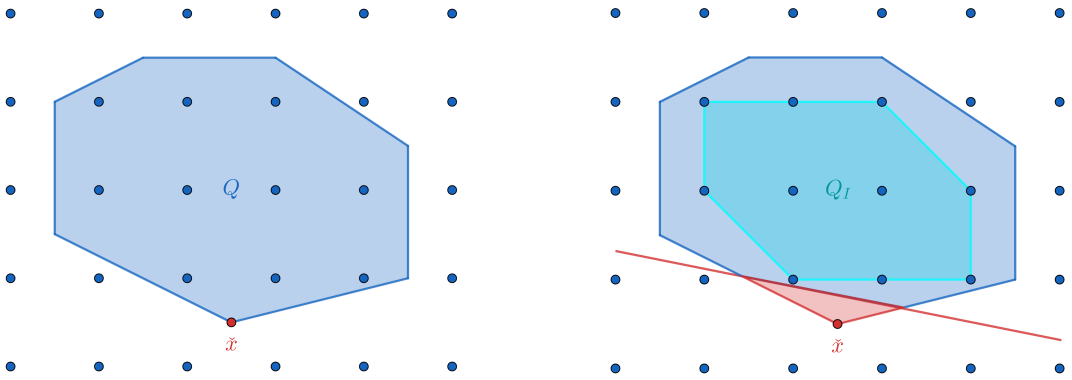


Figure 4.3: A cutting plane that separates the fractional LP solution $\check{x}$ from the convex hull Q_I of integer points of Q, which is the feasible set of the LP relaxation. Based on: [Ach07, Figure 2.3].

5 Branch-and-Cut Framework

The branch-and-cut paradigm is a hybrid of the branch-and-bound and cutting plane methods. It is central to a wide range of modern global optimization approaches, particularly mixed-integer linear and nonlinear programming solvers. Cutting planes, or cuts, tighten the relaxation of a given optimization problem and are experimentally known to significantly improve a branch-and-bound process. Most recently, Basu et al. [Bas+22] showed that using branch-and-cut can outperform either branching or cutting alone (sometimes by orders of magnitude).

In this section, we investigate the main components of the branch-and-cut framework and discuss a number of different approaches and algorithms for each step. Section 5.1 evaluates commonly used branching rules with pseudocost branching and strong branching being of particular interest. Section 5.2 presents and compares depth first search vs. best first search to select the next subproblem from the search tree to be processed. Section 5.3 introduces and proves two valid inequalities, namely the mixed-integer rounding cut and the Gomory mixed-integer cut, and concludes with an example.

For the sake of completeness, branching rules and node selection methods are presented, but the focus is on the cutting planes.

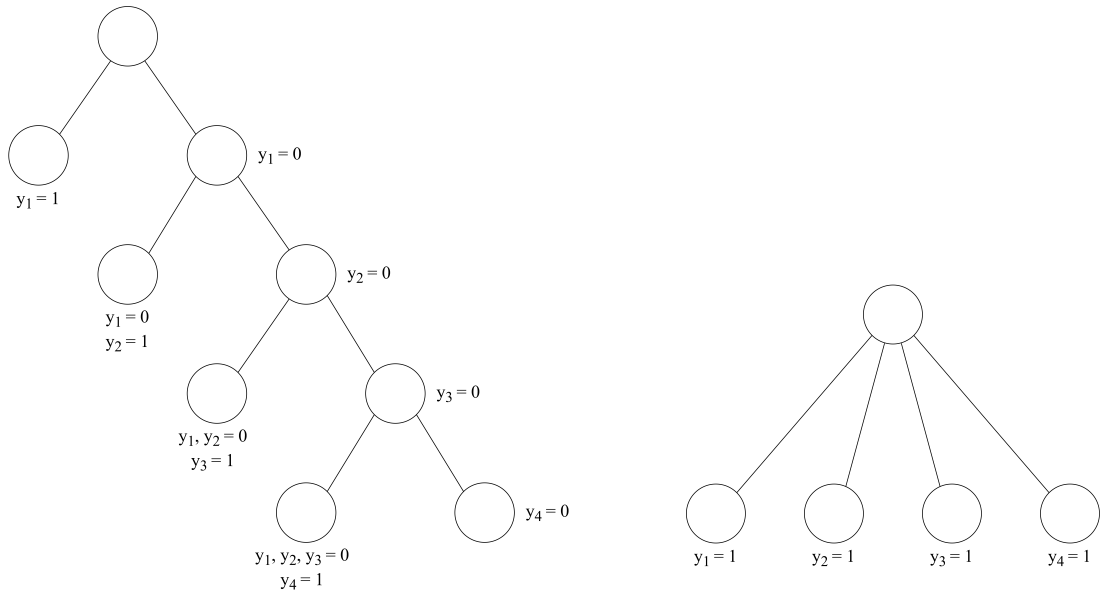
This chapter is based on parts of [Ach07, Chapter 5, 6, 8] as well as [Mor+16].

5.1 Branching

Since branching is one of the key ingredients of any branch-and-bound algorithm, finding good strategies is important to practical MILP solving. The choice of branching strategy determines how children are generated from a subproblem. Branching strategies can be categorized into two groups: binary branching strategies and non-binary (or wide) branching strategies. *Binary branching* strategies focus on subdividing a subproblem S into two mutually-exclusive, smaller subproblems. *Wide branching* on the other hand focuses on selecting one element from a set of different options and allows for potentially large reductions in the size of the search tree (in terms of depth). Figure 5.1 shows an example of how the two categories differ from each other.

The only way to split a problem Q within an LP based branch-and-bound algorithm is to branch on linear inequalities in order to keep the property of having an LP relaxation at hand, which falls into the category of binary branching. The easiest and most common inequalities are *trivial inequalities*, which are inequalities that split the feasible interval of a singleton variable, as seen in Figure 4.2. To provide more precise detail, if $x_j \in \mathbb{Z}$ is some integer variable with fractional value $\check{x} \notin \mathbb{Z}$ in the current optimal LP solution, we obtain two subproblems: one by adding the trivial inequality $x_j \leq \lfloor \check{x} \rfloor$ (called the *left subproblem* or *left child*, denoted by Q_j^-) and one by adding the trivial inequality $x_j \geq \lceil \check{x} \rceil$ (called the *right subproblem* or *right child*, denoted by Q_j^+). This technique of branching on trivial inequalities is also called *branching on variables*, since it only requires to change the bounds of variable x_j. Note that branching on more complicated inequalities or even splitting the problem into more than two subproblems, i.e., wide branching, is rarely incorporated into general MILP solvers. Algorithm 2 shows the procedure for generic variable selection.

In the following we focus on the most common variable selection rules, which are all



(a) Binary branching. (b) Wide branching

Figure 5.1: Binary branching versus wide branching. Given a set of 4 elements from which one must be selected, binary branching explicitly rejects elements 1, 2, and 3 before creating a branch that considers element 4. Conversely, wide branching can consider each of them immediately. Based on: [Mor+16, Figure 4].

variants of Algorithm 2. The difference is how the score in Step 2 is computed.

The ultimate goal is to find a computationally efficient branching strategy that mini-

Algorithm 2: Generic variable selection

Input : Current subproblem Q with an optimal LP solution $\check{z} = (\check{x}, \check{y}) \notin \mathcal{F}_{\text{MILP}}$.
Output: An integer variable x_j with fractional LP value $\check{x}_j \notin \mathbb{Z}$.
1. Let $F = \{j \in \{1, \dots, n\} : \check{x}_j \notin \mathbb{Z}\}$ be the set of branching candidates.
2. For all candidates $j \in F$, calculate a score value $s_j \in \mathbb{R}$.
3. Return an index $j \in F$ with $s_j = \max_{k \in F}\{s_k\}$.

mizes the number of branch-and-bound nodes that need to be evaluated. Since no global approach is known, one tries to find a branching variable that is at least a good choice for the current branching. One way to measure the quality of a branching is to look at the changes in the objective function of the LP relaxations of the two children Q_j^- and Q_j^+ compared to the relaxation of the parent node Q. There are many possibilities to compute these changes, however, we follow the traditional way of trying to improve the dual bound.

In order to compare branching candidates, for each candidate the two objective function changes $\Delta_j^- := \check{c}_{Q_j^-} - \check{c}_Q$ and $\Delta_j^+ := \check{c}_{Q_j^+} - \check{c}_Q$ or estimates of these values are mapped on a single score value. This is typically done by using a function of the form

$$\text{score}(q^-, q^+) = (1 - \mu) \cdot \min\{q^-, q^+\} + \mu \cdot \max\{q^-, q^+\}. \tag{5.1}$$

The *score factor* μ is some number between 0 and 1. It is usually an empirically determined constant, which is sometimes adjusted dynamically through the course of the algorithm.

The default score function in SCIP [Bes+21] is a product,

$$\text{score}(q^-, q^+) = \max\{q^-, \epsilon\} \cdot \max\{q^+, \epsilon\},$$

with $\epsilon = 10^{-6}$, which is superior to the weighted sum of Equation (5.1) [Ach07].

In the forthcoming explanations all cases are symmetric for the left and right subproblem. Therefore we will only consider one direction most of the time, the other will be analogous.

5.1.1 Most Infeasible Branching

This still very common rule chooses the variable with the fractional part closest to 0.5, i.e.,

$$s_j = \min\{\check{x}_j - \lfloor \check{x}_j \rfloor, \lceil \check{x}_j \rceil - \check{x}_j\}.$$

The heuristic reason behind this choice is that this selects a variable where the least tendency can be recognized to which "side" (up or down) the variable should be rounded. When comparing nine different branching rules computationally, Achterberg has discovered that *most infeasible branching* performed poorly [Ach07, Section 5.11].

5.1.2 Least Infeasible Branching

In contrast to the most infeasible branching rule, the *least infeasible branching* strategy prefers variables that are close to integrality:

$$s_j = \max\{\check{x}_j - \lfloor \check{x}_j \rfloor, \lceil \check{x}_j \rceil - \check{x}_j\}.$$

Like most infeasible branching, this strategy yields a very poor performance, even worse than selecting variables randomly [Ach07, Section 5.11].

5.1.3 Pseudocost Branching

The concept of pseudocost was first introduced by Bénichou et al. [Ben+71] to measure the "importance" of the different integer variables in a quantitative way and to forecast the deterioration of the functional value when forcing an integer variable from a non-integer to an integer value. *Pseudocost branching* is a sophisticated rule in the sense that it keeps a history of the success of the variables which have already been branched on. Based on this past experience, this strategy attempts to predict the per-unit change in the objective function for each candidate branching variable. By branching on a variable that is expected to produce a significant change in the objective function, it is more likely that the generated subproblems can be pruned. There are many variations of the original rule proposed in 1971, but in the following we present the one used in SCIP [Bes+21].

Let ς_j^- and ς_j^+ be the objective gains per unit change in variable x_j at node Q after branching in the corresponding direction, that is

$$\varsigma_j^- = \frac{\Delta_j^-}{f_j^-} \quad \text{and} \quad \varsigma_j^+ = \frac{\Delta_j^+}{f_j^+}$$

with $f_j^+ = \lceil \check{x}_j \rceil - \check{x}_j$ and $f_j^- = \check{x}_j - \lfloor \check{x}_j \rfloor$. Let σ_j^+ denote the sum of ς_j^+ over all problems Q, where x_j has been selected as branching variable and the LP relaxation of Q_j^+ has

already been solved and was feasible. Let η_j^+ be the number of these problems, and define σ_j^- and η_j^- to be analogue values for the downwards branch. Then the pseudocosts of variable x_j are calculated as the arithmetic means

$$\Psi_j^- = \frac{\sigma_j^-}{\eta_j^-} \quad \text{and} \quad \Psi_j^+ = \frac{\sigma_j^+}{\eta_j^+}.$$

Using $s_j = \text{score}(f_j^- \Psi_j^-, f_j^+ \Psi_j^+)$ in Algorithm 2 yields what is called *pseudocost branching*.

One difficulty with pseudocost branching is that no information about past branching behavior is available at the beginning of the algorithm, so the pseudocosts for each variable must be initialized in some way.

From a numerical point of view, pseudocost yields much better results than most infeasible branching with a similar compuational effort, hence, there is no reason to employ most infeasible branching in practice [Ach07, Section 5.11].

5.1.4 Strong Branching

Another sophisticated strategy is *strong branching*, which was developed in the context of the traveling salesman problem [App+95]. Strong branching means to test which of the fractional candidates gives the best progress in the dual bound before actually branching on any of them, i.e., this rule branches on the variable that induces the most change in the objective function. This test is done by temporarily introducing an upper bound $x_j \leq \lfloor \check{x} \rfloor$ and subsequently a lower bound $x_j \geq \lceil \check{x} \rceil$ for variable x_j with fractional value $\check{x}$ and solving the linear relaxations.

If we choose as candidate set the full set $F = \{j \in \{1, \ldots, n\} : \check{x}_j \notin \mathbb{Z}\}$ and if we solve the resulting LPs to optimality, we call the strategy *full strong branching*. This approach often leads to small search trees, but the time required to perform variable selection is often prohibitive. Accordingly, most branching rules presented in the literature, including pseudocost branching, may be interpreted as an attempt to find a (fast) estimate of what full strong branching actually means.

One possibility to speed up full strong branching, is to restrict the candidate set in some way, e.g., by considering only a subset $F' \subseteq F$ of the fractional variables.

When analyzing computationally, strong branching and full strong branching need by far the smallest number of branching nodes over all strategies [Ach07, Section 5.11].

5.1.5 Hybrid Strong/Pseudocost Branching

Even with the speedup indicated at the end of Section 5.1.4, the computational burden of strong branching is high, and the higher the speedup, the less precise the decisions are.

On the other hand, the weakness of pseudocost branching is that at the very beginning there is no information available, and s_j basically reflects only the fractionalities for all variables $j \in F$. Many of the early nodes located in the upper part of the search tree where the decisions have the largest impact on the structure of the tree and the subproblems therein. With pseudocost branching, these decisions are taken with respect to the pseudocost values that are not useful yet.

To circumvent these drawbacks the positive aspects of pseudocost and strong branching

are put together in the combination *hybrid strong/pseudocost branching*, where strong branching is applied in the upper part of the tree up to a given depth level d. For nodes with depth larger than d, pseudocost branching is used.

5.2 Node Selection

After a subproblem has been processed, the solving process can continue with any subproblem that is a leaf of the current search tree. The choice of search strategy for selecting the next subproblem has potentially significant consequences for the branch-and-bound procedure, as well as the amount of memory used. In some cases, for very large or challenging problems, it may be necessary to choose a search strategy that requires low memory usage. However, for problems in which memory is not a concern, other search strategies exist which may find an optimal solution very quickly, and thus explore potentially fewer subproblems.

All search methods can be broadly classified into two categories:

1. *uninformed* (or *exhaustive* or *blind*) *methods*, where the search is carried out without any additional information that is already provided in the problem statement, e.g., depth first search,

2. *informed* (or *heuristic*) *methods*, where the search is carried out by using additional information to determine the next step towards finding the solution, e.g., best first search.

Informed search methods are more efficient, low in cost and high in performance as compared to uninformed search methods.

In the MILP branch-and-bound search, the decision which subproblem to process next often entails compromising between two usually opposing goals:

1. finding good feasible MILP solutions to improve the primal (upper) bound, which helps to prune the search tree by bounding, and

2. improving the global dual (lower) bound.

Besides by employing primal heuristics, feasible MILP solutions can be obtained as solutions to LP relaxations of subproblems that happen to be integral. As Linderoth and Savelsbergh [LS99] have already observed, integral LP solutions are typically discovered at significant depths within the search tree. Consequently, to quickly identify feasible solutions of a MILP instance, strategies such as the *depth first search* approach appear to be the most intuitive decision.

Nevertheless, the second objective does not align well with the depth first search strategy, since the nodes with the best (i.e., smallest) lower bounds are usually close to the root node of the search tree. To improve the global dual bound as fast as possible, one should use *best first search* which is to always select a leaf with the currently smallest dual objective value.

5.2.1 Depth First Search

Depth first search (DFS), sometimes called depth first search with backtracking, or last-in/first-out search, was proposed by Little et al. [Lit+63] for the traveling salesman

problem and by Dakin [Dak65] for mixed-integer programming. This node selection rule always chooses a child of the current node as the next subproblem to be processed, i.e., the most recently generated subproblem. If the current node is pruned and therefore has no children, the search backtracks to the most recent ancestor that has another unprocessed child left and selects one of its children. Thus, one selects always a node from the leaf queue with maximal depth in the search tree as can be seen in Figure 5.2.

In addition to its low memory requirements, another advantage of DFS arises when

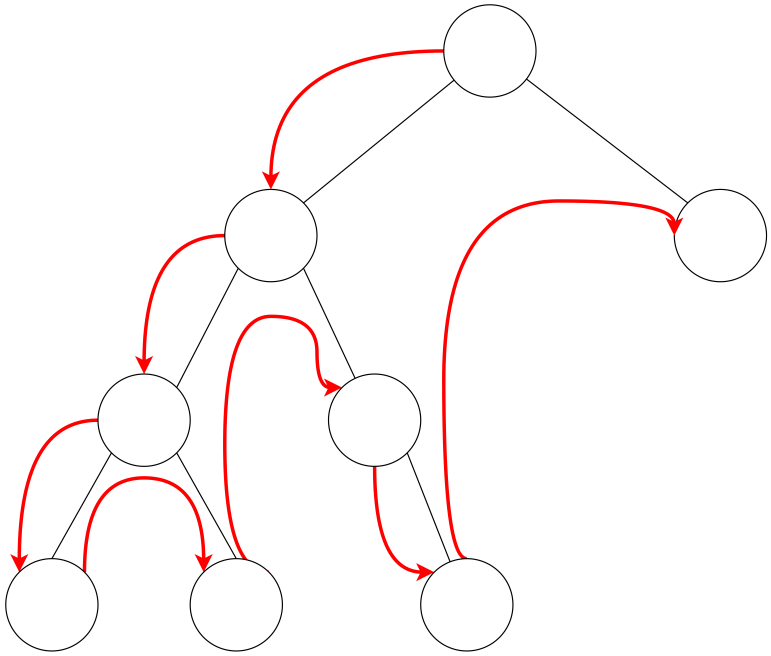


Figure 5.2: Example of how depth first search chooses the next subproblem to process.

solving integer programming problems that use the LP relaxations as lower bounds. Since many branching decisions do not significantly change the structure of the LP relaxation between parent and child nodes, the LP solver can often reuse information from the parent LP solution as a starting point for the child LP solution. Reusing the optimal basis and its LU factorization when solving the child LP is known as *hot starting*, while reusing only the optimal basis is known as *warm starting*.

Two problems arise with the use of the depth first search strategy. The first problem is that naive implementations do not use any information about problem structure or bounds, which means the search process can spend large amounts of exploration time in poor regions of the search space. A related phenomenon, called *thrashing*, occurs when different regions of the search space all fail for the same or similar reasons [Kum92]. For instance, perhaps the presence of a single branching constraint always leads to infeasibility, but the algorithm must explore many more subproblems before the infeasibility is detected.

A different problem arises when the search tree is extremely unbalanced. In other words, if some optimal solutions are close to the root, but there exist long paths that do not lead to an optimal solution, DFS can (unluckily) choose many long, bad paths before it explores a path leading to an optimal solution. However, this computation time often

could be avoided via pruning rules if the search strategy instead chose to explore a short optimal path first. In fact, this behavior of DFS was first noticed on problems where the search tree had unbounded depth [SA83], but the same problem exists in trees with a few extremely long paths.

5.2.2 Best First Search

In settings where sufficient memory is available to store the entire unexplored search tree, the *best first search* (BFS) strategy is often used. This strategy makes use of a heuristic *measure-of-best function* μ, which computes a value $\mu(Q)$ for every unexplored subproblem Q, and selects as the next subproblem to explore the one minimizing μ. There are many choices for the measure-of-best function; one common decision is to aim at improving the global dual bound as fast as possible by always selecting a subproblem with the smallest dual bound of all remaining leaves in the tree. This measure-of-best will encourage exploration of subproblems with better solutions as can be seen in Figure 5.3.

BFS offers a number of significant advantages over DFS; because it is not tied to

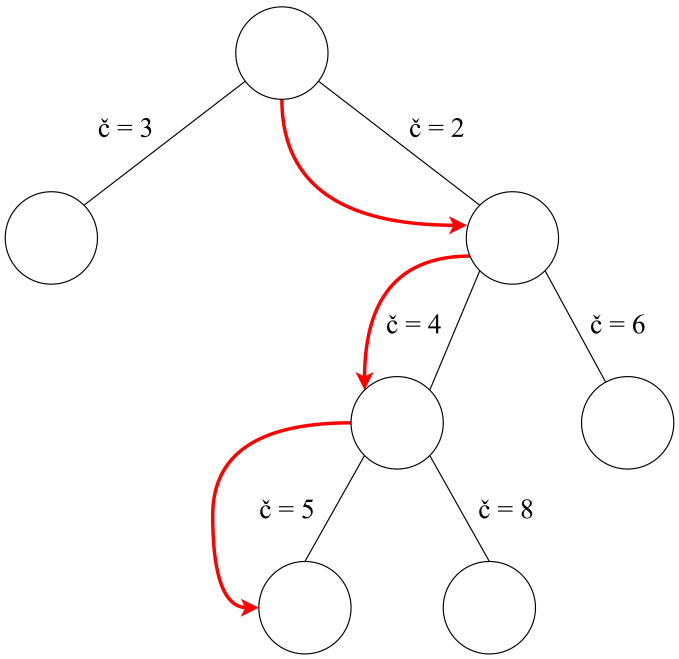


Figure 5.3: Example of a best first search using a measure-of-best function that minimizes the global dual (lower) bound $\check{c}$.

exploring one specific branch of the tree before any other, it is often able to find good solutions earlier in the search process. In fact, this notion has been formalized in a theorem by Dechter and Pearl [DP85] stating that, assuming an admissible μ with no ties and no dominance relations, BFS explores the fewest number of subproblems of any search strategy that has access to the same heuristic function and other information. Nevertheless, even though BFS yields the smaller search trees, DFS can compensate this disadvantage with a faster node processing time, which is the reason why overall, both

methods perform roughly similar [Ach07, Section 6.8].

However, as observed in [SJ12], BFS does still have one potential drawback – if there exist many subproblems for which $\mu(Q) = f(z^*)$, depending on the tie-breaking rule used, BFS may spend much time in middle regions of the search tree and never explore an optimal solution. In this situation, BFS may be slower than some other strategy. To overcome this, many BFS implementations employ diving heuristics or other heuristic methods to drive a subproblem towards a new incumbent solution that can aid in pruning [Bix+00].

5.3 Cut Separation

Cutting planes for integer and mixed-integer programming have been studied since the late 1950's. One of the most fundamental work in this area has been conducted by Gomory who introduced the first cutting plane method [Gom58]. This fundamental idea was applied to the branch-and-bound framework by Padberg and Rinaldi [PR91] to develop a paradigm called *branch-and-cut*. In this paradigm, new cutting planes (sometimes called valid inequalities) are added to the LP relaxation at every subproblem in the search tree (note that a valid inequality is a global constraint – it must apply at the LP relaxation of the root subproblem). The method of Padberg and Rinaldi [PR91] is specific to the well-known traveling salesman problem, but in [BCC96] a generalization of branch-and-cut for binary integer programs is presented. With the work of Balas et al. [Bal+96] in 1996 it became clear that cutting planes, in particular Gomory mixed-integer cuts, are very efficient if they are combined with branch-and-bound.

Like the branch-and-bound method, cutting plane algorithms start with an LP relaxation, but then reduce the admissible range of the relaxed problem by infering additional linear inequalities. While the branch-and-bound method works on some less productive areas of the search tree, cutting plane methods approach the optimal solution in a more targeted way but converge rather slowly in the late phase [Ach07].

The structure of the proofs is based on [Cor08].

5.3.1 Mixed-Integer Rounding Cuts

Nemhauser and Wolsey [NW88; NW90] and Wolsey [Wol98] introduced two definitions of mixed-integer rounding (MIR) inequalities. We follow Wolsey [Wol98].

Lemma 5.1 ([Cor08, Lemma 3])**.** Consider the 2-variable mixed-integer set

$$S := \{(x, y) \in \mathbb{Z} \times \mathbb{R}_{\geq 0} : \; x - y \leq b\}$$

with $b \in \mathbb{R}$. Let $f_0 := b - \lfloor b \rfloor$ be the fractional part of b. Then the *basic mixed-integer inequality*

$$x - \frac{1}{1 - f_0} y \leq \lfloor b \rfloor \tag{5.2}$$

is valid for $\text{conv}(S)$ (see Figure 5.4).

Proof. We show the validity of (5.2) in two different ways, for $x \leq \lfloor b \rfloor$ and $x \geq \lfloor b \rfloor + 1$. As x is an integer, these two cases are sufficient to cover all possibilities.

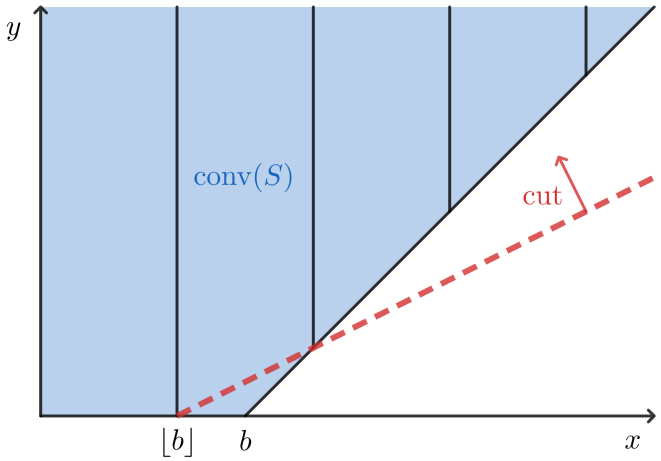


Figure 5.4: Convex hull of S and the basic mixed-integer cut. Based on: [Cor08, Figure 12].

- $x \leq \lfloor b \rfloor$: as y is nonnegative, we know that $-y \leq 0$ holds. Multiplying this inequality with the strictly positive factor $\frac{1}{1-f_0}$ yields $-\frac{1}{1-f_0}y \leq 0$. Adding this to our inequality $x \leq \lfloor b \rfloor$ we obtain (5.2).
- $x \geq \lfloor b \rfloor + 1$: this is equivalent to $-x \leq -\lfloor b \rfloor - 1$. Multiplying with $\frac{f_0}{1-f_0}$ yields

$$-\frac{f_0}{1-f_0}x \leq -\frac{f_0}{1-f_0}\lfloor b \rfloor - \frac{f_0}{1-f_0}. \tag{5.3}$$

  We also know $x - y \leq b$ due to the definition of S. Multiplying with $\frac{1}{1-f_0}$ we obtain

$$\frac{1}{1-f_0}x - \frac{1}{1-f_0}y \leq \frac{1}{1-f_0}b. \tag{5.4}$$

  Now, adding (5.3) to (5.4) and using $b = f_0 + \lfloor b \rfloor$ yields

$$\begin{aligned}
&-\frac{f_0}{1-f_0}x + \frac{1}{1-f_0}x - \frac{1}{1-f_0}y \leq -\frac{f_0}{1-f_0}\lfloor b \rfloor - \frac{f_0}{1-f_0} + \frac{1}{1-f_0}b \\
\iff \quad &\frac{-f_0+1}{1-f_0}x - \frac{1}{1-f_0}y \leq -\frac{f_0}{1-f_0}\lfloor b \rfloor - \frac{f_0}{1-f_0} + \frac{f_0}{1-f_0} + \frac{1}{1-f_0}\lfloor b \rfloor \\
\iff \quad &x - \frac{1}{1-f_0}y \leq \frac{-f_0+1}{1-f_0}\lfloor b \rfloor \\
\iff \quad &x - \frac{1}{1-f_0}y \leq \lfloor b \rfloor.
\end{aligned}$$

□

Now we can proof the validity of the *mixed-integer rounding cut* (MIR cut).

Theorem 5.2 ([Cor08, Theorem 8])**.** Consider a mixed-integer set defined by a single inequality

$$S := \left\{(x, y) \in \mathbb{Z}^n_{\geq 0} \times \mathbb{R}^p_{\geq 0} : a^\top x + g^\top y \leq b\right\}$$

with $a \in \mathbb{R}^n$, $g \in \mathbb{R}^p$, and $b \in \mathbb{R}$. Let $f_0 := b - \lfloor b \rfloor$ and $f_j := a_j - \lfloor a_j \rfloor$ be the fractional parts of b and a_j, respectively, with $0 < f_0 < 1$. Then the *mixed-integer rounding inequality*

$$\sum_{j=1}^{n} \left(\lfloor a_j \rfloor + \frac{\max\{f_j - f_0, 0\}}{1 - f_0} \right) x_j + \frac{1}{1 - f_0} \sum_{g_j < 0} g_j y_j \leq \lfloor b \rfloor \tag{MIR}$$

is valid for $\operatorname{conv}(S)$.

Proof. The main idea is to apply Lemma 5.1. We can split sums and obtain

$$a^\top x + g^\top y = \sum_{j=1}^{n} a_j x_j + \sum_{j=1}^{p} g_j y_j = \sum_{j:\, f_j \leq f_0} a_j x_j + \sum_{j:\, f_j > f_0} a_j x_j + \underbrace{\sum_{g_j \geq 0} g_j y_j}_{\geq 0} + \sum_{g_j < 0} g_j y_j$$

as $y \geq 0$. We want to bound this from below. Hence, we can just remove positive terms. We also know that for all $\alpha \in \mathbb{R}$ the inequality $\alpha \geq \lfloor \alpha \rfloor$ holds. As $x \geq 0$ we also know $\alpha x \geq \lfloor \alpha \rfloor x$, which yields another way to estimate

$$\begin{aligned} a^\top x + g^\top y &\geq \sum_{j:\, f_j \leq f_0} a_j x_j + \sum_{j:\, f_j > f_0} a_j x_j + \sum_{g_j < 0} g_j y_j \\ &\geq \sum_{j:\, f_j \leq f_0} \lfloor a_j \rfloor x_j + \sum_{j:\, f_j > f_0} a_j x_j + \sum_{g_j < 0} g_j y_j. \end{aligned}$$

Thus, we can conclude from $a^\top x + g^\top y \leq b$ that

$$\sum_{j:\, f_j \leq f_0} \lfloor a_j \rfloor x_j + \sum_{j:\, f_j > f_0} a_j x_j + \sum_{g_j < 0} g_j y_j \leq b. \tag{5.5}$$

Let

$$\begin{aligned} w &:= \sum_{j:\, f_j \leq f_0} \lfloor a_j \rfloor x_j + \sum_{j:\, f_j > f_0} \lceil a_j \rceil x_j, \\ z &:= \underbrace{- \sum_{g_j < 0} g_j y_j}_{\geq 0} + \sum_{f_j > f_0} \underbrace{(1 - f_j)}_{\geq 0} x_j, \end{aligned}$$

hence $z \in \mathbb{R}_{\geq 0}$. We have

$$\begin{aligned} w - z &= \sum_{j:\, f_j \leq f_0} \lfloor a_j \rfloor x_j + \sum_{j:\, f_j > f_0} \lceil a_j \rceil x_j + \sum_{g_j < 0} g_j y_j - \sum_{j:\, f_j > f_0} (1 - f_j)\, x_j \\ &= \sum_{j:\, f_j \leq f_0} \lfloor a_j \rfloor x_j + \sum_{j:\, f_j > f_0} \lceil a_j \rceil x_j - \underbrace{(1 - f_j)}_{1 - a_j + \lfloor a_j \rfloor} x_j + \sum_{g_j < 0} g_j y_j \\ &= \sum_{j:\, f_j \leq f_0} \lfloor a_j \rfloor x_j + \sum_{j:\, f_j > f_0} \lceil a_j \rceil x_j - x_j + a_j x_j - \lfloor a_j \rfloor x_j + \sum_{g_j < 0} g_j y_j \\ &= \sum_{j:\, f_j \leq f_0} \lfloor a_j \rfloor x_j + \sum_{j:\, f_j > f_0} \underbrace{(\lceil a_j \rceil - \lfloor a_j \rfloor)}_{=1} x_j - x_j + a_j x_j + \sum_{g_j < 0} g_j y_j \\ &= \sum_{j:\, f_j \leq f_0} \lfloor a_j \rfloor x_j + \sum_{j:\, f_j > f_0} a_j x_j + \sum_{g_j < 0} g_j y_j \overset{(5.5)}{\leq} b. \end{aligned}$$

To summarize, $w - z \leq b$, $w \in \mathbb{Z}$ and $z \in \mathbb{R}_{\geq 0}$, thus we can apply Lemma 5.1 and obtain

$$w - \frac{1}{1-f_0} z \leq \lfloor b \rfloor.$$

Substituting w and z yields

$$\sum_{j:\, f_j \leq f_0} \lfloor a_j \rfloor x_j + \sum_{j:\, f_j > f_0} \lceil a_j \rceil x_j - \frac{1}{1-f_0} \left(- \sum_{g_j < 0} g_j y_j + \sum_{j:\, f_j > f_0} (1 - f_j)\, x_j \right) \leq \lfloor b \rfloor,$$

which is equivalent to

$$\sum_{j:\, f_j \leq f_0} \lfloor a_j \rfloor x_j + \sum_{f_j > f_0} \left(\lceil a_j \rceil - \frac{1-f_j}{1-f_0} \right) x_j + \frac{1}{1-f_0} \sum_{g_j < 0} g_j y_j \leq \lfloor b \rfloor \tag{5.6}$$

after combining sums. Let us take a look at the second sum: as $f_j > f_0 \geq 0$, hence, $a_j \notin \mathbb{Z}$, we can use the representation $\lceil a_j \rceil - \lfloor a_j \rfloor = 1$ and rewrite

$$\begin{aligned}
\sum_{f_j > f_0} \left(\lceil a_j \rceil - \frac{1-f_j}{1-f_0} \right) x_j &= \sum_{f_j > f_0} \left(\lfloor a_j \rfloor + 1 + \frac{f_j - 1}{1-f_0} \right) x_j \\
&= \sum_{f_j > f_0} \left(\lfloor a_j \rfloor + \frac{(1-f_0) + f_j - 1}{1-f_0} \right) x_j \\
&= \sum_{f_j > f_0} \left(\lfloor a_j \rfloor + \frac{f_j - f_0}{1-f_0} \right) x_j
\end{aligned}$$

Thus, inequality (5.6) is equivalent to

$$\sum_{j:\, f_j \leq f_0} \lfloor a_j \rfloor x_j + \sum_{f_j > f_0} \left(\lfloor a_j \rfloor + \frac{f_j - f_0}{1-f_0} \right) x_j + \frac{1}{1-f_0} \sum_{g_j < 0} g_j y_j \leq \lfloor b \rfloor.$$

We want to combine the first two sums. The case of $f_j \leq f_0$ should yield $\lfloor a_j \rfloor x_j$, while the case of $f_j > f_0$ should obtain the additional term $\frac{f_j - f_0}{1-f_0} x_j$. We can construct this case distinction using a max-term as the latter case means $f_j - f_0 > 0$, hence,

$$\sum_{j=1}^{n} \left(\lfloor a_j \rfloor + \frac{\max\{f_j - f_0, 0\}}{1-f_0} \right) x_j + \frac{1}{1-f_0} \sum_{g_j < 0} g_j y_j \leq \lfloor b \rfloor.$$

□

5.3.2 Gomory Mixed-Integer Cuts

Gomory mixed-integer cuts (GMI cuts) can be stated as follows [Gom60; Bal+96].

Theorem 5.3 ([Cor08, Section 4.1])**.** Consider a mixed-integer set defined by a single equality constraint

$$S := \left\{ (x, y) \in \mathbb{Z}^n_{\geq 0} \times \mathbb{R}^p_{\geq 0} : \, a^\top x + g^\top y = b \right\}$$

with $a \in \mathbb{R}^n$, $g \in \mathbb{R}^p$, and $b \in \mathbb{R}$. Let $f_0 := b - \lfloor b \rfloor$ and $f_j := a_j - \lfloor a_j \rfloor$ be the fractional parts of b and a_j, respectively. Then the *Gomory mixed-integer inequality*

$$\sum_{f_j \leq f_0} \frac{f_j}{f_0} x_j + \sum_{f_j > f_0} \frac{1 - f_j}{1 - f_0} x_j + \sum_{g_j > 0} \frac{g_j}{f_0} y_j - \sum_{g_j < 0} \frac{g_j}{1 - f_0} y_j \geq 1 \tag{GMI}$$

is valid for S.

Proof. The idea of this proof is to rewrite the equality constraint $a^\top x + g^\top y = b$ to obtain another equality from which we pick an integer term $k \in \mathbb{Z}$ to motivate a case distinction. The two cases lead to an inequality, respectively, which yield the GMI cut when they are combined.

We can rewrite the equality constraint $a^\top x + g^\top y = b$ by using the representation $a_j = f_j + \lfloor a_j \rfloor$ and splitting sums,

$$\begin{aligned}
a^\top x + g^\top y &= \sum_{j=1}^{n} a_j x_j + \sum_{j=1}^{p} g_j y_j \\
&= \sum_{j=1}^{n} f_j x_j + \sum_{j=1}^{n} \lfloor a_j \rfloor x_j + \sum_{j=1}^{p} g_j y_j \\
&= \sum_{f_j \leq f_0} f_j x_j + \sum_{f_j > f_0} f_j x_j + \sum_{j=1}^{n} \lfloor a_j \rfloor x_j + \sum_{j=1}^{p} g_j y_j \\
&= \sum_{f_j \leq f_0} f_j x_j + \sum_{f_j > f_0} (f_j - 1) x_j + \sum_{j \colon f_j > f_0} x_j + \sum_{j=1}^{n} \lfloor a_j \rfloor x_j + \sum_{j=1}^{p} g_j y_j = b.
\end{aligned}$$

Combining the last equation with the representation $b = f_0 + \lfloor b \rfloor$, we obtain

$$\sum_{f_j \leq f_0} f_j x_j + \sum_{f_j > f_0} (f_j - 1) x_j + \sum_{j \colon f_j > f_0} x_j + \sum_{j=1}^{n} \lfloor a_j \rfloor x_j + \sum_{j=1}^{p} g_j y_j = f_0 + \lfloor b \rfloor,$$

which we can rewrite as

$$\sum_{f_j \leq f_0} f_j x_j + \sum_{f_j > f_0} (f_j - 1) x_j + \sum_{j=1}^{p} g_j y_j = \left(\lfloor b \rfloor - \sum_{j=1}^{n} \lfloor a_j \rfloor x_j - \sum_{j \colon f_j > f_0} x_j \right) + f_0. \tag{5.7}$$

Since $k := \left(\lfloor b \rfloor - \sum_{j=1}^{n} \lfloor a_j \rfloor x_j - \sum_{j \colon f_j > f_0} x_j \right) \in \mathbb{Z}$ as every parameter and variable involved is integer, we can distinguish between two cases:

- $k \leq -1$: inserting this inequality into equation (5.7), we obtain

  $$\sum_{f_j \leq f_0} f_j x_j + \sum_{f_j > f_0} (f_j - 1) x_j + \sum_{j=1}^{p} g_j y_j \leq -1 + f_0.$$

  Dividing by $1 - f_0$ yields

  $$- \sum_{f_j \leq f_0} \frac{f_j}{1 - f_0} x_j + \sum_{f_j > f_0} \frac{1 - f_j}{1 - f_0} x_j - \sum_{j=1}^{p} \frac{g_j}{1 - f_0} y_j \geq 1. \tag{5.8}$$

- $k \geq 0$: inserting this inequality into equation (5.7), we obtain

$$\sum_{f_j \leq f_0} f_j x_j + \sum_{f_j > f_0} (f_j - 1)\, x_j + \sum_{j=1}^{p} g_j y_j \geq f_0.$$

Similarly, dividing by f_0 yields

$$\sum_{f_j \leq f_0} \frac{f_j}{f_0} x_j - \sum_{f_j > f_0} \frac{1 - f_j}{f_0} x_j + \sum_{j=1}^{p} \frac{g_j}{f_0} y_j \geq 1. \tag{5.9}$$

Hence, the case distinction yields the inequalities (5.8) and (5.9), and is of the form "$c^\top z \geq 1$ or $d^\top z \geq 1$", which implies $\sum_j \max\{c_j, d_j\}\, z_j \geq 1$ for any $z \geq 0$. For each of the variables x_j and y_j, we have to pick the maximum coefficient from (5.8) and (5.9). Looking at the sums individually when comparing these two inequalities, for each j there is always a negative and a positive one. Hence, by choosing the positive sum we obtain

$$\sum_{f_j \leq f_0} \frac{f_j}{f_0} x_j + \sum_{f_j > f_0} \frac{1 - f_j}{1 - f_0} x_j + \sum_{g_j > 0} \frac{g_j}{f_0} y_j - \sum_{g_j < 0} \frac{g_j}{1 - f_0} y_j \geq 1.$$

□

Remark 5.4 ([Cor08, Remark 8])**.** In the pure integer programming case, the GMI inequality reduces to

$$\sum_{f_j \leq f_0} \frac{f_j}{f_0} x_j + \sum_{f_j > f_0} \frac{1 - f_j}{1 - f_0} x_j \geq 1.$$

Since $\frac{1-f_j}{1-f_0} < \frac{f_j}{f_0}$ when $f_j > f_0$, it follows that the GMI inequality dominates

$$\sum_{j=1}^{n} f_j x_j \geq f_0,$$

which is known as the *Gomory fractional cut* (GF cut). The fractional cut can also be derived using Chvátal's procedure which we will not take a closer look at as this thesis focusses on the strictly mixed-integer case.

We can show that the GMI inequality and the MIR inequality are identical.

Lemma 5.5 ([Cor08, Lemma 4])**.** Consider a mixed-integer set defined by a single inequality

$$S := \left\{(x, y) \in \mathbb{Z}^n_{\geq 0} \times \mathbb{R}^p_{\geq 0} : a^\top x + g^\top y \leq b\right\}$$

with $a \in \mathbb{R}^n$, $g \in \mathbb{R}^p$, and $b \in \mathbb{R}$. Then the MIR inequality (MIR) is identical to the GMI inequality (GMI).

Proof. Let us outline the idea of this proof. As the MIR cut only considers a mixed-integer set defined by a single inequality constraint, while the GMI cut considers the stronger mixed-integer set defined by an equality constraint, we add a slack variable $s \in \mathbb{R}^m_{\geq 0}$ such that $a^\top x + g^\top y + s = b$ to generate the GMI inequality in the (x, y, s)-space. We then rewrite this inequality and substitute $s = b - a^\top x - g^\top y$ to get the GMI inequality in the (x, y)-space. Lastly, we show that the resulting inequality is equivalent to the MIR

inequality.

Recall the two inequalities:

$$\sum_{f_j \le f_0} \frac{f_j}{f_0} x_j + \sum_{f_j > f_0} \frac{1-f_j}{1-f_0} x_j + \sum_{g_j > 0} \frac{g_j}{f_0} y_j - \sum_{g_j < 0} \frac{g_j}{1-f_0} y_j \ge 1 \tag{GMI}$$

and

$$\sum_{j=1}^{n} \left(\lfloor a_j \rfloor + \frac{\max\{f_j - f_0, 0\}}{1-f_0} \right) x_j + \frac{1}{1-f_0} \sum_{j:\, g_j < 0} g_j y_j \le \lfloor b \rfloor. \tag{MIR}$$

The Gomory mixed-integer inequality is obtained by adding a slack variable $s \in \mathbb{R}^m_{\ge 0}$ such that $a^\top x + g^\top y + s = b$, generating the GMI inequality (GMI) in the (x, y, s)-space,

$$\sum_{f_j \le f_0} \frac{f_j}{f_0} x_j + \sum_{f_j > f_0} \frac{1-f_j}{1-f_0} x_j + \sum_{g_j > 0} \frac{g_j}{f_0} y_j - \sum_{g_j < 0} \frac{g_j}{1-f_0} y_j + \frac{1}{f_0} s \ge 1. \tag{5.10}$$

This can be understood by modifying the proof of Theorem 5.3 in order to add the slack variable: Equation (5.7) then entails the additional slack term,

$$\sum_{f_j \le f_0} f_j x_j + \sum_{f_j > f_0} (f_j - 1)\, x_j + \sum_{j=1}^{p} g_j y_j + s = k + f_0.$$

The case distinction $k \le -1$ yields

$$-\sum_{f_j \le f_0} \frac{f_j}{1-f_0} x_j + \sum_{f_j > f_0} \frac{1-f_j}{1-f_0} x_j - \sum_{j=1}^{p} \frac{g_j}{1-f_0} y_j - \frac{1}{1-f_0} s \ge 1$$

and $k \ge 0$ yields

$$\sum_{f_j \le f_0} \frac{f_j}{f_0} x_j - \sum_{f_j > f_0} \frac{1-f_j}{f_0} x_j + \sum_{j=1}^{p} \frac{g_j}{f_0} y_j + \frac{1}{f_0} s \ge 1$$

as modifications of (5.8) and (5.9), respectively. When choosing the maximum,

$$\max\left\{ -\frac{1}{1-f_0} s, \frac{1}{f_0} s \right\} = \frac{1}{f_0} s,$$

it is added to the left-hand side of the original GMI inequality (GMI), resulting in (5.10).

Now we want to obtain the MIR inequality from (5.10). We multiply the inequality with f_0,

$$\sum_{f_j \le f_0} f_j x_j + f_0 \sum_{f_j > f_0} \frac{1-f_j}{1-f_0} x_j + \sum_{g_j > 0} g_j y_j - f_0 \sum_{g_j < 0} \frac{g_j}{1-f_0} y_j + s \ge f_0 = b - \lfloor b \rfloor.$$

Bringing b to the other side and multiplying with -1, we obtain

$$-\sum_{f_j \le f_0} f_j x_j - f_0 \sum_{f_j > f_0} \frac{1-f_j}{1-f_0} x_j - \sum_{g_j > 0} g_j y_j + f_0 \sum_{g_j < 0} \frac{g_j}{1-f_0} y_j - s + b \le \lfloor b \rfloor.$$

Substituting $s = b - a^\top x - g^\top y$ to get the modified GMI inequality in the (x, y)-space, the result is

$$-\sum_{f_j \le f_0} f_j x_j - f_0 \sum_{f_j > f_0} \frac{1-f_j}{1-f_0} x_j - \sum_{g_j > 0} g_j y_j + f_0 \sum_{g_j < 0} \frac{g_j}{1-f_0} y_j + \sum_{j=1}^{n} a_j x_j + \sum_{j=1}^{p} g_j y_j \le \lfloor b \rfloor.$$

By merging the last two sums with the first ones and using the representations $b = f_0 + \lfloor b \rfloor$ and $a_j = f_j + \lfloor a_j \rfloor$, we obtain

$$\begin{aligned}
\lfloor b \rfloor &\ge -\sum_{f_j \le f_0} f_j x_j - f_0 \sum_{f_j > f_0} \frac{1-f_j}{1-f_0} x_j - \sum_{g_j > 0} g_j y_j + f_0 \sum_{g_j < 0} \frac{g_j}{1-f_0} y_j + \sum_{j=1}^{n} a_j x_j + \sum_{j=1}^{p} g_j y_j \\
&= \sum_{f_j \le f_0} (a_j - f_j)\, x_j + \sum_{f_j > f_0} \left(a_j - f_0 \frac{1-f_j}{1-f_0} \right) x_j + \sum_{g_j > 0} (g_j - g_j)\, y_j + \sum_{g_j < 0} \left(g_j + f_0 \frac{g_j}{1-f_0} \right) y_j \\
&= \sum_{f_j \le f_0} \lfloor a_j \rfloor x_j + \sum_{f_j > f_0} \underbrace{\left(\lfloor a_j \rfloor + f_j - f_0 \frac{1-f_j}{1-f_0} \right)}_{= \lfloor a_j \rfloor + \frac{f_j(1-f_0) - f_0\left(1-f_j\right)}{1-f_0} = \lfloor a_j \rfloor + \frac{f_j - f_0}{1-f_0}} x_j + \sum_{g_j < 0} \underbrace{\left(1 + \frac{f_0}{1-f_0} \right)}_{= \frac{1-f_0+f_0}{1-f_0} = \frac{1}{1-f_0}} g_j y_j \\
&= \sum_{f_j \le f_0} \lfloor a_j \rfloor x_j + \sum_{f_j > f_0} \left(\lfloor a_j \rfloor + \frac{f_j - f_0}{1-f_0} \right) x_j + \frac{1}{1-f_0} \sum_{g_j < 0} g_j y_j.
\end{aligned}$$

We can combine the first two sums of the last equation similarly to the way we did in the proof of Theorem 5.2, yielding

$$\sum_{j=1}^{n} \left(\lfloor a_j \rfloor + \frac{\max\{f_j - f_0, 0\}}{1-f_0} \right) x_j + \frac{1}{1-f_0} \sum_{j:\, g_j < 0} g_j y_j \le \lfloor b \rfloor,$$

which is the mixed-integer rounding inequality (MIR). □

Example 5.6 ([Wol98, Example 8.12])**.** We consider the mixed-integer program

$$\begin{array}{rrcrcrcl}
z = \max & 4x_1 & - & x_2 & & & & \\
 & 7x_1 & - & 2x_2 & \le & 14 & & \\
 & & & x_2 & \le & 3 & & \\
 & 2x_1 & - & 2x_2 & \le & 3 & & \\
 & x_1 & \in & \mathbb{Z}_+, & x_2 & \ge 0. & &
\end{array}$$

We can draw the feasible set with these inequality constraints, yielding Figure 5.5. We introduce slack variables $x_3, x_4, x_5 \ge 0$ and transform our program to standard form,

$$\begin{array}{rrcrcrcrcl}
z = \max & 4x_1 & - & x_2 & & & & & & \\
 & 7x_1 & - & 2x_2 & +x_3 & & & & = & 14 \\
 & & & x_2 & & +x_4 & & & = & 3 \\
 & 2x_1 & - & 2x_2 & & & +x_5 & & = & 3 \\
 & x_1 & \in \mathbb{Z}_+, & x_2, & x_3, & x_4, & x_5 & & \ge & 0.
\end{array} \tag{5.11}$$

Solving as a linear program gives the solution $(\frac{20}{7}, 3) \notin \mathbb{Z}_+ \times \mathbb{R}$, as seen in Figure 5.6, and the simplex tableau

$$\begin{array}{rrrrrrrcl}
z = \max \frac{59}{7} & & & -\frac{4}{7}x_3 & -\frac{1}{7}x_4 & & & & \\
 & x_1 & & +\frac{1}{7}x_3 & +\frac{2}{7}x_4 & & & = & \frac{20}{7} \\
 & & x_2 & & +x_4 & & & = & 3 \\
 & & & -\frac{2}{7}x_3 & +\frac{10}{7}x_4 & +x_5 & & = & \frac{23}{7} \\
 & x_1 \in \mathbb{Z}_+, & x_2, & x_3, & x_4, & x_5 & & \ge & 0.
\end{array}$$

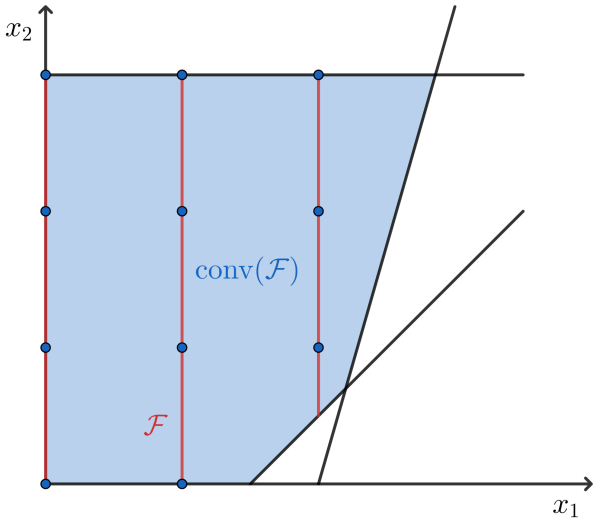


Figure 5.5: Feasible set $\mathcal{F}$ and its convex hull $\text{conv}(\mathcal{F})$.

We want to introduce a cutting plane to accelerate the performance of our solving algo-

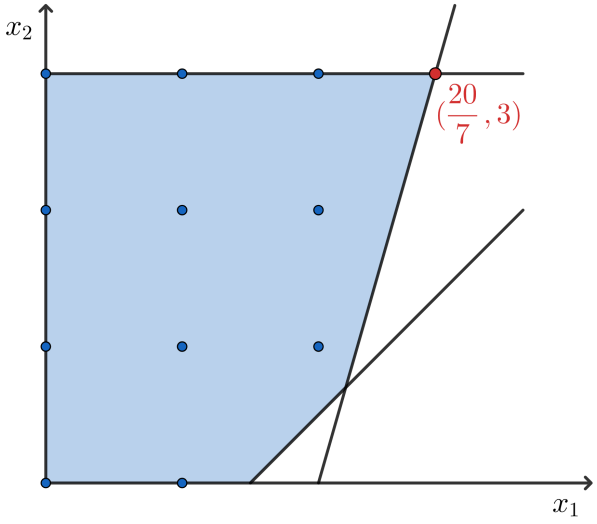


Figure 5.6: The point calculated by the simplex algorithm is not feasible.

rithm. The basic variable x_1 is fractional and the first row implies

$$x_1 \leq \frac{20}{7}.$$

Applying the MIR cut (MIR) to this inequality yields

$$\begin{aligned} &\left(\lfloor a_1 \rfloor + \frac{\max\{f_1 - f_0, 0\}}{1 - f_0}\right) x_1 \leq \lfloor b \rfloor \\ \iff &\left(\lfloor 1 \rfloor + \frac{\max\{0 - \frac{6}{7}, 0\}}{1 - \frac{6}{7}}\right) x_1 \leq \lfloor \frac{20}{7} \rfloor \\ \iff &\; x_1 \leq 2. \end{aligned}$$

Alternatively, we can also apply the GMI cut (GMI) to the first row and obtain

$$\begin{aligned} & \frac{f_1}{f_0}x_1 + \frac{g_3}{f_0}x_3 + \frac{g_4}{f_0}x_4 \geq 1 \\ \iff & \frac{0}{\frac{6}{7}}x_1 + \frac{\frac{1}{7}}{\frac{6}{7}}x_3 + \frac{\frac{2}{7}}{\frac{6}{7}}x_4 \geq 1 \\ \iff & \frac{1}{6}x_3 + \frac{1}{3}x_4 \geq 1. \end{aligned}$$

Using the equalities

$$x_4 = -x_2 + 3 \quad \text{and} \quad x_3 = -7x_1 + 2x_2 + 14$$

from the second and third row of the standard form (5.11), and substituting, we further obtain

$$\begin{aligned} \frac{1}{6}\left(-7x_1 + 2x_2 + 14\right) + \frac{1}{3}\left(-x_2 + 3\right) &\geq 1 \\ \iff -\frac{7}{6}x_1 + \frac{14}{6} &\geq 0 \\ \iff 2 &\geq x_1. \end{aligned}$$

We can see that we have just reproduced the idea of the proof of Lemma 5.5 by calculating a concrete example, namely the first row of (5.6), and have obtained the same cut: $x_1 \leq 2$. Adding this cut to our LP problem and reapplying the simplex algorithm leads to the solution $x = (2, \frac{1}{2})$, which is feasible and therefore optimal for the mixed-integer program. This can also be seen graphically in Figure 5.7.

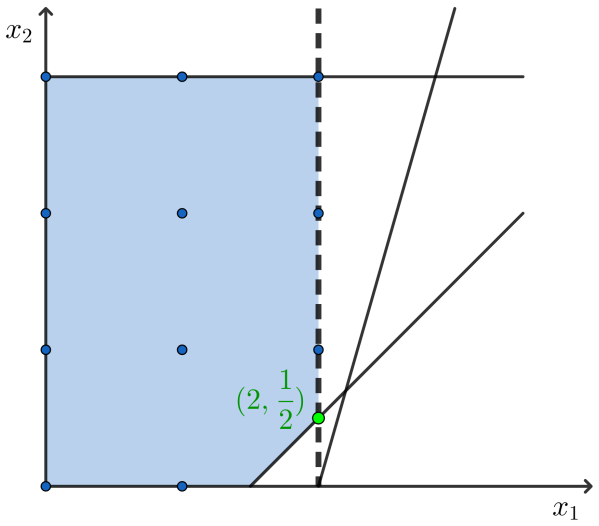


Figure 5.7: Added cut and optimal point.

6 Conclusion and Outlook

In this thesis, we presented the branch-and-cut framework for solving mixed-integer linear optimization problems, which is a hybrid of the branch-and-bound algorithm and the cutting planes method.

We gave a brief overview of some branching rules with both *most* and *least infeasible branching* yielding by far worse results than sophisticated methods like *pseudocost branching* or *(full) strong branching*. Combining these two methods to the *hybrid strong/pseudocost branching* combines their advantages and balances out their disadvantages. We took a closer look at two different node selection methods, namely *depth first search* (DFS) and *best first search* (BFS). BFS yields smaller search trees, but DFS can compensate this disadvantage with a faster node processing time – overall both methods perform roughly similar. The structure of those two sections was based on [Ach07, Chapter 5, 6], we explained the brief overview given there in greater detail. In particular, we categorized the methods in a broader context and addressed their advantages and disadvantages. The focus was on plane separation: We introduced and proved the *mixed-integer rounding cut* and the *Gomory mixed-integer cut*. We then showed that the cuts are equivalent. The reviewed literature only provided a brief outline of the ideas behind the proofs, hence, we retraced the proofs in much more detail. The example in the last section was also explained and visualized in more detail than in the textbook.

Having focused on the theoretical foundations, the next logical step would be the numerical implementation and to analyze the efficiency of branch-and-cut computationally – varying and comparing branching rules, node selection, and cutting plane methods. It would also be interesting to either specify and take a closer look at the pure integer case, as Remark 5.4 already hinted at, or to generalize and study nonlinear problems.

A Appendix

A.1 Complexity: $\mathcal{P}$ and $\mathcal{NP}$

This section contains content from [Pam22].

In theoretical computer science and mathematics, computational complexity theory is a field of study that seeks to understand and analyze the inherent difficulty of computational problems. It provides a framework for classifying problems based on their computational requirements (i.e., the amount of resources needed to solve them, such as time and storage) and for understanding the limits of efficient computation.

- $\mathcal{P}$ is a fundamental complexity class which contains all decision problems that can be solved in polynomial time by a deterministic Turing machine[3].
- $\mathcal{NP}$ is a complexity class used to classify decision problems. $\mathcal{NP}$ is the set of decision problems solvable in polynomial time by a nondeterministic Turing machine, or alternatively, the set of decision problems verifiable in polynomial time by a deterministic Turing machine.
- A problem A is $\mathcal{NP}$-hard when every problem B in $\mathcal{NP}$ can be reduced in polynomial time to A; that is, assuming a solution for A takes 1 unit time, A's solution can be used to solve B in polynomial time. As a consequence, finding a polynomial time algorithm to solve any $\mathcal{NP}$-hard problem would give polynomial time algorithms for all the problems in $\mathcal{NP}$. As it is suspected that $\mathcal{P} \neq \mathcal{NP}$ [Wol98], it is unlikely that such an algorithm exists, i.e., $\mathcal{P} = \mathcal{NP}$ (see Figure A.1).
- A problem is $\mathcal{NP}$-complete if it is $\mathcal{NP}$-hard and contained in $\mathcal{NP}$.

For more details about complexity, I refer to [Wol98, Chapter 6].

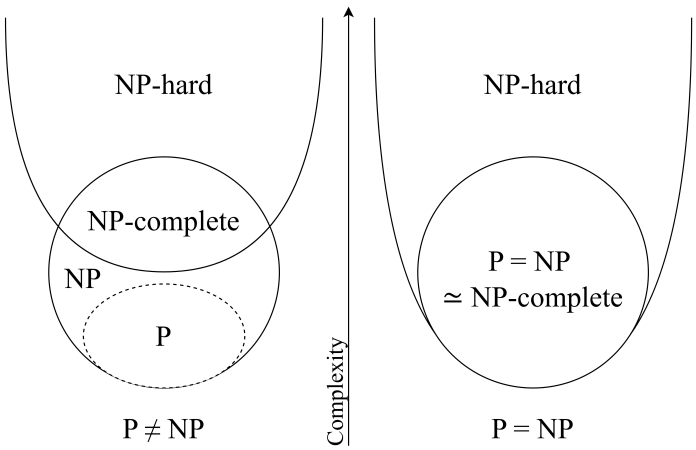


Figure A.1: Euler diagram for $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-complete, and $\mathcal{NP}$-hard problems. Under the assumption that $\mathcal{P} \neq \mathcal{NP}$, the existence of problems within $\mathcal{NP}$ but outside both $\mathcal{P}$ and $\mathcal{NP}$-complete was established by Ladner [Lad75].

[3]See [Pam22] for the definition of a Turing machine.

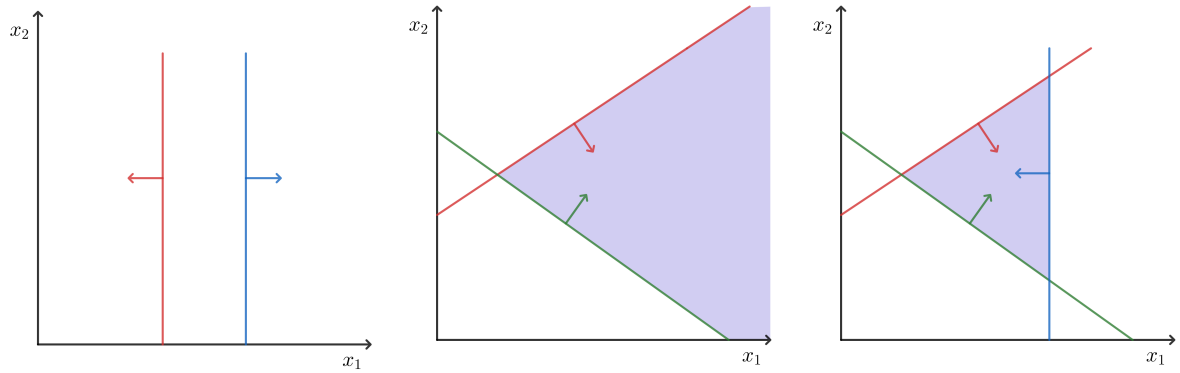
A.2 Linear Programming: Simplex

The simplex algorithm, developed by George B. Dantzig in 1947, operates on linear programs in the canonical form,

$$\begin{aligned} \max z &= c^\top x \\ \text{s.t. } Ax &\leq b \\ x &\geq 0 \end{aligned}$$

with $c, x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$. The feasible set $\mathcal{F}$ is characterized by the nonnegativity of x and m inequalities $Ax \leq b$, which are visualized exemplarily in Figure A.2 for $n = 2$.

There are three possible cases for the feasible set: it can be empty (i.e., the LP is infeasible), unbounded or bounded. If the feasible set is bounded, it is convex (a convex polytope to be precise), and an extreme point or vertex of this convex polytope is known as a *basic feasible point*.



(a) $\mathcal{F}$ empty, LP is infeasible. (b) $\mathcal{F}$ unbounded, LP unbounded. (c) $\mathcal{F}$ bounded, LP bounded.

Figure A.2: There are three possibilities how the feasible set $\mathcal{F}$ can look like.

The *fundamental theorem of linear programming* by Dantzig states that if the feasible set is bounded, the optimal solution is obtained at a vertex of the polytope:

Theorem A.1. [Van14, Theorem 3.4] For a linear programming (LP) problem, the following statements hold:

(i) If there is no optimal solution, then the problem is either infeasible or unbounded.

(ii) If a feasible solution exists, then a basic feasible solution exists.

(iii) If an optimal solution exists, then a basic optimal solution exists.

The idea of the simplex algorithm is that in each iteration, it moves along the vertices and tries to increase the value of the objective function. Since there is only a finite number of vertices, the algorithm terminates.

For more details about the theoretical foundations and numerical implementation of the simplex, I refer to [Jäk17, Section 2].

B Bibliography

[Ach07] Tobias Achterberg. "Constraint Integer Programming". PhD thesis. Berlin: Technische Universität Berlin, 2007.

[App+95] David Applegate et al. *Finding cuts in the TSP*. Technical Report. DIMACS, Mar. 1995. URL: http://www.optimization-online.org/DB_HTML/2021/12/8728.html.

[ASS17] Amir Ahmadi-Javid, Pardis Seyedi, and Siddhartha S. Syam. "A survey of healthcare facility location". In: *Computers & Operations Research* 79 (2017), pp. 223–263. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2016.05.018. URL: https://www.sciencedirect.com/science/article/pii/S0305054816301253.

[Bal+96] Egon Balas et al. "Gomory cuts revisited". In: *Operations Research Letters* 19.1 (1996), pp. 1–9. ISSN: 0167-6377,1872-7468. DOI: 10.1016/0167-6377(96)00007-7. URL: https://doi.org/10.1016/0167-6377(96)00007-7.

[Bas+22] Amitabh Basu et al. "Complexity of branch-and-bound and cutting planes in mixed-integer optimization—II". In: *Combinatorica. An International Journal on Combinatorics and the Theory of Computing* 42 (2022), pp. 971–996. ISSN: 0209-9683,1439-6912. DOI: 10.1007/s00493-022-4884-7. URL: https://doi.org/10.1007/s00493-022-4884-7.

[BCC96] Egon Balas, Sebastián Ceria, and Gérard Cornuéjols. "Mixed 0-1 Programming by Lift-and-Project in a Branch-and-Cut Framework". In: *Manage. Sci.* 42 (Sept. 1996), pp. 1229–1246. ISSN: 0025-1909. DOI: 10.1287/mnsc.42.9.1229. URL: https://doi.org/10.1287/mnsc.42.9.1229.

[Ben+71] Michel Benichou et al. "Experiments in mixed-integer linear programming". In: *Mathematical Programming* 1 (1971), pp. 76–94. DOI: 10.1007/BF01584074. URL: https://doi.org/10.1007/BF01584074.

[Bes+21] Ksenia Bestuzheva et al. *The SCIP Optimization Suite 8.0*. Technical Report. Optimization Online, Dec. 2021. URL: http://www.optimization-online.org/DB_HTML/2021/12/8728.html.

[Bix+00] Robert E. Bixby et al. "MIP: Theory and Practice — Closing the Gap". In: *System Modelling and Optimization*. Ed. by M. J. D. Powell and S. Scholtes. Boston, MA: Springer US, 2000, pp. 19–49. ISBN: 978-0-387-35514-6.

[Cor08] Gérard Cornuéjols. "Valid inequalities for mixed integer linear programs". In: 112.1 (2008), pp. 3–44. ISSN: 0025-5610,1436-4646. DOI: 10.1007/s10107-006-0086-0. URL: https://doi.org/10.1007/s10107-006-0086-0.

[Dak65] Robert J. Dakin. "A tree-search algorithm for mixed integer programming problems". In: *The Computer Journal* 8.3 (Jan. 1965), pp. 250–255. ISSN: 0010-4620. DOI: 10.1093/comjnl/8.3.250. URL: https://doi.org/10.1093/comjnl/8.3.250.

[DP85] Rina Dechter and Judea Pearl. "Generalized best-first search strategies and the optimality of A*". In: *Journal of the Association for Computing Machinery* 32.3 (1985), pp. 505–536. ISSN: 0004-5411,1557-735X. DOI: 10.1145/3828.3830. URL: https://doi.org/10.1145/3828.3830.

[Füh11] Florian Führer. "Standortplanung unter Berücksichtigung von Tourenplanungsaspekten am Beispiel der Konsumgüterindrustrie". Diplomarbeit. Wien: Universität Wien, 2011.

[Gom58] Ralph E. Gomory. "Outline of an algorithm for integer solutions to linear programs". In: *Bulletin of the American Mathematical Society* 64 (1958), pp. 275–278. ISSN: 0002-9904. DOI: 10.1090/S0002-9904-1958-10224-4. URL: https://doi.org/10.1090/S0002-9904-1958-10224-4.

[Gom60] Ralph E. Gomory. "Solving linear programming problems in integers". In: *Proc. Sympos. Appl. Math., Vol. 10*. Amer. Math. Soc., Providence, RI, 1960, pp. 211–215.

[Jäk17] Christian Jäkle. "Simplex und das Branch-and-Bound-Verfahren mit Implementierung in Python". Bachelorarbeit. Konstanz: Universität Konstanz, 2017.

[Kar72] Richard M. Karp. "Reducibility among combinatorial problems". In: *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*. Plenum, New York, 1972, pp 85–103.

[KD15] Imdat Kara and Tusan Derya. "Formulations for Minimizing Tour Duration of the Traveling Salesman Problem with Time Windows". In: *Procedia Economics and Finance* 26 (2015). 4th World Conference on Business, Economics and Management (WCBEM-2015), pp. 1026–1034. ISSN: 2212-5671. DOI: https://doi.org/10.1016/S2212-5671(15)00926-0. URL: https://www.sciencedirect.com/science/article/pii/S2212567115009260.

[Kha79] Leonid Genrikhovich Khachiyan. "A polynomial algorithm in linear programming". In: *Dokl. Akad. Nauk SSSR* no. 5, (1979), pp. 1093–1096. ISSN: 0002-3264.

[Kum92] Vipin Kumar. "Algorithms for Constraint-Satisfaction Problems: A Survey". In: *AI Magazine* 13.1 (Mar. 1992), p. 32. DOI: 10.1609/aimag.v13i1.976. URL: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/976.

[Lad75] Richard E. Ladner. "On the Structure of Polynomial Time Reducibility". In: *J. ACM* 22.1 (Jan. 1975), pp. 155–171. ISSN: 0004-5411. DOI: 10.1145/321864.321877. URL: https://doi.org/10.1145/321864.321877.

[Law+85] Eugene L. Lawler et al., eds. *The traveling salesman problem*. Wiley-Interscience Series in Discrete Mathematics. A guided tour of combinatorial optimization, A Wiley-Interscience Publication. John Wiley & Sons, Ltd., Chichester, 1985, pp. x+465. ISBN: 0-471-90413-9.

[Lee+04] Ji Youn Lee et al. "Solving traveling salesman problems with DNA molecules encoding numerical values". In: *Biosystems* 78.1 (2004), pp. 39–47. ISSN: 0303-2647. DOI: https://doi.org/10.1016/j.biosystems.2004.06.005. URL: https://www.sciencedirect.com/science/article/pii/S0303264704001157.

[Lit+63] John D. C. Little et al. "An Algorithm for the Traveling Salesman Problem". In: *Operations Research* 11.6 (1963), pp. 972–989. URL: https://doi.org/10.1287/opre.11.6.972.

[LS99] Jeffrey T. Linderoth and Martin W. P. Savelsbergh. "A computational study of search strategies for mixed integer programming". In: vol. 11. 2. Combinatorial optimization and network flows. 1999, pp. 173–187. DOI: 10.1287/ijoc.11.2.173. URL: https://doi.org/10.1287/ijoc.11.2.173.

[Mor+16] David R. Morrison et al. "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning". In: *Discrete Optimization* 19 (2016), pp. 79–102. ISSN: 1572-5286. DOI: https://doi.org/10.1016/j.disopt.2016.01.005. URL: https://www.sciencedirect.com/science/article/pii/S1572528616000062.

[NW88] George L. Nemhauser and Laurence A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. A Wiley-Interscience Publication. John Wiley & Sons, Inc., New York, 1988, pp. xvi+763. ISBN: 0-471-82819-X. DOI: 10.1002/9781118627372. URL: https://doi.org/10.1002/9781118627372.

[NW90] George L. Nemhauser and Laurence A. Wolsey. "A recursive procedure to generate all cuts for 0-1 mixed integer programs". In: *Mathematical Programming* 46.3 (1990), pp. 379–390. ISSN: 0025-5610,1436-4646. DOI: 10.1007/BF01585752. URL: https://doi.org/10.1007/BF01585752.

[Pam22] Barbara Pampel. *Konzepte der Informatik*. Lecture Notes, University of Konstanz. 2022.

[PR91] Manfred Padberg and Giovanni Rinaldi. "A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems". In: *SIAM Review* 33.1 (1991), pp. 60–100. DOI: 10.1137/1033004. URL: https://doi.org/10.1137/1033004.

[RPK20] Isaac Michael Ross, Ronald J Proulx, and Mark Karpenko. "An optimal control theory for the traveling salesman problem and its variants". In: *arXiv preprint arXiv:2005.03186* (2020). DOI: https://doi.org/10.48550/arXiv.2005.03186.

[SA83] David J. Slate and Lawrence R. Atkin. "CHESS 4.5—The Northwestern University chess program". In: *Chess Skill in Man and Machine*. Ed. by Peter W. Frey. New York, NY: Springer New York, 1983, pp. 82–118. ISBN: 978-1-4612-5515-4. DOI: 10.1007/978-1-4612-5515-4_4. URL: https://doi.org/10.1007/978-1-4612-5515-4_4.

[SJ12] Edward C. Sewell and Sheldon H. Jacobson. "A branch, bound, and remember algorithm for the simple assembly line balancing problem". In: 24.3 (2012), pp. 433–442. ISSN: 1091-9856,1526-5528. DOI: 10.1287/ijoc.1110.0462. URL: https://doi.org/10.1287/ijoc.1110.0462.

[Van14] Robert J. Vanderbei. *Linear programming*. Fourth. Vol. 196. International Series in Operations Research & Management Science. Foundations and extensions. Springer, New York, 2014, pp. xxii+414. ISBN: 978-1-4614-7629-0; 978-1-4614-7630-6. DOI: 10.1007/978-1-4614-7630-6. URL: https://doi.org/10.1007/978-1-4614-7630-6.

[Wol98] Laurence A. Wolsey. *Integer programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. A Wiley-Interscience Publication. John Wiley & Sons, Inc., New York, 1998, pp. xx+264. ISBN: 0-471-28366-5.

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema

Exploring Advanced Techniques in Mixed-Integer Linear Programming:
A Study of the Branch-and-Cut Framework

selbstständig verfasst und keine anderen Hilfsmittel als die angegebenen verwendet habe. Die Stellen, die anderen Werken dem Wortlauf oder dem Sinne nach entnommen sind, habe ich in jedem einzelnen Fall durch Angaben der Quelle, auch der benutzten Sekundärliteratur, als Entlehnung kenntlich gemacht.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum	Thanh-Van Huynh