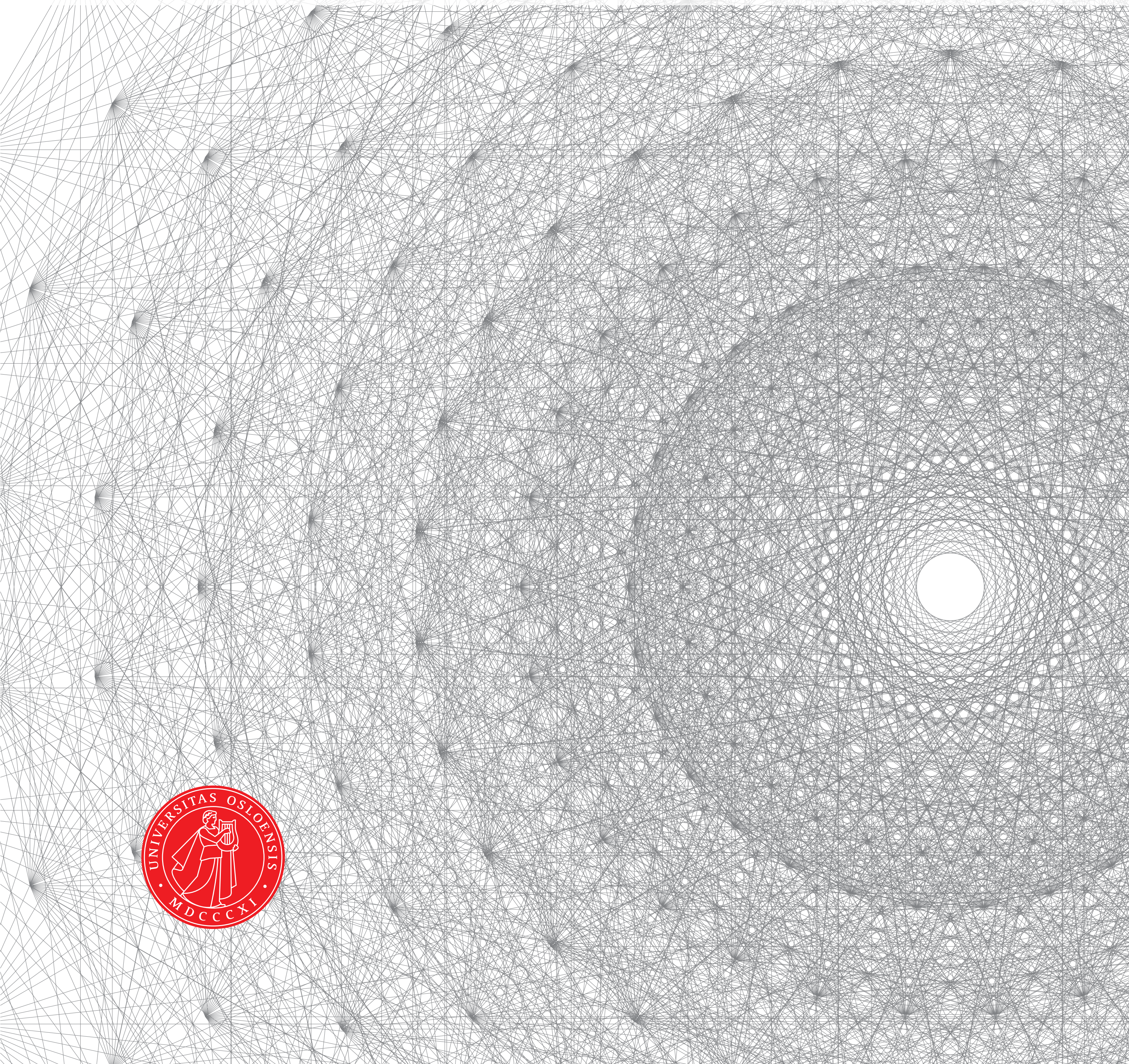
Mixed Integer Linear Programming Formulations For A Resource Constrained Project Scheduling Problem

Comparisons And New Approaches

Marie Lindland

Master's Thesis, Spring 2023



Abstract

The Resource Constrained Project Scheduling Problem (RCPSP) is an $\mathcal{NP}$-hard job scheduling problem. This thesis finds and compares mixed integer linear programming formulations for an extension to the standard RCPSP with the weighted tardiness objective. We experiment computationally on test instances inspired by real data to find a formulation that gives good and stable solver performance. A new Big-M formulation for disjunctive constraints is introduced. This ”permutation formulation” has limited purpose and needs further study, but nevertheless outperforms a time-indexed formulation on some large instances. We deduce cutting planes from the set of disjunctive, resource, precedence and tardiness constraints and inspect how adding them to a formulation impacts solver performance. A family of strong cutting planes is deduced from the precedence constraints, and for these we describe a separation algorithm and a separation heuristic. Furthermore, we prove that these inequalities along with general constraints suffice to describe the convex hull of a specific integral set. A Block Decomposition Heuristic is designed to find feasible solutions to RCPSP instances where the precedence relations give rise to chains. Overall, we find that a classic full time-indexed binary formulation is suitable for solving RCPSP instances of various sizes and constraint characteristics. This result also holds when our RCPSP is formulated as a rescheduling problem.

Acknowledgements

This thesis marks the end of a Master's Degree in Computational Science and Applied Mathematics with a main focus on optimization. I wish to thank my main supervisor Geir Dahl for his thorough guidance and valuable feedback over the last year. To my co-supervisors Carlo Mannino and Andreas Nakkerud I also owe a large gratitude. They have shown a continuous interest in my work and inspired me to investigate every idea that has come up along the way.

I wish to thank all the people in the Optimization Group at SINTEF for welcoming me and allowing me to work on the OptiPlan project. I am proud to be continuing my work with them full-time this upcoming fall. I wish to thank Pasquale Avella and Atle Riise for fruitful collaboration on the project. Thank you to all my fellow students in the study hall at the Department of Mathematics for making my time here pleasant. A special thanks to my friend Agnethe Hansen Kjensli for suggestions and feedback.

Finally, I wish to thank my husband Tobias Lindland. Along with the rest of my family, he has been supportive and made home a place to relax and reload between working hours.

Marie Lindland
May 2023

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Pick up your phone and look at your calendar. What do you see? Dense or spacious, detailed or vague; your calendar shows a schedule of your planned activities the upcoming days or weeks, and you probably aim to follow it in order to accomplish all tasks which lie ahead. Schedules plan, organize and visualize our time. For large companies and production facilities it is vital to find some way of coordinating the large set of operations that must be executed in order to meet deadlines and maintain overall productivity. Finding such a schedule grows in complexity when considerations like personnel utilization, safety regulations, economic costs etc. are taken into account.

A *scheduling problem* deals with allocating tasks to a time slot in a way that satisfies all constraints, and if such a schedule exists, finding one which minimizes or maximizes some objective. The study of deterministic sequencing and scheduling problems in the industrial engineering and operations research literature ranges back several decades. Since the sixties one major topic in this field has been to formulate and solve these optimization problems by means of mixed integer linear programming (MILP) methods [Pin22]. Scheduling is one of many practical applications of MILP. The desire to solve hard scheduling problems efficiently motivates both theoretical research as well as the development of exact or heuristic MILP based algorithms.

The standard Resource Constrained Project Scheduling Problem (RCPSP) is an $\mathcal{NP}$-hard scheduling problem for which there exist many variants in the literature [HB10; Węg+11]. In this thesis we study an extension to the standard RCPSP motivated by a real-life problem in the industry. For simplicity we refer to our problem as *the* RCPSP. Many RCPS problems are hard optimization problems for which a natural mathematical programming formulation of the set of feasible schedules may be either too large or weak for an exact branch-and-bound based solver to perform well. Clever techniques like reformulations, cutting planes or decompositions may improve the solver's ability to tackle a great variety of problem instances. However, there is a trade-off here; customizing and tweaking the formulation and solution algorithm too much may lead to poor performance when real data does not fit one's assumptions, while a very general algorithm may have unused potential with exploiting characteristics common for real instances of the problem. Through computational experiments this border can be analyzed to gain more knowledge on how a problem suitable algorithm and formulation should look.

Practical Applications at SINTEF

This thesis is written as a contribution to the ongoing OptiPlan project in the Optimization Group at SINTEF. The project aims at improving an existing decision support tool for preventive and corrective maintenance planners by including optimization. SINTEF presented us with an initial idea for a problem statement and a few classic MILP formulations for the different constraints. From here we decided by ourselves where to put in effort in order to contribute as much as possible with relevant results to the project.

Goal for the Thesis

We set the following as the goal for this thesis:

> *Find and compare MILP formulations for the set of feasible schedules to the RCPSP theoretically and computationally and inspect how they can be strengthened. Conclude on which formulation that ensures stable high solver performance on test instances for the RCPSP.*

Our emphasis will be on the modelling side as we find mathematical formulations of personal and theoretical interest. Some algorithmic aspects will be discussed, but this amount is limited as we will frequently be using the Gurobi Optimizer [Gur22] as MILP solver, as this will be used in the OptiPlan project.

Our work with this thesis has involved acquiring knowledge on scheduling and integer programming approaches for this. We contributed in defining the scheduling problem and its rescheduling version, for which we created test instances and implemented and tested different formulations. It was not until the last quarter of our work that we discovered our problem is actually an extension of the standard RCPSP. Had this been known at a previous stage we could to a larger extent have benefited from results in the RCPSP literature.

We wish to emphasize some work that we consider to be fully our own and possibly of interest in the RCPSP literature:

- To our knowledge this exact RCPSP is not studied in the literature.
- The adaption of all classic scheduling constraints to suit our problem.
- The permutation formulation in Section 5.2.
- The proof of Theorem 5.3.3, which expands a proposition given by Wolsey [Wol90] in some specific cases.
- The search for valid inequalities in Chapter 5 and all proofs in this chapter.
- The separation heuristic in Section 6.3.
- The Block Decomposition Heuristic in Appendix B.

Outline of Thesis

Chapter 2 explains preliminary mathematical theory.

Chapter 3 gives a brief introduction to scheduling and explains necessary terminology.

Chapter 4 introduces the RCPSP. We present different mathematical formulations for the set of feasible schedules and also introduce the rescheduling version of the RCPSP. Information is given on why our RCPSP is modeled as it is and how it relates to the standard RCPCP.

Chapter 5 inspects the different constraints in the RCPSP and deduces new valid inequalities which may work as cutting planes.

Chapter 6 explains how the valid inequalities found in the previous chapter can be examined computationally. A separation algorithm and separation heuristic is made for a set of precedence cuts.

Chapter 7 defines what we mean by solver performance and reports, compares and discusses how the solver performance is affected when different formulations for the RCPSP are used, as well as how cutting planes affect this performance.

Chapter 8 contains concluding remarks and highlights further work.

Appendix A gives relevant information on the models for the RCPSP and the test data instances.

Appendix B describes the Block Decomposition Heuristic which we designed to solve the RCPSP heuristically. We include this in an appendix as it is a side-track from the main focus in this thesis.

CHAPTER 2

Mathematical Background

In order to formulate and do computations on a real-world job scheduling problem we shall rely on the use of a mathematical framework. This chapter presents mathematical and computational theory which will lie as a foundation for the work done throughout this thesis. Some background material is assumed to be known by the reader, e.g., the basic notation for graphs, but for the sake of clarity and completeness we have tried to limit this amount. The chapter is structured as follows: Section 2.1 introduces central definitions and notation from convexity and polyhedral theory. Section 2.2 gives an overview of linear and integer programming, emphasizing formulations of integral sets. The presentation in these sections is based on the notes from Dahl [Dah10], Dahl and Mannino [DM12] and Pulleybank [Pul89]. Section 2.3 introduces the idea of cutting planes and presents known families of cutting planes for sets like the knapsack polytope. This presentation is based on the papers of Gabrel and Minoux [GM02], Kaparis and Letchford [KL08], Kardos et al. [Kar+21] and Wolsey [Wol90]. Finally, Section 2.4 gives a brief overview of computational complexity and relates this to linear and integer programming. Throughout the whole chapter the books on integer optimization from Nemhauser and Wolsey [WN88] and Schrijver [Sch86] have been of great inspiration.

2.1 Convexity and Polyhedral Theory

The theory of convexity is a necessary starting point for understanding polyhedra and linear programming. We try to narrow this huge field down to giving some core definitions. Throughout this section and the whole chapter we let $\mathbb{R}^n$ denote the set of real vectors of dimension n, where n is a natural number. We will use italic letters both for real column vectors $x \in \mathbb{R}^n$ and scalar values $t \in \mathbb{R}$.

Convexity

A subset S of $\mathbb{R}^n$ is called a *convex set* if for all $x_1, x_2 \in S$ and $\lambda \in [0, 1]$ the point $z = (1 - \lambda)x_1 + \lambda x_2$ also lies in S. Let $x_1, \ldots, x_t$ be a finite number of vectors in $\mathbb{R}^n$ and let $\lambda_1, \ldots, \lambda_t \geq 0$ be such that $\sum_{j=1}^{t} \lambda_j = 1$. The point $z = \sum_{j=1}^{t} \lambda_j x_j$ is called a *convex combination* of $x_1, \ldots, x_t$. For $t = 2$ the set of convex combinations of two points equals the line segment between them. It may be shown that $S \subseteq \mathbb{R}^n$ is a convex set if and only if it contains all convex combinations of its points. From this it follows that a convex set is closed under taking convex combinations. Not all sets that we will encounter are convex, so we will need some way of relating a non-convex set S to a convex set. For any set $S \subseteq \mathbb{R}^n$ we define the *convex hull* of S, denoted $\text{conv}(S)$, as the set of all convex combinations of points in S. This set is always convex, and it may be viewed as the smallest convex set containing S. If S is convex $\text{conv}(S) = S$, otherwise $S \subseteq \text{conv}(S)$. In general a complete description of $\text{conv}(S)$ may be hard or practically impossible to find even though S is known. Large parts of this thesis will be dedicated to the quest of finding better and better descriptions of precisely $\text{conv}(S)$. To see why, we need to define some more concepts.

Polytopes and polyhedra

We define a *polyhedron* $P \subseteq \mathbb{R}^n$ as the set of solutions to a linear system of inequalities in $\mathbb{R}^n$. The notation used is $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ for some real matrix $A \in \mathbb{R}^{m \times n}$ where m is finite and $b \in \mathbb{R}^m$. Any polyhedron is closed and convex. An inequality of the form $a^T x \leq \alpha$ is *valid* for P if $a^T x \leq \alpha$ holds for all $x \in P$. The inequality *supports* P if it is valid for P and there exists some $x' \in P$ such that $a^T x' = \alpha$. The set of points that satisfy such a valid, supporting inequality is called a *face* of P. By the term *proper face* we mean all faces F of P which are not empty or equal to P itself. Using duality (see Section 2.2) it can be shown that F is a face of P if and only if there exists some subsystem $A_0 x \leq b_0$ of $Ax \leq b$ such that $F = \{x \in P : A_0 x = b_0\}$. If F has dimension zero it is called an *extreme point* or a *vertex* of P, two definitions that coincide for polyhedra. The set of extreme points of P is denoted $\text{ext}(P)$. A face of dimension one is called an *edge*, and a face of dimension $\dim(P) - 1$ is called a *facet*.

Knowing the edges and vertices of the polyhedron P we can find another equivalent description of P which describes the inner points in P. Any edge of P must be either a line segment, a halfline or a line. If the edge is a halfline it is called an *extreme halfline*. Any extreme halfline may be written as $F = x_0 + \text{cone}(\{z\})$ where $x_0 \in P$ and z is the direction vector of F. Let S be some closed convex set in $\mathbb{R}^n$. By $\text{cone}(S)$ we mean the set of all non-negative combinations of points in S. Thus, $\text{cone}(\{z\})$ is the ray from the origin with direction vector z. If P does not contain any (infinite) lines we say that P is *pointed*. We now state the famous representation theorem for polyhedra proved in 1936 by T.S. Motzkin. It is also known as the main theorem for polyhedra. A proof may be found in [Dah10].

Theorem 2.1.1 (Motzkin's representation theorem)**.** *Any polyhedron $P \subseteq \mathbb{R}^n$ may be written as*

$$P = \text{conv}(V) + \text{cone}(Z)$$

for finite sets $V, Z \subseteq \mathbb{R}^n$. In particular, if P is pointed we may let $V = \text{ext}(P)$ and Z be the direction vectors of extreme halflines of P.

Conversely, if V and Z are finite sets in $\mathbb{R}^n$ the set $P = \text{conv}(V) + \text{cone}(Z)$ is a polyhedron, i.e., there exists a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $b \in \mathbb{R}^m$ such that

$$P = \{x \in \mathbb{R}^n : Ax \leq b\}.$$

This important theorem gives another way of representing a polyhedron that uses extreme points and edges instead of linear inequalities. From the theorem we also see that if Z is empty then P is a bounded polyhedron, also called a polytope. A *polytope* in $\mathbb{R}^n$ is the convex combination of some finite set of points. As it is bounded there exist some finite $l, u \in \mathbb{R}^n$ such that the box constraints $l \leq x \leq u$ hold. Polytopes will be of special interest in this thesis as the inclusion of any box constraints makes P a polytope, e.g., $0 \leq x \leq 1$.

Lastly, we consider polyhedra and polytopes with integral vertices. A polyhedron $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ is *rational* if all elements of A and b are rational numbers. The *integer hull* of a nonempty polyhedron $P \subseteq \mathbb{R}^n$ is defined as $P_I = \operatorname{conv}(P \cap \mathbb{Z}^n)$, that is, the convex hull of the integer points in P. It can be shown that if P is a rational polyhedron then P_I is also a polyhedron, see [Sch86]. If P is a polytope then P contains a finite number of integral points, so P_I is also a polytope. If $P = P_I$ we say that P is an *integral polyhedron*. All the vertices of an integral polyhedron are integral. When the vertices of P are not explicitly known it may be difficult to answer the question of whether P is an integral polyhedron or not. We shall later list some well-established techniques used for answering this question, as working with integral polyhedra is of special interest in many practical applications of linear programming.

2.2 Linear and Integer Programming

By *linear programming*, or simply LP, we mean the problem of minimizing or maximizing some linear function over a polyhedron. For this part we will consider maximization problems, so we write an LP problem on the form

$$\begin{array}{rl} \max & c^T x \\ \text{subject to} & Ax \leq b \\ & x \geq 0, \end{array} \tag{LP}$$

or equivalently $\max\{c^T x : x \in P\}$ over the polyhedron $P = \{x \in \mathbb{R}^n : Ax \leq b,\ x \geq 0\}$. Here, $c \in \mathbb{R}^n$ is the *coefficient vector*, $A \in \mathbb{R}^{m \times n}$ is the *coefficient matrix*, $x \in \mathbb{R}^n$ is the vector of *decision variables* whose values are to be determined and the function $x \mapsto c^T x$ is the *objective function*. Each inequality in P is called a *constraint*. The optimal value of (LP) is denoted z_{LP}. If z_{LP} is unbounded from above we define $z_{LP} = \infty$ and say that the maximization problem is *unbounded*. If $P = \emptyset$ the problem is *infeasible* and we set $z_{LP} = -\infty$. Note that if P is a nonempty polytope z_{LP} must be finite, but if P is not a polytope (LP) may be unbounded.

Some optimization problems may require x to be an integral vector. An *integer linear programming* problem is similar to an LP problem, but with the additional constraint that x should be an integral vector. When discussing such problems we will for convenience only use the wider term *integer programming* (IP), and we use the notation

$$\begin{array}{rl} \max & c^T x \\ \text{subject to} & Ax \leq b \\ & x \geq 0 \\ & x \text{ integer}. \end{array} \tag{IP}$$

IP problems arise in different forms. If we demand $x \in \{0,1\}^n$ the problem is a *binary programming* problem (BP). If $x = (y, z)$ where $y \in \mathbb{Z}^{n_1},\ z \in \mathbb{R}^{n_2}$ and $n_1 + n_2 = n$, the problem includes both real and integer variables and is called a *mixed integer programming* problem (MIP). In general, IP problems are much harder to solve than LP problems, and different algorithms are used. For a walk-through of the Simplex method used for solving LP problems and the branch-and-bound method used for solving IP problems we refer to [WN88].

Duality

If we let the above (LP) denote a *primal problem*, the *dual problem* is

$$\begin{array}{rl} \min & b^T y \\ \text{subject to} & y^T A \geq c \\ & y \geq 0, \end{array} \tag{D}$$

where $y \in \mathbb{R}^m$. This is an LP problem that is connected to the primal problem by the concept of *duality*. The idea is that every constraint in the primal problem corresponds to a variable in the dual problem, and vice versa. By the famous *weak duality theorem* any feasible solution to (D) gives a dual value which is an upper bound for the value of any primal feasible solution, i.e., $b^T y^* \geq c^T x^*$ for any primal feasible x^* and dual feasible y^*. From this result we can observe that if one of the problems is unbounded, the other problem must be infeasible. When both problems are feasible we may ask: How large is the gap between their optimal values? The next theorem, which is fundamental in duality theory, states that if this is the case the optimal values will coincide.

Theorem 2.2.1 (Strong duality). *If $x^* \in \mathbb{R}^n$ is a primal optimal solution, then the dual problem has an optimal solution $y^* \in \mathbb{R}^m$ such that*

$$b^T y^* = c^T x^*.$$

For a proof of this theorem using Farkas Lemma see [Sch86], or for a proof using the primal and dual Simplex algorithms see [WN88].

Formulations and relaxations

Let $X \subseteq \mathbb{Z}^n$ be the feasible region for the integer programming problem (IP). A *formulation* of X is a polyhedron $P \subseteq \mathbb{R}^n$ such that $X = P \cap \mathbb{Z}^n$. The integer points of P are then precisely X, and as $X \subseteq P$ it follows that $\max\{c^T x : x \in X\} \leq \max\{c^T x : x \in P\}$. Usually one wishes to find some P which makes the gap between these values as small as possible, ideally zero. Let P_1 and P_2 be formulations of X. We say that P_1 is a *stronger* or *tighter* formulation than P_2 if $P_1 \subseteq P_2$, which is equivalent to $\max\{c^T x : x \in P_1\} \leq \max\{c^T x : x \in P_2\}$ for all $c \in \mathbb{R}^n$. Informally, we will say that a formulation P of X is *strong* if it is stronger than most other known valid formulations of X, or in the opposite case that P is *weak*. The strongest possible formulation of X is obtained when P is an integral formulation, i.e., $P = \text{conv}(X)$. See Figure 2.1. Here, the solid black dots are the feasible region X, while the white circles are infeasible integer points. The light grey area shows

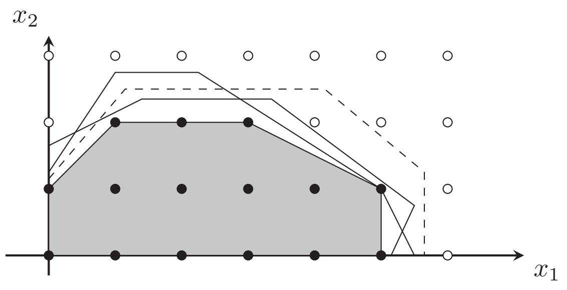


Figure 2.1: Formulations for an integer program

conv(X). Two other valid formulations are given by the solid black polytopes, but the dashed polytope is not a formulation as it contains an infeasible point.

Let Q be a polyhedron in $\mathbb{R}^p$ for some $p > n$. We say that Q is an *extended formulation* of X if its projection onto $\mathbb{R}^n$ is a formulation of X. That is, if the polyhedron $\text{proj}_x(Q) = \{x \in \mathbb{R}^n : \exists\, z \in \mathbb{R}^{p-n} \text{ such that } (x, z) \in Q\}$ is such that $\text{proj}_x(Q) \cap \mathbb{Z}^n = X$. In some cases an extended formulation may have constraints on a familiar form or be polynomial in its number of constraints while the original formulation is exponential. In such cases it may be beneficial to work with the extended formulation of X.

A *relaxation* of (IP) is any problem (R) on the from $\max\{w^T x : x \in T\}$ where $X \subseteq T$ and $c^T x \leq w^T x$ for all $x \in X$. We will further on consider relaxations with $w \equiv c$, so by a relaxation of (IP) we simply mean solving the same problem over a larger set. Relaxations come in many forms. The *LP relaxation* of an IP problem is obtained by removing the integrality constraints on x and possibly adding the corresponding linear constraints. Thus, the LP relaxation of (IP) is simply (LP). For $(0, 1)$-programming the constraint $x_j \in \{0, 1\}$ is relaxed to $0 \leq x_j \leq 1$ for each $j = 1, \ldots, n$.

When solving IP problems we lean on the use of clever relaxations and strong formulations. In the next section we shall see how formulations can be strengthened by adding so-called cutting planes. But first, let us consider the problem of deciding whether a given formulation P of X is integral or not.

Total unimodularity and integral polyhedra

Some polyhedra $P \subseteq \mathbb{R}^n$ are known to be integral. Let $P = \{x \in \mathbb{R}^n : Ax \leq b,\ x \geq 0\}$ be a polyhedron where $A \in \mathbb{R}^{m \times n},\ b \in \mathbb{R}^m$. The matrix A is *totally unimodular* (TU) if every subdeterminant of A is -1, 0 or 1.

Proposition 2.2.2. *Let $P = \{x \in \mathbb{R}^n : Ax \leq b,\ x \geq 0\}$ be a nonempty polyhedron. If A is TU then P is integral for all $b \in \mathbb{Z}^m$.*

We omit the proof here, which may be found in most books on integer programming. The proposition can actually be formulated as an ”if and only if” statement, see [WN88]. The result also holds if constraints on the form $l \leq x \leq u$ are added to P for $l, h \in \mathbb{Z}^n$. Even though most coefficient matrices for IP problems are not TU there exist several types of matrices which are known to be TU, like the incidence matrices for bipartite graphs and directed graphs.

Assume that P is a formulation of some integral set $X \subseteq \mathbb{Z}^n$. Knowing whether P is integral can be of great interest when solving an IP problem, see Section 2.4. If P is integral any optimal solution to the LP relaxation of (IP) will be integral. We therefore list some ways of showing that P is an integral polyhedron, i.e., $P = \text{conv}(X)$. If this is the case we say that P gives a *complete linear description* of $\text{conv}(X)$. The list is taken from [PW06].

Proposition 2.2.3. *Let $X \subset \mathbb{Z}^n$ be non-empty and let $P = \{x \in \mathbb{R}^n : Ax \leq b,\ x \geq 0\}$ be a formulation for X. If one of the following holds $P = \text{conv}(X)$.*

- *All extreme points of P are integral.*
- *All points in P with $x \notin \mathbb{Z}^n$ are not extreme points of P.*
- *The primal problem $\max\{c^T x : x \in P\}$ has an optimal solution $x^* \in X$ for all $c \in \mathbb{R}^n$.*
- *For all $c \in \mathbb{R}^n$ the dual problem $\min\{b^T y : y^T A \geq c,\ y \geq 0\}$ has a feasible solution y^* such that $b^T y^* = c^T x^*$ for some $x^* \in X$.*
- *There exists some extended formulation Q of X such that $\text{proj}_x(Q) = P$ and Q is integral.*
- *A is TU and $b \in \mathbb{Z}^m$.*

2.3 Cutting Planes

When solving IP problems we work under the theoretical and computational assumption that optimizing over a strong formulation P of the set $X \subseteq \mathbb{Z}^n$ is better than using a weak formulation. We thus seek to find constraints that lie as close to the perfect formulation $\text{conv}(X)$ as possible. Recall that if $a^T x \leq \alpha$ is satisfied by every $x \in \text{conv}(X)$ this inequality is valid for $\text{conv}(X)$. Informally, we will say that a valid inequality for $\text{conv}(X)$ is *strong* if adding it to P makes P a strong formulation of X. Given two valid inequalities for $\text{conv}(X)$ we say that $a_1^T x \leq \alpha_1$ is *stronger* than $a_2^T x \leq \alpha_2$ if $P \cap \{x \in \mathbb{R}^n : a_1^T x \leq \alpha_1\}$ is a stronger formulation than $P \cap \{x \in \mathbb{R}^n : a_2^T x \leq \alpha_2\}$. The strongest valid inequalities for $\text{conv}(X)$ are the ones that define facets of $\text{conv}(X)$. For well-studied linear programs families of strong valid inequalities for $\text{conv}(X)$ may already be known. However, the number of such inequalities can be exponential. Adding all of them to the formulation a priori to solving the problem is usually inefficient as it may make it very time-consuming to solve the LP relaxation. As an alternative, strong valid inequalities may be added as so-called *cutting planes* during the solution process.

Let $x^* \in P$ be a point in a formulation P of X such that $x^* \notin \text{conv}(X)$. Any valid inequality $a^T x \leq \alpha$ for $\text{conv}(X)$ that has $a^T x^* > \alpha$ *separates* or *cuts off* x^* from $\text{conv}(X)$. Given $x^* \in P$ the problem of finding such a separating inequality or deciding that no such exists is called the *separation problem.* A *separation algorithm* is some algorithm that solves the separation problem and returns one or more violated inequalities if some are found. Adding one violated inequality, also called a *cutting plane*, to the formulation P will strengthen the formulation, see Figure 2.2 for an illustration. If we have an inequality $a^T x \leq \alpha$

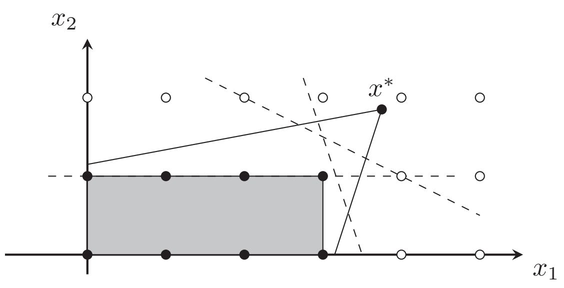


Figure 2.2: Cutting planes

valid for X and we want to show it is a cutting plane, it suffices to find some fractional point $x^* \in P$ which is violated by the inequality.

We shall now look into some well-established families of cutting planes for constraints on a general form. A *knapsack constraint* is any constraint of the form

$$\sum_{j \in N} a_j x_j \leq b,$$

where $N = \{1, \ldots, n\},\ b \in \mathbb{R}$ and $a_j \geq 0,\ j \in N$. This makes knapsack constraints a special case of normal linear programming constraints where the coefficients are all non-negative. Let $P = \{x \in \mathbb{R}^n : \sum_{j \in N} a_j x_j \leq b,\ 0 \leq x \leq 1\},\ X = P \cap \{0,1\}^n$. Then $\text{conv}(X)$ is called the *0-1 single knapsack polytope.*

Cover inequalities and lifted inequalities

Any subset $C \subseteq N$ such that $\sum_{j \in C} a_j x_j > b$ is called a *cover.* If C is a cover then the *cover inequality* (CI)

$$\sum_{j \in C} x_j \leq |C| - 1$$

is valid for $\text{conv}(X)$. A cover C is *minimal* if the set $C \setminus \{j\}$ is not a cover for all $j \in C$. The strongest CIs arise when C is minimal, in fact, the CI is facet-defining for $\text{conv}(X) \cap \{x \in \mathbb{R}^n : x_j = 0,\ j \in N \setminus C\}$ if and only if C is a minimal cover, see e.g., [CCZ14]. In general, we must include variables from $N \setminus C$ in the inequality for the CI to be facet-defining for $\text{conv}(X)$. One way to strengthen the CI is to consider its extension. Define the *extension* of C as $E(C) = \{j \in N \setminus C : a_j \geq \max_{i \in C} a_i\}$. The *extended cover inequality* (ECI)

$$\sum_{j \in E(C)} x_j \leq |C| - 1$$

is then valid for $\text{conv}(X)$. If $E(C) \setminus C \neq \emptyset$ the ECI is stronger than the corresponding CI as the left side includes more variables. Extending the CI is a special case of a more general procedure called *lifting.* In general, the *lifted cover inequalities* (LCIs) take the form

$$\sum_{j \in C} x_j + \sum_{j \in N \setminus C} \alpha_j x_j \leq |C| - 1,$$

where $0 \leq \alpha_j \leq |C| - 1,\ j \in N \setminus C$. The α_j coefficients are called the *lifting coefficients*. There exist several approaches and algorithms for finding these lifting coefficients, and LCIs can also be expressed in more general forms. We refer to [KL08] for more reading on this. The perhaps most known procedure finds the α_j coefficients sequentially by so-called *up-lifting*. Assume $\dim(X) = n$, let C be a minimal cover and choose some ordering $j_1, \ldots, j_l$ of the indices in $N \setminus C$. Let $C_0 = C,\ C_h = C_{h-1} \cup \{j_h\}$ for $h = 1, \ldots, l$. To find α_{j_h} we ask: Given $x_{j_h} = 1$ and $x_{j_k} = 0$ for all $k > h$, what is the largest value $\sum_{j \in C_{h-1}} \alpha_j x_j$ can have such that all valid inequalities for X still hold? More formally, if $\sum_{j \in C} \alpha_j x_j \leq \beta$ is facet-defining for $\text{conv}(X) \cap \{x \in \mathbb{R}^n : x_j = 0,\ j \in N \setminus C\}$, then $\sum_{j \in N} \alpha_j x_j \leq \beta$ is facet-defining for $\text{conv}(X)$ where $\alpha_{j_h} = \beta - \max\{\sum_{j \in C_{h-1}} \alpha_j x_j : x \in X,\ x_{j_h} = 1,\ x_{j_k} = 0,\ k > h\}$ for $h = 1, \ldots, l$. We note that sequential lifting may produce different LCIs for different orderings of the variables in $N \setminus C$, and that other procedures not depending on solving several small knapsack problems do exist.

GUB inequalities

A *generalized upper bound* (GUB) inequality is of the form

$$\sum_{j \in J} x_j \leq U$$

for some subset $J \subseteq N$ and upper bound $U \in \mathbb{R}$. Constraints on this form limit the number of variables x_j for $j \in J$ that can be equal to one. We will also use this term for inequalities of the form $\sum_{j \in J} x_j = U$ for some $U \in \mathbb{N}$.

In real-life binary programs GUB constraints may arise together with knapsack constraints. We shall consider such a problem introduced in [Wol90]. In this variant of the knapsack problem the variables are split into two groups, $i = 1, 2$. Let $x \in \mathbb{R}^n_+$ where $n = |N|,\ N = N_1 \cup N_2,\ N_1 \cap N_2 = \emptyset$. Let the coefficients be $a_j > 0$ for $j \in N$. For each $i = 1, 2$ we partition N_i into $|I_i|$ non-overlapping sets $S_s,\ s \in I_i$ such that $\bigcup_{s \in I_i} S_s = N_i$ and $I_1 \cap I_2 = \emptyset$. Let $X = P \cap \{0, 1\}^n$ where $P \subseteq \mathbb{R}^n_+$ is the following polytope:

$$\begin{aligned}
&\sum_{j \in N_1} a_j x_j - \sum_{j \in N_2} a_j x_j \leq b, \\
&\sum_{j \in S_s} x_j \leq 1 && s \in I_1 \cup I_2, \\
&x \in \mathbb{R}^n_+.
\end{aligned}$$

We note that, although there is a negative sign on the left side, the first above inequality remains a knapsack constraint as we can set $b = b' - \sum_{j \in N_2} a_j$ in the knapsack constraint $\sum_{j \in N_1} a_j x_j + \sum_{j \in N_2} a_j y_j \leq b'$ where $y_j = 1 - x_j$. Wolsey [Wol90] defines a *GUB cover* for X as the set $C = C_1 \cup C_2$ where

(i) $C_i \subseteq N_i$ for $i = 1, 2$,
(ii) $|C_i \cap S_s| \leq 1$ for $i = 1, 2,\ s \in I_i$,
(iii) $\sum_{j \in C_1} a_j - \sum_{j \in C_2} a_j > b$.

Given a cover C, define the set $I_i^+ = \{s \in I_i : C_i \cap S_s \neq \emptyset\}$ for $i = 1, 2$, and the set

$$S_s^+ = \begin{cases} \{j \in S_s : a_j \geq a_l \text{ for } l \in C_1 \cap S_s\} & \text{for } s \in I_1^+, \\ \{j \in S_s : a_j \leq a_l \text{ for } l \in C_2 \cap S_s\} & \text{for } s \in I_2^+. \end{cases}$$

Using these sets we can find new valid inequalities for conv(X).

Proposition 2.3.1 ([Wol90])**.** *The inequality*

$$\sum_{s \in I_1^+} \sum_{j \in S_s^+} x_j \leq |C_1| - 1 + \sum_{s \in I_2^+} \sum_{j \notin S_s^+} x_j + \sum_{s \in I_2 \setminus I_2^+} \sum_{j \in S_s} x_j$$

is valid for X.

The proof may be found in [Wol90]. Knowing that these inequalities are valid for X we may use them in concrete cases where the set X is more specified in hope to obtain a stronger formulation of X. For some polyhedra these inequalities are known to define facets, and in some cases they may even give a complete linear description of conv(X). In Section 5.3 we shall look at such a specific job scheduling problem and prove that the inequalities suffice to describe conv(X) in that case.

Clique inequalities and CG-cuts

The final type of constraints we shall consider are strongly related to graph theory, and we shall introduce their relevance with a well-studied example. Let $G = (V, E)$ be a simple undirected graph where V is a finite set of vertices with $|V| = n$ and E is the set of edges of G. A set $U \subseteq V$ in which no pair of vertices are connected by an edge is called a *stable set.* Let $P = \{x \in \mathbb{R}^n : 0 \leq x \leq 1,\ x_i + x_j \leq 1 \text{ for all } \{i, j\} \in E\}$ and $X = P \cap \mathbb{Z}^n$. Then conv(X) is called the *stable set polytope.* It can be shown that $P =$ conv(X) if and only if G is bipartite. For other graphs P can be strengthened by adding cutting planes. Any subset $\Delta \subseteq V$ such that all pairs of vertices in Δ are connected by an edge is called a *clique.* The clique is *maximal* if $\Delta \cup \{v\}$ is not a clique for any $v \in V \setminus \Delta$. If Δ has k elements we say that Δ is a *k-clique* or that it has size k. A *clique constraint* is of the form

$$\sum_{j \in \Delta} x_j \leq 1,$$

for some clique $\Delta \subseteq V$. We shall show that these inequalities are cutting planes for P. The inequalities of the form $x_i + x_j \leq 1$ in P are called *edge constraints.* Consider a clique $\Delta = \{i, j, k\}$ in G. If we add the three edge constraints for the pairs in Δ together this gives $2(x_i + x_j + x_k) \leq 3$, which when divided by 2 gives $x_i + x_j + x_k \leq \frac{3}{2}$. As x is a binary vector this inequality is strengthened by $x_i + x_j + x_k \leq 1$, which is a valid cutting plane as it, e.g., cuts of fractional points where $x_i = x_j = x_k = \frac{1}{2}$. This procedure of summation, division and rounding can be repeated when Δ is of larger size. The inequalities returned will be of increasing strength.

The above procedure is an example of the use of the *Chvátal-Gomory procedure* (CG). Consider the general IP program (IP) and let X denote the feasible region. Let $\lambda \in [0,1]^m$. As

$$Ax \leq b$$

is valid for X, also

$$\lambda^T Ax \leq \lambda^T b$$

is valid. For $x \geq 0$ the left side may be rounded down, so

$$\lfloor \lambda^T A \rfloor x \leq \lambda^T b$$

holds. Finally, as x is integral, the stronger inequality

$$\lfloor \lambda^T A \rfloor x \leq \lfloor \lambda^T b \rfloor$$

must also be valid. This cut is called a *Chvátal-Gomory cut*. Applying this procedure once gives an inequality in the rank 1 Chvátal closure of the edge constraints [Chv73]. Another round gives rank 2, and the final clique inequality $\sum_{j \in \Delta} x_j \leq 1$ has Chvátal rank $(k-2)$ where $k = |\Delta|$. For a clique Δ this gives the strongest clique cut.

2.4 Computational Complexity Theory

When working with an optimization problem, or any type of computational problem, a central question is: How hard is this problem to solve? The theory of *computational complexity* builds an extensive framework for answering questions like this, and we shall give a very brief overview of some important concepts and results. We mainly follow the presentation given in [Ner19]. Our overview will have an applied algorithmic point of view practical for our study. For formal and precise definitions related to Turing machines, languages and decidability we refer to [Mar11; Vit12].

Time complexity

A *decision problem* P is any problem where the answer is either YES or NO. As an example the decision version of LP is: Given A, b, c and U, does there exist some x such that $Ax \leq b$ and $c^T x \geq U$? When these variables have been assigned values we get an *instance* of the problem. An *algorithm* A for P is some sequence of operations on a machine which given an instance of P as *input* solves the problem in finite time, i.e., A returns YES for instances of P that should return YES and NO for instances that should return NO. Usually one knows more than one algorithm for solving a problem, but their performance may vary. To determine some measure of an algorithm's efficiency we will consider the so-called *time complexity* of the algorithm. By this term we mean the number of elementary operations (summation, assigning values to variables etc.) required by the algorithm in order to return the correct answer. Actually, the exact such number of operations is not of great interest as it makes comparison troublesome. We instead measure the time complexity $t(n)$ as a function of the size n of the input.

The size of the input to an algorithm can be measured in different ways. When using a graph algorithm for $G = (V, E)$ we are interested in the number of nodes $|V|$ and edges $|E|$, while for a sorting algorithm we are interested in the number k of elements in the array. For LP- and IP problems with rational data we will refer to the size of the input as the number of bits needed to represent the input data $A,\ b,\ c$ on a computer.

Let now A be an algorithm for some decision problem P. Some instances of P of size n may be solved by A in very few steps, while harder instances of the same size may require a larger number of operations to be solved. An example is the problem of deciding whether the character 'B' is included in some word. If A reads the input word from left to right the instance "BANANA" is solved at once while "ADVERB" is a worst-case example. When defining $t(n)$ for an algorithm A we will always consider *worst-case* input of size n, so that $t(n)$ measures the maximum number of operations needed by A among all the input of size n. Another approach is to look at the average number of steps which is done in *average-case analysis*. The Simplex method for solving LP problems is an example of an algorithm that needs an exponential number of operations in worst-case, but runs very fast in average-case [Sch86].

Closely related to time complexity is the concept of *big-O notation*. We say that a function $t(n)$ is $\mathcal{O}(g(n))$ if there exist some constants C and N such that $t(n) \leq Cg(n)$ for all $n \geq N$. For example, if $t(n) = n^3 + 2n^2 + \log(n)$ then $t(n) = \mathcal{O}(n^k)$ for $k \geq 3$. Strictly speaking $\mathcal{O}(g(n))$ is a set of functions, but we use the notation $t(n) = \mathcal{O}(g(n))$ as is done in most literature. We are interested in the "lowest possible" $g(n)$, so in the example we prefer $\mathcal{O}(n^3)$. Some advantages of using big-O notation are the focus on the dominating term in $t(n)$, the ignorance of the remaining ones and that we get an upper bound for how fast $t(n)$ grows. When an algorithm has time complexity $t(n) = \mathcal{O}(g(n))$ we say that it has *running time* $\mathcal{O}(g(n))$ or simply that the algorithm is $\mathcal{O}(g(n))$. Some examples are constant algorithms $\mathcal{O}(1)$, linear algorithms $\mathcal{O}(n)$ and exponential algorithms $\mathcal{O}(2^n)$.

The classes $\mathcal{P}$ and $\mathcal{NP}$

When a decision problem has an algorithm which is $\mathcal{O}(n^k)$ for some $k \in \mathbb{N}$ we say that the problem is solvable in *polynomial time*. The class of problems that have a polynomial algorithm is denoted $\mathcal{P}$. In 1979 Khachiyan [Kha79] developed the polynomial Ellipsoid algorithm for solving LP problems, thereby showing that LP is a problem in $\mathcal{P}$. We refer to problems in $\mathcal{P}$ as *simple* or *tractable* problems, even though n and k in theory may be very big.

The complexity class $\mathcal{NP}$ consists of all problems whose solution can be verified in polynomial time. That is, if we know x is a solution to the problem, a verifying algorithm should be able to check this in at most polynomial time. Several famous problems are known to be in $\mathcal{NP}$, like the Vertex Cover problem and 3-SAT. We do not know whether there may exist polynomial algorithms for solving these problems as it remains an open question whether $\mathcal{P} = \mathcal{NP}$ or not. The letters $\mathcal{NP}$ stand for "nondeterministic polynomial", a definition related to Turing machines that must not be confused with "nonpolynomial".

A problem is *$\mathcal{NP}$-hard* if it is at least as hard to solve as the hardest problems in $\mathcal{NP}$. A problem is *$\mathcal{NP}$-complete* if it is $\mathcal{NP}$-hard and lies in $\mathcal{NP}$. We know that $\mathcal{NP}$-complete problems exist due to SAT and the Bounded Halting problem; two problems shown to be $\mathcal{NP}$-complete from the formal definition. The decision version of IP is in general $\mathcal{NP}$-complete as problems that are known to be $\mathcal{NP}$-complete can be formulated as an integer program. The optimization version of integer programming, (IP), is in general $\mathcal{NP}$-hard, although some special cases are known to be solvable in polynomial time. This is the case when the feasible region is known to be an integral polyhedron, since then one can solve the LP relaxation (LP) which we know is a problem in $\mathcal{P}$. For large real-life optimization problems this may result in a huge reduction in the running time needed to solve the problem.

CHAPTER 3

Scheduling Problems

This thesis is devoted to finding and comparing mathematical formulations to the set of feasible solutions to a Resource Constrained Project Scheduling Problem (RCPSP). The core challenge in RCPSP is, precisely, scheduling. In one of the classical books on the topic the following definition of scheduling is given [Pin22]:

> Scheduling is a decision-making process that is used on a regular basis in many manufacturing and services industries. It deals with the allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives. (Michael L. Pinedo)

From scheduling preventive maintenance in a production facility to assigning arriving planes gateways at an airport, via on-line train dispatching and classroom timetabling – scheduling problems cover a wide range of problems. This thesis only considers scheduling problems on an operational level, assuming that all strategic and tactical planning has been made on a higher level in the industry. This eases the decision-making process and problem modelling by, e.g., removing the need of choosing which operations that should be scheduled. In the industry loads of information will flow between the scheduling and planning levels, making the processes heavily dependent and increasing the need for day-to-day decision-making. The flow chart in Figure 3.1 from [Pin22] illustrates this for a manufacturing system. Our emphasis will lie on the decisions made within the dashed rectangle.

The purpose of this chapter is to give information on scheduling problems in general and to introduce necessary definitions. The outline of the chapter is as follows: In Section 3.1 we present the classic scheduling problem. Central definitions and constraints will be presented that are needed to understand the RCPSP introduced in Chapter 4. Section 3.2 gives an introduction to different formulations and variables used when modelling scheduling problems, like time-indexed formulations and Big-M variables. We will use these formulations when modelling the main problem in Chapter 4.

3.1 The World of Job Scheduling Problems

The classic job scheduling problem deals with allocating a set of n jobs to m machines, where a machine is something that can process one job at a time. From this simple starting point there grows a whole world of scheduling

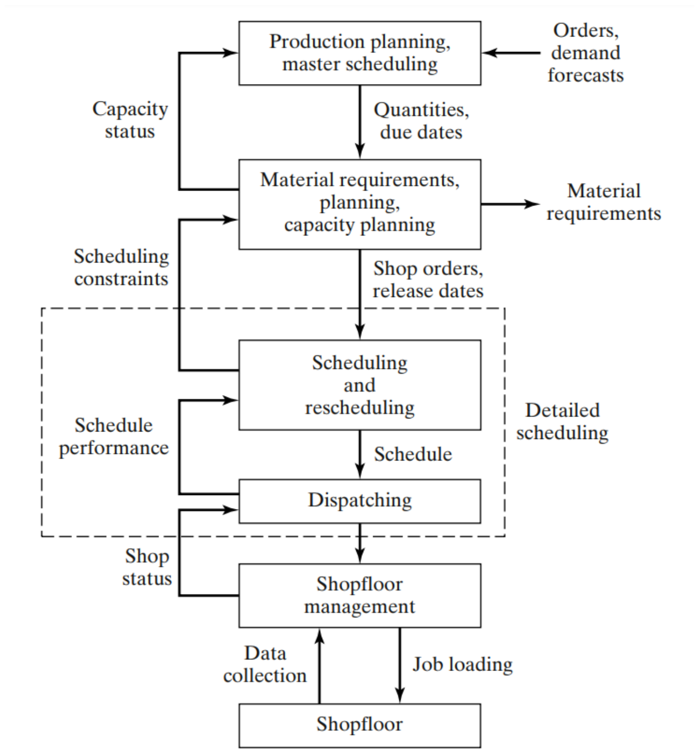


Figure 3.1: Flow diagram in a manufacturing system [Pin22]

problems that differ from another in machine environment (α) processing characteristics and constraints (β) and objective (γ). With this notation a scheduling problem is described by a triplet $\alpha \mid \beta \mid \gamma$. The perhaps most intuitive scheduling problem $1 \mid\mid C_{\max}$ consists of one single machine with the objective of minimizing the completion time of the job that finishes last, the *makespan*. Other problems may consider machines that run in parallel with different speeds, introduce constraints on the feasible orderings of the jobs or consider more complicated objectives. Some problems seem very similar, but have different complexity. On a single machine the problem of minimizing total weighted completion times $1 \mid\mid \sum w_j C_j$ is easily solved by comparing the ratios between the operations' weight and duration [Smi56], while the problem of minimizing the total weighted tardiness $1 \mid\mid \sum w_j T_j$ is $\mathcal{NP}$-hard [Law77]. Overall, the great variety in scheduling problems alongside their practical value and varying complexity makes scheduling a field of comprehensive study through many years. For an overview of well-studied variants of the scheduling problem and their mathematical notation we refer to [Pin22, Chapter 2].

There are variants of scheduling problems that are not fully covered by the simple above framework, such as personnel scheduling, production planning or resource constrained scheduling [Pin22]. The RCPSP studied in this thesis is strongly related to the latter one. Below we will introduce definitions, objectives and constraints relevant for both the RCPSP and scheduling problems in general. Our source of inspiration is a survey on the standard RCPSP and its extensions by Węglarz et.al. [Węg+11].

Definitions

The majority of scheduling problems gives rise to many definitions. The following ones will be necessary to understand the RCPSP introduced in Chapter 4.

Operations: The individual activities that are to be scheduled are called *operations* or *tasks*. A collection of operations is called a *work order*.

Schedule: A *schedule* contains start times for the processing of each operation. Each schedule is assumed to be finite in time, e.g. one week or two months ahead. This time is called the *planning horizon*. The schedule may be discretized into time steps of one hour or one day. A *scheduling problem* is the problem of finding a schedule that is feasible, i.e., it satisfies all constraints, and which maximizes or minimizes a given objective function.

Duration and due date: Each operation has a predetermined *duration*. This denotes the operation's processing time from start to finish. Some problems allow for *preemption*, i.e., temporarily interrupting the performance of a task. Operations may have a *due date*. An operation's *lateness* measures the time between its completion time and due date. The *tardiness* or *delay* is zero if the operation finishes before its due date, otherwise it equals the lateness. Usually, delay is penalized with a high cost. If an operation has a strict due date this is called a *deadline*. Some operations may have a *release date*, which is a specified earliest start date for the operation.

Resources: We use this term almost equivalent to a machine in the above sense. A *resource* is something that is required by an operation in order to be processed. Examples are human resources, a railway segment, a manufacturing machine or gasoline. A resource *unit* is one amount of a resource, a resource *type* is a set of resource units that can perform the same tasks and a resource *category* is a set of resource types that share the same characteristics. The three basic resource categories are *renewable resources* with constant capacity, *nonrenewable resources* with limitations on the total consumption and *doubly constrained* resources which have both limited capacity and consumption. Newer categories have been introduced and studied over the last years. One of these is *partially renewable resources* where the capacity varies over different time periods. Operations have specified *resource demands* for each available resource type, given as integers if the resources come in discrete amounts. An operation's duration may be resource-dependent, leading to a shorter duration if more resource units process the operation. We then say that en operation has different *modes*, and one mode must be chosen for each operation.

Objectives

The common approach for optimizing schedules is to solve a minimization problem. Some classic single objectives are to minimize the makespan, the total weighted lateness, the total weighted tardiness, the total weighted completion times, the maximum lateness or the number of tardy jobs. The weights give some measure of an operation's priority or importance compared to other operations. As mentioned above, the choice of objective may affect the computational complexity of the problem and thereby the state-of-the-art methods used to solve it.

All the objectives mentioned above are time-based. Other common objectives concern the resources. The usage of each resource unit may be associated with a cost, and the objective could be to minimize the resource costs needed to obtain a feasible schedule. These two objective types are usually in conflict: In order to process many operations simultaneously one needs a high amount of resource units at each time, but with a low resource usage the operations' resource demand causes late completion times.

Practical applications of scheduling will lead to objectives that also consider other aspects than resources and time, like financial or environmental objectives. When problems are multi-modal or include decision making on a strategic or tactical level this can lead to a complicated objective function defined in a high-dimensional space. When selecting an objective a choice must be made by weighing the generality and simplicity of the model on one side against the level of detail in the objective on the other side.

Constraints

Constraints limit the set of feasible schedules to a scheduling problem. In addition to general constraints that, e.g., ensure each operation starts exactly once, other classes of constraints may be needed to remove infeasible solutions.

Precedence constraints An operation may have prerequisite operations. In the work order of changing exterior windows on a building the operation of building the scaffold must be completed before the operation of removing the current windows can begin. Precedence relations are one of the most common constraint types, frequently appearing in RCPS problems.

A minimal (maximal) *time-lag* specifies the minimal (maximal) time that should be between the processing of two operations. Time-lags can be expressed as start-to-start, start-to-finish, finish-to-start or finish-to-finish relations. The classic precedence constraint is a finish-to-start relation with zero minimal time-lag or, equivalently, a start-to-start relation with minimal time-lag of the predecessor's duration. With such constraints between consecutive pairs of operations, the operations can easily be represented as a simple directed activity-to-node graph with an arc between the predecessor and successor in every precedence relation. See Figure 3.2 for an illustration. This graph is called an open *chain*. On the right we show a Gantt chart representation of a schedule satisfying these constraints. An operation is visualized by a rectangle which starts at the operation's starting time and ends when the operation is completed. The y-axis is needed to illustrate overlapping operations.

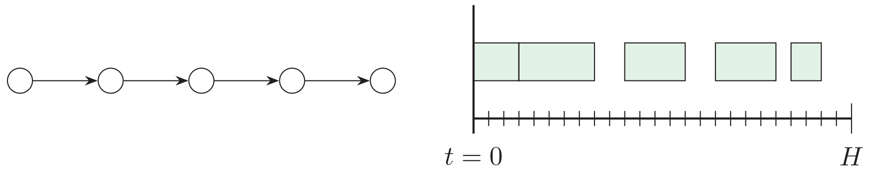


Figure 3.2: Precedence constraints

Scheduling problems with precedence constraints have varying computational complexity. For a further description of the different precedence relations that exist and a list of the complexity of different machine scheduling problems with precedence constraints we refer to the survey of Bellenguez-Morineau and Prot [PB18]. One classic result was shown by Lawler in 1978 [Law78]. He showed that when the precedence relations give rise to a series parallel activity-to-node graph the problem $1 \mid sp - graph \mid \sum w_j C_j$ is polynomially solvable. Chains are a simple type of series parallel graphs. In Section 5.3 we inspect if we can find a complete linear description of the convex hull of the feasible region for a problem with chained precedence constraints.

Non-simultaneous operations constraints There may be circumstances limiting a pair of operations from being processed at the same time, like safety restrictions or necessary production. In a factory with two backup power supply units operations involving maintenance on the first unit cannot be in progress when operating on the other unit, as at least one power supply unit has to be functioning. For a pair of non-simultaneous operations (nonops) A and B, either A has to proceed B or B has to proceed A. Either one is allowed.

Constraints of this form arise in single-machine scheduling problems. There, the nature of the machine limits any pair of operations to be processed at the same time. The constraints are called *disjunctive constraints*. For RCPS problems two operations demanding the same resource may be processed at the same time as long as there are enough resource units available. The nonops constraints then arise from the nature of the operations themselves, not from the resource capacities. See Figure 3.3 for an illustration.

The disjunctive constraints for machine scheduling problems have been widely studied. Fowler et. al [KKF09] give a brief and clear introduction to different formulations of the disjunctive constraints for single-machine scheduling problems, as well as some computational results. We shall mention some in Section 3.2 and Chapter 4. In Section 5.2 we introduce a formulation which to our knowledge is uncommon in the scheduling literature.

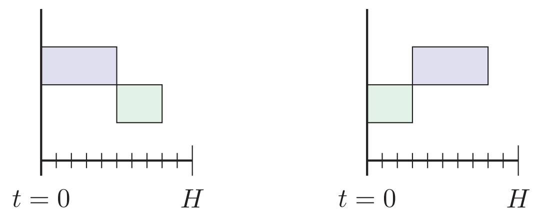


Figure 3.3: Non-simultaneous operations

Windows of opportunity constraints There may be constraints on which points in time an operation can be processed. Outside work may rely on good weather conditions like the wind being below a certain threshold, and some operations may need to start at the beginning of a work day in order to be completed by the end of the same day. Such windows of opportunity depend only on the suitable starting times for the operation, not on the status of other operations. See Figure 3.4 for an illustration where the operation has to start somewhere in the pink intervals.

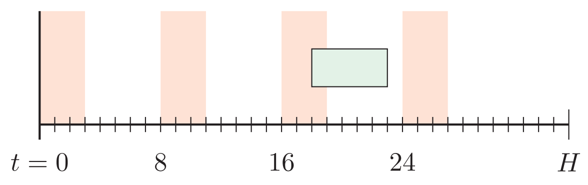


Figure 3.4: Windows of opportunity

3.2 Formulations for Scheduling

The problem of finding a feasible solution to a job scheduling problem can sometimes be very easy. A harder problem, on the other hand, is finding a good schedule. One way of accomplishing this is by designing problem specific heuristics that exploit the structure of the unique problem in order to obtain a good solution. In general this does not guarantee optimality. If the scheduling problem can be formulated as a combinatorial optimization problem there exists a great variety of algorithms which can solve the problem to optimality. This also holds within the framework of (mixed) integer linear programming. When working with a branch-and-bound/branch-and-cut solver a field of study is to find tight formulations of the feasible region to the problem. To do so there exist different variable types and formulation techniques used to model the problem. The models differ in the quality of the bound returned from solving the LP relaxation as well as the solution time spent by the solver. We shall mention some common formulations for scheduling problems here.

Time-indexed formulations

If the schedule is discretized into smaller units of time and the formulation includes variables for each such point in time we say that the formulation is *time-indexed*. If H denotes the planning horizon then a point in time is referred to as $t \in \{0, 1, \dots, H\}$. For an operation i that should be scheduled we let the binary variable y_i^t be equal to one if i starts at time t and zero otherwise. From these decision variables one can express many constraints and let the parameter values vary with the time t. This enables the possibility of being precise. Time-indexed formulations are known for producing tight bounds. One of the major drawbacks of using time-indexed formulations, however, is that the number of necessary variables and constraints increases when H is large. Large scheduling problems may become computationally intractable due to this.

Continuous formulations

If the schedule is not discretized we will call the formulations *continuous*. In a continuous formulation the starting time of an operation can take any real value as long as all constraints are satisfied. A strength of continuous formulations is that the worst-case number of constraints is normally reduced from time-indexed formulations. Some drawbacks, however, are the fact that some constraints are more difficult to formulate and that there is often need for additional helping variables. For complicated scheduling problems the cost of adding all this extra machinery may be higher than the reward of using a continuous formulation.

Big-M formulations

When formulating an IP problem we may encounter *disjunctive constraints* of the form "either this happens, or this must happen". A great example is two non-simultaneous operations A and B. Let $t_A, t_B \geq 0$ denote their starting time and $\alpha \in \mathbb{R}$ some minimum time gap between them. In a continuous formulation we must then have

$$t_A \geq t_B + \alpha \qquad \textit{or} \qquad t_B \geq t_A + \alpha.$$

We can linearize this *or*-equation by including a big constant M in a smart way. First, let z_{AB} be a binary linear ordering variable that is equal to one if A precedes B and zero otherwise. The *or*-equation can then be stated as

$$\begin{aligned} t_A &\geq t_B + \alpha && \textit{if } z_{BA} = 1, \\ t_B &\geq t_A + \alpha && \textit{if } z_{AB} = 1, \end{aligned}$$

where only one of z_{AB} and z_{BA} can be equal to one. We can remove the *if* conditions by using the Big-M variable:

$$\begin{aligned} t_A &\geq t_B + \alpha - M z_{AB}, \\ t_B &\geq t_A + \alpha - M z_{BA}. \end{aligned}$$

If $z_{BA} = 1$ then the first inequality will be $t_A \geq t_B + \alpha$. For M large enough the second one will be redundant as $t_B \geq 0$. The Big-M thus activates one of the constraints and leaves the other redundant.

The value of the Big-M must be chosen appropriately. If it is set too small the constraints may cut off feasible solutions. This could happen in the above case if $M < t_A + \alpha$. On the other side, M should not be set too big as this will lead to a weak formulation. Weak formulations may lead to poor bounds returned from solving the LP relaxation, which makes it harder to prune nodes in the branching tree of the branch-and-bound algorithm. The strength of the Big-M formulation, however, is that it reduces the number of constraints from time-indexed formulations.

Big-M constraints can be used when the variables have lower bounds implied by the rest of the formulation. Many also use the term Big-M constraints for *variable upper bounds* constraints of the form $x_1 \leq M x_2$. We also note that Big-M can refer to a method used to find an initial feasible solution in the Simplex algorithm, see [Sch86].

CHAPTER 4

The Main Problem: RCPSP

In this chapter we present the Resource Constrained Project Scheduling Problem (RCPSP) that will be studied in this thesis. This problem can be modelled in different ways, e.g., as an IP or MIP problem. We will introduce different formulations for the set of feasible solutions, and these will be further improved and compared in Chapter 5 and Chapter 7. The chapter is structured as follows: Section 4.1 gives an example of a simple instance of the RCPSP and works as an introduction before explaining the problem more generally in the upcoming sections. In Section 4.2 we explain the RCPSP with words and give some basic assumptions needed for the problem formulation. Section 4.3 introduces the different formulations of the constraints in the problem. We separate the time-indexed formulations from the continuous formulations. Section 4.4 introduces the weighted delay objective mathematically. In Section 4.5 we explain how the different models for the RCPSP will be named, and we give an example of a valid formulation of the set of feasible schedules. In Section 4.6 we explain how the RCPSP can be formulated as a rescheduling problem. The problem of rescheduling an already existing timetable in an optimal way was presented as one of the main challenges by our industry partners. We introduce new time-indexed variables and constraints in order to present a rescheduling approach strongly connected to the original scheduling problem. Our modelling decisions and assumptions made have been heavily influenced by the needs and current practices of our industry partners. A short discussion on these decisions will be given in Section 4.7. Finally, we relate our RCPSP to the standard RCPSP in Section 4.8. An overview of the symbols, model names and test data instances for the RCPSP is given in Appendix A.

4.1 An Example Problem

We start this chapter by giving a numerical example of an instance of the RCPSP in order to establish some notation. Let $\mathcal{I} = \{1, 2, 3\}$ be a set of operations that should be scheduled. The planning horizon H is one day consisting of eight work hours. Every operation needs some amount of the renewable resource r to be processed. The availability of r is $C_r^t = 3$ for every hour $t \in \{0, \dots, 8\}$. Operation 2 has one predecessor which is operation 1. We summarize the parameter values in Table 4.1. Here, i is an operation, L_i is the duration of i measured in hours, R_{ir} is the amount i needs of resource r and $\mathcal{P}_i$ is the set of predecessors to operation i.

i	L_i	R_{ir}	$\mathcal{P}_i$
1	3	2	$\emptyset$
2	4	1	$\{1\}$
3	2	1	$\emptyset$

Table 4.1: Example data for a small instance of the RCPSP

A schedule contains starting times for each operation. The objective in this example is to find a feasible schedule that minimizes the makespan. The schedule has to satisfy the resource capacity constraint saying that, for each hour, the demand for r of the operations in progress can not be greater than the availability. The schedule must also satisfy the precedence constraint saying that operation 1 must precede operation 2. An optimal solution to this problem is illustrated in Figure 4.1.

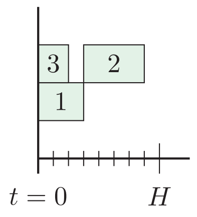


Figure 4.1: An optimal schedule for the example problem

4.2 Modelling Assumptions

In general, we are given a finite set $\mathcal{I} = \{1, \ldots, |\mathcal{I}|\}$ of operations that should be scheduled. One operation is usually denoted $i \in \mathcal{I}$. For two operations we will write i and j, and when considering a subset of k operations we will for simplicity choose $\{1, 2, \ldots, k\} \subseteq \mathcal{I}$. Every operation in $\mathcal{I}$ has to be scheduled in a way that satisfies all constraints (Section 4.3), or else the scheduling problem is considered infeasible. We are seeking to find the starting time t_i for each $i \in \mathcal{I}$.

The schedule is discretized into units of hours. The planning horizon H denotes the hour all operations must finish within and is an input parameter to the problem. Every operation must be scheduled within the horizon. A point in the schedule $t \in \{0, \ldots, H\}$ corresponds to an hour between, and only between 07:00-15:00 from Monday to Friday. In this way $\{0, \ldots, 7\}$ could refer to a Monday, and the next time index $t = 8$ then corresponds to 07:00 the following Tuesday. In the same way a Friday afternoon is followed by Monday morning. This time division is a simplification that ignores evenings, nights, weekends and other holidays. It is assumed that operations may be paused at the end of a work day and picked up at the beginning of the next work day without any added duration. If an operation is uninterruptible we assume that the duration is less than eight hours and model this by using windows of opportunity as illustrated in Figure 3.4. We will by $t \in [a, b]$ denote an integral time point in the closed interval from a to b.

Each operation i has a due date T_i and an associated cost of overrunning it, but no operations are given a strict deadline other than the planning horizon H. For every operation i the duration L_i is assumed to be positive and integer valued, and all relevant set-up and cleaning times are included in this duration. We also assume $L_i \leq T_i$. If some real-life operations have a duration of less than one hour these can, e.g., be merged as one operation i with $L_i = 1$. Every operation $i \in \mathcal{I}$ belongs to exactly one work order $w \in \mathcal{WO}$. The due date is similar for every operation within the same work order. Each operation i has a criticality that influences the cost B_i of delaying it. The problem does not consider release dates, and preemption is not allowed.

4.3 Constraints

We now present mathematical formulations of the RCPSP. A list of all the symbols that are used is given in Appendix A.1. In the following we assume that every sum limit is positive and within the relevant range. If, for instance, $t \in \mathcal{H}$ is the sum variable, terms where $t < 0$ and $t > H$ are defined to be zero. This is easily coded by using the limits $max(0, t)$ and $min(t, H)$, respectively. Sums with a lower bound strictly greater than the upper bound are defined to be zero.

General constraints

At first, it must be ensured that all operations are scheduled within the horizon. Using a time-indexed formulation we demand

$$\sum_{t=0}^{H} y_i^t = 1 \qquad i \in \mathcal{I}. \tag{4.1}$$

Here, y_i^t is equal to one if operation i starts at time t and zero otherwise. As every operation must finish before the planning horizon we additionally fix $y_i^t = 0$ for $t \in [H - L_i + 1, H]$ for each $i \in \mathcal{I}$. From the binary y_i^t variables we can express all the needed constraints and the objective. An extended formulation can be obtained by introducing start time variables $t_i \geq 0$ for each operation i. These can be linked to the time-indexed variables by using (4.2).

$$t_i = \sum_{t=0}^{H} t y_i^t \qquad i \in \mathcal{I} \tag{4.2}$$

Extending time-indexed formulations with t_i variables will make the modelling of some other constraints easier, however, it is uncertain at this point how it will affect the running time of the solver used to solve the problem. Continuous formulations will only use the t_i variables and demand $0 \leq t_i \leq H - L_i$.

Resource constraints

Each operation is assumed to require some amount of one or more partially renewable resources in order to be processed. The integer valued amount of resource type r required by operation i is denoted $R_{ir} \geq 0$. For each $t \in \mathcal{H}$ the availability for resource r is given by $C_r^t \geq 0$. At all times it must be ensured that the resource demand of the operations in progress does not exceed the capacity. This can be expressed by a time-indexed constraint [PWW69]:

$$\sum_{i\in\mathcal{I}} \sum_{s=t-L_i+1}^{t} R_{ir} y_i^s \leq C_r^t \qquad t \in \mathcal{H},\ r \in \mathcal{R} \tag{4.3}$$

This constraint ensures that no operation can be started unless there are enough qualified resources available to handle the task during its whole duration. For each operation i and each resource r we assume that there exists at least one t such that $C_r^s \geq R_{ir}$ for all $s \in [t - L_i + 1, t]$.

For this problem we will only consider human workers as resources. A resource type will refer to a craft, e.g., mechanics or plumbers, and a resource unit refers to one physical worker. We assume that each worker is fully interchangeable with other workers of the same type, and switching such resource units during an operation's performance is allowed. This allows an electrician to be replaced by another one while he is on a break, for instance. All resources of the same type are equally efficient, so we do not consider apprentices or injured workers. All resources are non-consumable in the sense that no injuries or reductions are inflicted on the resource after it has been used. Finally, each operation is only assumed to have one mode. This removes the possibility of assigning more workers to an operation to reduce its duration.

Interval flow formulation If the capacity of some craft is somewhat constant over time we can exploit this to get a formulation of the resource constraint which is not time-indexed in order to reduce the number of constraints when H is large. There exist a few approaches in the literature for finding non-time-indexed formulations of the resource capacity constraint, see e.g. [Kon+11] for RCPS problems. We will consider two approaches using multi-commodity flow constraints. For theory on flows in networks we refer to any book on integer programming and graph theory, e.g., [WN88]. We skip an introduction to this theory as it will only be relevant in this section.

Let $r \in \mathcal{R}$ be a resource type, e.g., electricians. As we consider eight-hour work days and a five-day work week it is possible that the availability is constant over some time period, e.g., five electricians available each work hour for the next month. Let $\mathcal{T}_r = \{t \colon C_r^{t-1} \neq C_r^t\} \cup \{0, H\}$ be the set of time points at which the availability of resource r changes. Let it be of the form $\mathcal{T}_r = \{T_r^0 = 0, T_r^1, T_r^2, \ldots, T_r^{K_r} = H\}$ and let $\mathcal{K}_r = \{0, \ldots, K_r - 1\}$. Given a $k \in \mathcal{K}_r$ we call the interval $[T_r^k, T_r^{k+1} - 1]$ for interval k. Let $C_r^{[k]}$ be the availability of resource r in time interval $k \in \mathcal{K}_r$. In the case $K_r = 1$ the availability of r is constant in the whole schedule, while for $K_r = H$ the availability changes for each hour. Let $C_r = \max_{k\in\mathcal{K}_r} C_r^{[k]}$.

Example 4.3.1. *Consider a schedule with a horizon of four weeks. The availability of electricians is five every work hour each week except for the second week where only three are available each hour due to some holiday. Then* $\mathcal{H} = \{0, \ldots, 160\},\ \mathcal{T}_r = \{0, 40, 80, 160\},\ C_r = C_r^{[0]} = C_r^{[2]} = 5,\ C_r^{[1]} = 3.$

We can model the resource constraints using multi-commodity flow equations where we model the flow of units of each resource type. A formulation for a problem with renewable resources with constant capacity is given by Artigues et al. in their book on the standard RCPSP [ADN08]. Our formulation with time intervals is inspired by Mannino and Riise [RM12]. Choose some $r \in \mathcal{R}$ and let $\mathcal{I}_r = \{i \in \mathcal{I} \colon R_{ir} > 0\}$ be the set of operations that have a positive demand for resource r. Consider the activity-to-node graph $G_r = (V_r, E_r)$ where $V_r = \mathcal{I}_r \cup S_r$ and S_r is a set of sources s_r^k and sinks e_r^k for the resource units flowing into and out from interval $k \in \mathcal{K}_r$. For each $i, j \in \mathcal{I}_r,\ i \neq j$ let $f_{ij}^r \geq 0$ be the amount of resource r that moves from processing operation i to processing operation j when i is finished. We let f_{ij}^r represent the flow going along the arc between node i and j in G_r. See Figure 4.2 for an illustration. Here, the labels along the arcs represent the flow. It may happen that an operation starts in one time interval, but ends in another interval. In the figure we represent this by using arcs with either no source or no sink.

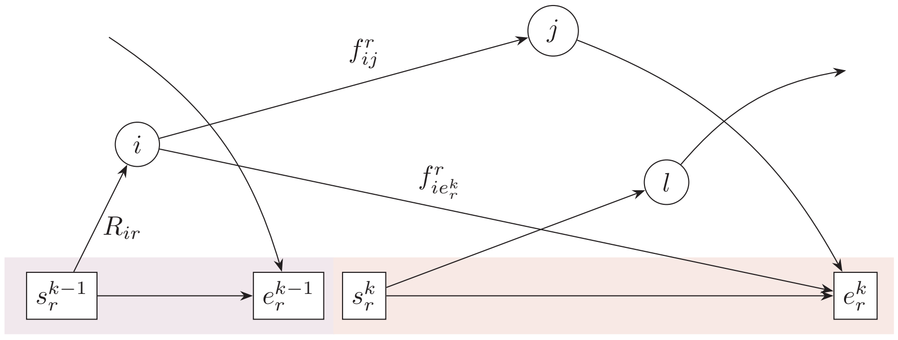


Figure 4.2: Flow of resource units

We allow two operations to run in parallel as long as their total resource demand for r does not exceed the capacity and as long as the operations are not non-simultaneous. Let z_{ij} be the linear ordering variable that is equal to one if i finishes before j starts and zero otherwise. Clearly, if $z_{ij} = 1$ then $z_{ji} = 0$ and vice versa, but we allow for both z_{ij} and z_{ji} to be zero if they run in parallel. Using a slightly extended Big-M approach from the one explained in Section 3.2 we can link the z_{ij} variables to the starting times of each operation. Along with non-negativity constraints a valid formulation for the resource constraints is as follows:

$$z_{ij} + z_{ji} \leq 1 \qquad r \in \mathcal{R},\ i < j \in \mathcal{I}_r \tag{4.4}$$

$$\sum_{t=0}^{H} t y_j^t \geq \sum_{t=0}^{H} t y_i^t + (L_i + H) z_{ij} - H \qquad r \in \mathcal{R},\ i \neq j \in \mathcal{I}_r \tag{4.5}$$

$$f_{ij}^r \leq C_r z_{ij} \qquad r \in \mathcal{R},\ i \neq j \in \mathcal{I}_r \tag{4.6}$$

$$\sum_{j\in\mathcal{I}_r} f^r_{ji} + \sum_{k\in\mathcal{K}_r} f^r_{s^k_r i} = R_{ir} \qquad r\in\mathcal{R},\ i\in\mathcal{I}_r \tag{4.7}$$

$$\sum_{j\in\mathcal{I}_r} f^r_{ij} + \sum_{k\in\mathcal{K}_r} f^r_{ie^k_r} = R_{ir} \qquad r\in\mathcal{R},\ i\in\mathcal{I}_r \tag{4.8}$$

$$\sum_{i\in\mathcal{I}_r} f^r_{s^k_r i} + f_{s^k_r e^k_r} = C^{[k]}_r \qquad r\in\mathcal{R},\ k\in\mathcal{K}_r \tag{4.9}$$

$$\sum_{i\in\mathcal{I}_r} f^r_{ie^k_r} + f_{s^k_r e^k_r} = C^{[k]}_r \qquad r\in\mathcal{R},\ k\in\mathcal{K}_r \tag{4.10}$$

$$f_{s^k_r i} \le C^{[k]}_r \sum_{t=T^k}^{T^{k+1}-1} y^t_i \qquad r\in\mathcal{R},\ k\in\mathcal{K}_r,\ i\in\mathcal{I}_r \tag{4.11}$$

$$f_{ie^k_r} \le C^{[k]}_r \sum_{t=T^k-L_i}^{T^{k+1}-L_i-1} y^t_i \qquad r\in\mathcal{R},\ k\in\mathcal{K}_r,\ i\in\mathcal{I}_r \tag{4.12}$$

Equation set 4.1: Interval flow formulation of the resource constraints

Here, (4.4) expresses that either i is scheduled before j, j is scheduled before i or they run in parallel. Equation (4.5) links the linear ordering variables to the starting times, where we have used the planning horizon H as the Big-M constant. Equation (4.6) is a variable linking constraint that forces the flow f^r_{ij} to be zero if i is not scheduled before j, and it forces i to be scheduled before j if f^r_{ij} is greater than zero. Equations (4.7)-(4.8) are flow balance equations that ensure demand satisfaction for every operation that requires resource r. Resource capacities are given in (4.9)-(4.10) for the sources and sinks, respectively. Finally, equations (4.11)-(4.12) ensure that there is no flow from s^k_r (i) to i (e^k_r) if i does not start (end) in time interval k.

The number of constraints in this *interval flow* formulation is $\mathcal{O}(H|\mathcal{R}||\mathcal{I}|)$ in the worst case when $K_r = H$ and the availabilities change for every time step. The time-indexed formulation with size $\mathcal{O}(H|\mathcal{R}|)$ is then preferred. In the best case the resources are renewable with $K_r = 1$ for each $r \in \mathcal{R}$ and the size of the interval flow formulation reduces to $\mathcal{O}(|\mathcal{R}||\mathcal{I}|)$. In general this is not the case. In order to reach this we shall introduce another flow formulation that is not interval-dependent or in need of the binary y^t_i variables.

Continuous flow formulation Resources with capacities that vary over time can be converted into renewable resources with constant capacity by using an approach inspired by Bartusch et al. [BMR88]. For each resource $r \in \mathcal{R}$ and time interval $k \in \mathcal{K}_r$ we introduce a "dummy operation" i_{rk} and schedule it to be processed in interval k, i.e., we fix $t_{i_{rk}} = T^k_r$ and $L_{i_{rk}} = T^{k+1}_r - T^k_r$. The operation should have a demand of the gap between the maximum capacity of r and the capacity of r in that interval, so we set $R_{i_{rk}r} = C_r - C^{[k]}_r$ and add i_{rk} to $\mathcal{I}_r$. In Example 4.3.1 a dummy operation with the demand for two electricians would be introduced and set to be processed the second week. After introducing every dummy operation and fixing their staring times we can for each $r \in \mathcal{R}$ set $C_r^t = C_r$ for all $t \in \mathcal{H}$, which ensures that every resource r has $K_r = 1$. Bartusch et al. prove that such an updated problem is polynomially equivalent to the original one.

In the new interval-free flow formulation we only need one source s_r and sink e_r for each resource $r \in \mathcal{R}$. This also removes the need for constraints (4.11)-(4.12), and thereby the need for the binary y_i^t variables as t_i can be used in (4.5) by (4.2). Recall then the bounds $0 \leq t_i \leq H - L_i$.

$$(4.4), (4.6)$$

$$t_j \geq t_i + (L_i + H)z_{ij} - H \qquad r \in \mathcal{R},\ i \neq j \in \mathcal{I}_r \tag{4.13}$$

$$\sum_{j \in \mathcal{I}_r} f_{ji}^r + f_{s_r i}^r = R_{ir} \qquad r \in \mathcal{R},\ i \in \mathcal{I}_r \tag{4.14}$$

$$\sum_{j \in \mathcal{I}_r} f_{ij}^r + f_{ie_r}^r = R_{ir} \qquad r \in \mathcal{R},\ i \in \mathcal{I}_r \tag{4.15}$$

$$\sum_{i \in \mathcal{I}_r} f_{s_r i}^r + f_{s_r e_r} = C_r \qquad r \in \mathcal{R},\ k \in \mathcal{K}_r \tag{4.16}$$

$$\sum_{i \in \mathcal{I}_r} f_{ie_r}^r + f_{s_r e_r} = C_r \qquad r \in \mathcal{R},\ k \in \mathcal{K}_r \tag{4.17}$$

Equation set 4.2: Continuous flow formulation of the resource constraints

We still need (4.4) and (4.6) from the interval flow formulation, but (4.13) replaces (4.5). Equations (4.14)-(4.15) are flow balance constraints for the operations that demand r. Equations (4.16)-(4.17) ensure that every resource unit flows through the network.

Precedence constraints

We will formulate precedence constraints between pairs of operations. Using start time variables the constraint

$$t_i \geq t_j + L_j \qquad i \in \mathcal{I},\ j \in \mathcal{P}_i \tag{4.18}$$

expresses that, if j is a predecessor to i, j has to finish before or at the same time point as i starts [PWW69]. This finish-to-start relation is the only precedence relation relevant to our RCPSP. Maximal time-lags are not considered. By inserting (4.2) we obtain a time-indexed constraint

$$\sum_{t=0}^{H} t y_i^t \geq \sum_{t=0}^{H} t y_j^t + L_j \qquad i \in \mathcal{I},\ j \in \mathcal{P}_i, \tag{4.19}$$

where we must additionally fix $y_i^t = 0$ for $t < L_j$.

The above constraints can express precedence relations between any pairs of operations $i, j \in \mathcal{I}$, regardless of which work order they are contained in. This is necessary as no general assumptions will be made on which operations that have predecessors or successors, besides that the constraints should not give rise to cycles. Some structures are common for the real-life problems studied in the industry, like having a fixed ordering on some or all of the operations within a work order [Ind22]. Structures like these will be assumed common in the test instances used in Chapter 7. The heuristic that we make in Appendix B will assume that each work order has a predefined ordering of its operations. In Section 5.3 we inspect how the above formulation of the precedence constraints can be strengthened by finding valid cutting planes.

Non-simultaneous operations constraints

Given two nonops i and j we must either have that $t_i \geq t_j + L_j$ or that $t_j \geq t_i + L_i$ to prevent them from being active at the same time. One way to ensure this is by using a Big-M constraint and linear ordering variables. Then

$$z_{ij} + z_{ji} = 1 \qquad \{i,j\} \in \mathcal{X} \tag{4.20}$$

is a logical constraint that must be valid as the operations can not overlap. Recall that H is the planning horizon. A valid Big-M formulation with $M = H$ for the nonops constraints is obtained by using (4.20) and the two constraints

$$\begin{aligned} t_i &\geq t_j + L_j - Hz_{ij} \qquad && \{i,j\} \in \mathcal{X}, && (4.21)\\ t_j &\geq t_i + L_i - Hz_{ji} \qquad && \{i,j\} \in \mathcal{X}, && (4.22)\end{aligned}$$

or, with the time-indexed variables

$$\begin{aligned} \sum_{t=0}^{H} t y_i^t &\geq \sum_{t=0}^{H} t y_j^t + L_j - Hz_{ij} \qquad && \{i,j\} \in \mathcal{X}, && (4.23)\\ \sum_{t=0}^{H} t y_j^t &\geq \sum_{t=0}^{H} t y_i^t + L_i - Hz_{ji} \qquad && \{i,j\} \in \mathcal{X}. && (4.24)\end{aligned}$$

The constraint for a pair of nonops can also be formulated using a time-indexed formulation without the Big-M. There are different ways of doing so. One is, loosely speaking, to look forward in time and express that from t and onwards the two operations i and j cannot be active at the same time:

$$\sum_{s=t}^{t+L_j-1} y_i^s + \sum_{s=t}^{t+L_i-1} y_j^s \leq 1 \qquad \{i,j\} \in \mathcal{X},\ t \in \mathcal{H} \tag{4.25}$$

For a pair $\{i,j\} \in \mathcal{X}$ of nonops this constraint must be generated for every $t \in \mathcal{H}$. This is a drawback of the time-indexed formulation compared to the Big-M formulation, as there only two constraints are needed. Instead of looking only forward, one can look forward and backward from t finding the times j cannot be in progress if i starts at t. That is,

$$y_i^t + \sum_{s=t-L_j+1}^{t+L_i-1} y_j^s \leq 1 \qquad \{i,j\} \in \mathcal{X},\ t \in \mathcal{H}. \tag{4.26}$$

Note that to have a valid formulation it suffices to add the constraints in this form. Switching the roles of i and j and adding those constraints as well is redundant. One last possibility is to look only backward from t and say that both operations cannot start so that they are active in t:

$$\sum_{s=t-L_i+1}^{t} y_i^s + \sum_{s=t-L_j+1}^{t} y_j^s \leq 1 \qquad \{i,j\} \in \mathcal{X},\ t \in \mathcal{H} \tag{4.27}$$

We will not make any assumptions about which operations that are non-simultaneous. In some situations we may encounter sets of more than two nonops or observe that the nonops require the same resources. We shall exploit this to find valid cutting planes in Chapter 5.

For RCPS problems non-simultaneous operations constraints are not common. If they are, the standard approach is to view the nonops as operations that have a demand of one for some renewable resource with a constant capacity of one, and to model these constraints together with the resource constraints. Such resources are called *dedicated resources*. We have not found any RCPS problem with another approach.

Windows of opportunity

Using a time-indexed formulation we can restrict the intervals in which an operation can start by demanding

$$\sum_{t \in \mathcal{W}_i} y_i^t = 1 \qquad i \in \mathcal{I}, \tag{4.28}$$

where $\mathcal{W}_i$ is the set of feasible start times for operation i. If $\mathcal{W}_i = \mathcal{H}$ we avoid adding this inequality as it will be redundant. This constraint is therefore only necessary for operations with a window of opportunity that is a strict subset of the set of all time points. An alternative approach to introducing (4.28) is to fix $y_i^t = 0$ for all $t \in \mathcal{H} \setminus \mathcal{W}_i$.

Windows of opportunity are not included in most RCPS problems. Drexl et al. [Dre+00] are some of the few who include a variant of these constraints. With each operation i they associate one time window $[\mathrm{EF}_i, \mathrm{LF}_i]$ denoting the earliest and latest finish time of operation i, as well as a set N_i of forbidden time periods. This approach is inspired by classroom timetabling problems. Together with job-subset restrictions and mode selection they formulate a GUB inequality which ensures each job finishes in a non-forbidden time period.

We propose a new approach for modelling the windows of opportunity by considering them as time intervals with availability for some resource. For every operation i we create a fictive resource r_i which is only available within the windows of opportunity for operation i. Since $\mathcal{W}_i$ includes the valid starting times for operation i we let $C^s_{r_i} = 1$ for all $s \in [t, t + L_i - 1]$ if $t \in \mathcal{W}_i$. From these time-indexed availabilities we can create the interval availabilities $C^{[k]}_{r_i}$ as explained earlier in this section. Likewise we set $R_{ir_i} = 1$, $\mathcal{I}_{r_i} = \{i\}$ and add r_i to $\mathcal{R}$. In this way the windows of opportunity constraints can be viewed as resource capacity constraints. We will consider this approach when using the interval flow formulation (4.4)-(4.12) and the continuous flow formulation (4.4),(4.6),(4.13)-(4.17) for the resource capacity constraints.

4.4 Objective

We will consider the single objective of minimizing weighted tardiness, also called weighted delay. The delay of operation i is the number of hours i finishes after the due date T_i. The cost of delaying i with one time unit is denoted B_i and will be used as the weights in the objective. Thus, we can write the objective as

$$\min \quad \sum_{i \in \mathcal{I}} \sum_{t=T_i+1}^{H} B_i(t - T_i) y_i^{t-Li}. \tag{4.29}$$

Continuous delay variable An alternative to the above objective is to introduce a continuous delay variable $d_i \geq 0$ measuring how delayed i is. The objective is then

$$\min \quad \sum_{i \in \mathcal{I}} B_i d_i, \tag{4.30}$$

where we additionally must add one of the following constraints to the formulation:

$$d_i \geq t_i + L_i - T_i \qquad i \in \mathcal{I} \tag{4.31}$$

$$d_i \geq \sum_{t=0}^{H} t y_i^t + L_i - T_i \qquad i \in \mathcal{I} \tag{4.32}$$

Binary delay variables We propose another approach, which is to binarize the delay variable d_i. This can be done when all the parameter values, and thereby d_i, are integral. Let w_i^t be equal to one if i is delayed t hours and zero otherwise. As every operation must finish within H we fix w_i^t to zero for all $t \in [H - T_i + 1, H]$. By adding the two constraints

$$\sum_{t=0}^{H} w_i^t = 1 \qquad i \in \mathcal{I} \tag{4.33}$$

$$\sum_{t=0}^{H} t w_i^t \geq \sum_{t=0}^{H} t y_i^t + L_i - T_i \qquad i \in \mathcal{I} \tag{4.34}$$

to the formulation, the objective can be written as

$$\min \quad \sum_{i\in\mathcal{I}} B_i \left(\sum_{t=0}^{H} t w_i^t \right). \tag{4.35}$$

We will inspect how these three objectives affect the running time of the optimization solver in Chapter 7. In Chapter 5 we will search for cutting planes for the delay constraints (4.31) and (4.34).

4.5 Naming the Models

In the RCPSP our goal is to find a feasible schedule that minimizes the weighted tardiness objective. Given the constraints introduced above the RCPSP can be stated in the following general optimization form:

$$\begin{array}{rl} \min & \text{weighted tardiness} \\ & \\ \text{subject to} & \text{general constraints} \\ & \text{resource constraints } (r) \\ & \text{precedence constraints } (p) \\ & \text{nonops constraints } (n) \\ & \text{delay constraints } (d) \\ & \text{windows of opportunity } (w) \\ & \text{variable bounds} \\ & \text{some variables binary} \end{array} \tag{RCPSP}$$

We will denote the set of feasible schedules to the RCPSP by X. Mathematically this will be a set whose dimension depends on which variables that are used to model the problem. For simplicity we will always refer to the feasible region as X independent of which formulation that is used.

When choosing a formulation of X it is possible to "mix and match" different constraint types, e.g., using the time-indexed resource constraint, but the continuous Big-M formulation of the nonops constraints. Our wish is to compare different formulations with respect to running time and quality of lower bounds in order to get a clearer picture of which formulations that are well suited to use for real-life instances of the RCPSP. In order to separate different formulations for X from each other we will introduce a rule for naming the different polyhedra. A formulation of X will be named in the following way:

$$Pr()p()n()d()w()$$

Here, $r()$ refers to the resource constraint that is used. A reference to the constraint will be given inside the parenthesis. In the same way $p()$ refers to the precedence constraint, $n()$ to the nonops constraint(s), $d()$ to the delay constraint(s) and $w()$ to the windows of opportunity constraint. We write two letters next to each other if the contents inside the parenthesis are equal, e.g., we then write $rw()$ instead of $r()w()$. We start the name with a P to specify that it is a formulation for X and that it is a polyhedron (all constraints are linear). An objective is denoted by $o()$ with a reference to the objective in parenthesis. By a *model* for the RCPSP we mean the choice of an objective and a formulation for X. A model for the RCPSP is denoted $(o(), Pr()p()n()d()w())$. We assume that all relevant variables, general constraints, variable bounds constraints and binary variable constraints are added to the model as well. In Table A.1 in Appendix A we give abbreviations that will be used instead of referring to the equations by numbers. There, TI stands for time-indexed, BM for Big-M and C for continuous. As an example, when using the time-indexed formulation (4.3) for the resource constraint we will write $r(TI)$ instead of $r(4.3)$ to emphasize that we use a time-indexed constraint.

Example 4.5.1 (A time-indexed model with Big-M and delay variable)**.** *The MIP model for the RCPSP using objective* (4.30)*, the time-indexed resource constraint* (4.3)*, time-indexed precedence constraint* (4.19)*, time-indexed Big-M constraints for non-simultaneous operations* (4.20)*,* (4.23)*-*(4.24)*, delay constraint* (4.32) *and time-indexed window of opportunity constraint* (4.28) *is named*

$$(o(C), Prpw(TI)n(BMTI)d(CTI)).$$

The full model is written

$$\begin{array}{lll}
\min & \sum_{i\in\mathcal{I}} B_i d_i & \\[1em]
\text{subject to} & \sum_{t=0}^{H} y_i^t = 1 & i \in \mathcal{I} \\[1em]
& \sum_{i\in\mathcal{I}} \sum_{s=t-L_i+1}^{t} R_{ir} y_i^s \leq C_r^t & r \in \mathcal{R},\ t \in \mathcal{H} \\[1em]
& \sum_{t=0}^{H} t y_i^t \geq \sum_{t=0}^{H} t y_j^t + L_j & i \in \mathcal{I},\ j \in \mathcal{P}_i \\[1em]
& z_{ij} + z_{ji} = 1 & \{i,j\} \in \mathcal{X} \\
& \sum_{t=0}^{H} t y_i^t \geq \sum_{t=0}^{H} t y_j^t + L_j - H z_{ij} & \{i,j\} \in \mathcal{X} \\
& \sum_{t=0}^{H} t y_j^t \geq \sum_{t=0}^{H} t y_i^t + L_i - H z_{ji} & \{i,j\} \in \mathcal{X} \\[1em]
& d_i \geq \sum_{t=0}^{H} t y_i^t + L_i - T_i & i \in \mathcal{I} \\[1em]
& \sum_{t\in\mathcal{W}_i} y_i^t = 1 & i \in \mathcal{I} \\[1em]
& d_i \geq 0 & i \in \mathcal{I} \\
& y_i^t \in \{0,1\} & i \in \mathcal{I},\ t \in \mathcal{H} \\
& z_{ij}, z_{ji} \in \{0,1\} & \{i,j\} \in \mathcal{X}.
\end{array}$$

4.6 A Rescheduling Model

In the above sections we have introduced models for the RCPSP. The RCPSP is a scheduling problem with the goal of finding feasible starting times for every operation in a way that minimizes the weighted delay. The planning horizon can vary from days, weeks or even months ahead. During the operational phase when work orders are processed the chances of encountering unexpected interruptions that affect the schedule are high. Such unplanned events could be missing crew, a critical machine failure or new incoming work orders. When such interruptions arise a rescheduling of the already existing schedule must be done such that, e.g., the updated resource capacities are not exceeded or the incoming work order is assigned a starting time. In Figure 4.3 we have illustrated this for the case when a fire breaks out. The red line shows the upcoming point in time $t = t'$ and the new operation of "putting out the fire" has due date $T_i = t'$ with an extremely high cost of overrunning it. In this case the operation planners will send a crew to put out the fire immediately and perform rescheduling from $t = t'$ to H. When rescheduling from t' we allow operations that are in progress at t' to be paused and picked up again at a later point in time.

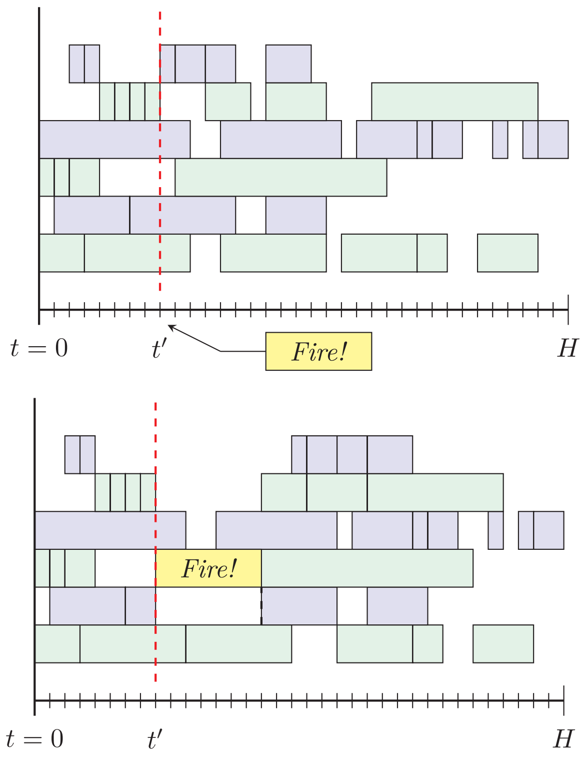


Figure 4.3: A rescheduling scenario

When rescheduling we assume that we are given a feasible schedule to the RCPSP, ideally an optimal one. In Table 4.2 we introduce some new variables used when modelling the rescheduling problem. We split the set of operations into two disjoint sets $\mathcal{I} = \mathcal{I}_{OLD} \cup \mathcal{I}_{NEW}$. If there are no new operations to schedule we set $\mathcal{I}_{NEW} = \emptyset$. In this case rescheduling has to be done due to changes in the problem constraints, e.g., updated windows of opportunity which the current schedule violates. In either case our objective will be to find a new schedule that schedules new operations and satisfies all updated constraints.

$\mathcal{I}_{OLD}$	set of operations in the schedule
$\mathcal{I}^*_{OLD}$	set of critical operations in the schedule
$\mathcal{I}_{NEW}$	set of new operations that need to be scheduled
Q_i	scheduled start time of operation $i \in \mathcal{I}_{OLD}$
W_i^+	cost of postponing i one hour
W_i^-	cost of pushing i one hour forward
t_i	scheduled start time of operation i after rescheduling
δ_i^+	hours i is postponed after rescheduling
δ_i^-	hours i is pushed forward after rescheduling

Table 4.2: Symbols used in the rescheduling problem

The new tasks $i \in \mathcal{I}_{NEW}$ may potentially be highly critical with a due date of "as early as possible", like putting out the fire in Figure 4.3. Minimizing the delays of such operations is therefore crucial. However, this may lead to many deviations from the old schedule, creating confusion and inefficiency among the workers [Ind22]. To keep the new schedule as close to the old one as possible the rescheduling models can minimize $\sum_{i\in\mathcal{I}_{OLD}} |Q_i - t_i|$. This is a nonlinear term that can be linearized by introducing new variables. Let $\delta_i = (\delta_i^+, \delta_i^-)$ and ensure $\delta \geq |Q_i - t_i|$ by adding the constraints $\delta_i^+ \geq t_i - Q_i$ and $\delta_i^- \geq Q_i - t_i$ to the rescheduling model along with non-negativity constraints. By adding weights $W_i = (W_i^+, W_i^-)$ one can minimize the weighted distance to the old schedule $\sum_{i\in\mathcal{I}_{OLD}} W_i^T \delta_i$.

Let $I^*_{OLD} \subseteq \mathcal{I}_{OLD}$ denote the old operations that are critical, i.e., their B_i value is large. We add the delay costs of these operations to the objective function to ensure that a new schedule still prioritizes not to delay these operations. Using an objective of the form (4.30) this results in the following model for the rescheduling problem:

$$\begin{array}{lll}
\min & \sum_{i\in\mathcal{I}_{OLD}} W_i^T \delta_i + \sum_{i\in\mathcal{I}_{NEW}\cup\mathcal{I}^*_{OLD}} B_i d_i & \\
\text{subject to} & \delta_i^+ \geq t_i - Q_i & i \in \mathcal{I}_{OLD} \\
& \delta_i^- \geq Q_i - t_i & i \in \mathcal{I}_{OLD} \\
& \textit{updated RCPSP constraints} & \\
& & \\
& \delta_i^+, \delta_i^- \geq 0 & i \in \mathcal{I}_{OLD}
\end{array} \tag{4.36}$$

Recall that t_i can be replaced by (4.2), and that the RCPSP constraints can be formulated in different ways. We will test how different models perform on the rescheduling problem in Chapter 7.

4.7 From Problem to Model

The road going from a real-life practical or scientific problem to a mathematical model is a road filled with crossroads and shortcuts. Decisions need to be made which usually need to weigh the level of detail and similarity with the real world against the model's simplicity, low complexity and generality. We claim that such decisions are of particular interest in the field of computational science and applied mathematics. Real-world and scientific problems can be complex, and they often need to be significantly simplified in order to be understood, modelled and solved within a reasonable time to a satisfying level. The computational scientist must design a framework in which to fit the data and choose appropriate algorithms for running experiments under well suited conditions, hoping to find that the simulated results are realistic. Once the problem can be related to earlier well-studied problems one has expectations and results to lean on which help validate the achieved results. Within integer programming and job scheduling this is especially useful when choosing how to formulate the constraints for the problem.

The RCPSP studied in this thesis springs out from a real-world maintenance scheduling problem in the industry. When defining the optimization problem our emphasis was to understand which constraints, rules and practices that were important to include in our problem formulation, as well as understanding what was meant by a ”good schedule”. Once the core problem had been defined we made simplifications in order to get a clean mathematical model, but also to be able to solve problems with low data quality. We mention some of these simplifications below.

Eight hour work days The current schedules used by our industry partners measure durations in hours and schedule operations to start at an hour. This lead us to believe that integer programming with time-indexed formulations of units of hours was well suited for modelling the problem. However, the division of days into eight hours is a simplification. Our industry partner uses shifts and has workers available each evening and night, although most operations are performed in daytime. Introducing shift modelling would have increased the complexity of the mathematical model by, e.g., introducing decision variables for each resource unit. The required data quality would also be much higher, so high, that it was early decided to ignore shifts and only model eight-hour work days. The operation planners can manually choose to process operations in evenings or nights if the need arises.

Human resources To process an operation one needs more resources than just human workers. Our models can in principle support the need for any partially renewable resource with specified availability at each hour, like safety equipment or machines. We do not model the need for consumption of nonrenewable resources like gasoline, duct tape or money from a budget. This is mainly due to the need for better data quality, like an estimate of each operation's demand as well as refill times.

Simple objective The true measure of a ”good schedule” is complicated. It may include different measures like robustness, worker utilization percentages, financial perspectives or connections to activities outside the schedule, e.g., production activities in other areas. Our choice of the simple weighted tardiness objective ensures a feasible schedule that minimizes delay costs, ideally ensuring that all work orders are finished within their due date. Given this optimal schedule one can easily use the solution values to calculate other measures of quality and use rescheduling as a what-if analysis tool.

4.8 Relation to the Standard RCPSP

After walking the road from a real-life problem to a general optimization model and working with the models for many months, we realized that our problem is strongly connected to the standard well-studied RCPSP. The standard RCPSP is denoted $PS \mid prec \mid C_{\max}$ and is studied in the book by Artigues et al. [ADN08]. The problem considers the scheduling of n non-preemptive operations in one project (work order) in a way that satisfies renewable resource constraints and finish-to-start precedence constraints. The objective is to minimize the makespan. This problem is $\mathcal{NP}$-hard and the decision version of determining whether there exists a schedule with makespan less than some constant M is $\mathcal{NP}$-complete. The RCPSP studied in this thesis differs from the standard RCPSP in some areas:

- The resources are partially renewable with capacities varying in time.
- We consider multiple work orders which may require the same resources.
- Non-simultaneous operations constraints and windows of opportunity are included.
- The objective is to minimize weighted tardiness.

Although there are some differences from the standard RCPSP we point out that the problems are very similar. For instance there are no release dates, set-up times or maximal time-lags, no nonrenewable resources and preemption is not allowed except when solving the rescheduling problem. These are some of the characteristics and constraints which are considered in other extensions of the RCPSP. We refer to the surveys of Węglarz et al. [Węg+11] and Hartman and Briskorn [HB10] for more information on the different variants of the RCPSP. References to some relevant articles on variants of the RCPSP have been given in the above sections. To the best of our knowledge the exact RCPSP studied in this thesis has not been studied before.

CHAPTER 5

Strengthening The Formulations

When solving job scheduling problems with integer programming approaches the search for strong formulations of the feasible region is important. Adding valid inequalities to a formulation may increase the quality of the bound returned when solving the LP relaxation, however, strengthening the formulation by adding too many cutting planes may increase the solution time used by the solver algorithm considerably. It is of theoretical and computational interest to study valid inequalities and cutting planes arising from scheduling problems, and we will do this now for the RCPSP.

The chapter is structured as follows: In Section 5.1 we refer to some cut generation software that was used as inspiration for deriving cutting planes by hand. Then, Section 5.2 studies how the non-simultaneous operations constraints can be reformulated and strengthened. We here introduce the permutation formulation, which to our knowledge gives a new approach in the literature for Big-M constraints. Section 5.3 studies the precedence constraints. We find and give proof of the convex hull of a specific precedence constrained polyhedron. In Section 5.4 we study cutting planes for the delay constraints. Finally, Section 5.5 studies the time-indexed resource constraint. We study this by itself and in combination with other constraints.

In this chapter we will let X denote the feasible region for (RCPSP), as mentioned in Section 4.5. Recall that the symbols used in the formulations of X are given in Appendix A.1. The RCPSP includes resource constraints (r), precedence constraints (p), nonops constraints (n), delay constraints (d) and windows of opportunity constraints (w), as well as relevant general constraints and variable bounds which we assume are always included in the formulation. When we consider problems with only precedence constraints we will denote the feasible region by X_p. If the problem also contains resource capacity constraints we write X_{rp}, and so on. This notation is needed when we study the sets arising from only some types of constraints. We let $X_{rpndw} = X$. A formulation for X will be denoted as P, a formulation for X_{rp} as P_{rp} and so on. As we shall consider cutting planes that separate points in P from $\text{conv}(X)$ we will frequently use the term *P/X cutting plane*.

Definition 5.0.1 (P/X cutting plane)**.** *Let P be a formulation for X. We say that an inequality $a^T x \leq \alpha$ is a P/X* cutting plane *if it is valid for* $\text{conv}(X)$ *and there exists at least one point $x^* \in P$ such that $a^T x^* > \alpha$.*

5.1 Tools for Finding Cutting Planes

Cutting planes valid for conv(X) may be deduced by general methods using addition of inequalities and rounding procedures. Such cuts are called *general cutting planes.* For the RCPSP our wish was to search for cutting planes that arise from the structure of the problem, e.g., from combining different constraint types. As a starting point for this search we used optimization software on small test instances of the RCPSP to see which cuts that were generated. When observing similar structures in several cuts we used this as inspiration to derive cutting planes on a more general form by hand.

PORTA, which stands for POlyhedron Representation Transformation Algorithm, is a free software for analyzing polytopes and polyhedra [CL22]. Recall Motzkin's representation theorem, Theorem 2.1.1. When presented with a "small" polyhedron described either by a system of linear inequalities or explicitly by its extreme points and extreme rays PORTA can calculate and return the other equivalent description. For binary programs in small dimensional spaces PORTA can take as input the constraints in a formulation for $X \subseteq \{0,1\}^n$ and return the constraints that define conv(X). This feature helped us on the way towards finding Theorem 5.3.3.

FICO Xpress Solver is a commercial solver which can be used to solve optimization problems like LP, IP and MIP problems [Fai22]. One of the features Xpress supports is to return cutting planes which are generated throughout the solution process. We used this feature to examine the cuts that were generated at the root node of the branching tree when solving the LP relaxation of different problem instances.

5.2 Non-simultaneous Operations Constraints

We start by studying the non-simultaneous operations. A pair of non-simultaneous operations $\{i,j\} \in \mathcal{X}$ is a pair of operations that cannot overlap in the schedule. Consider the three time-indexed formulations (4.25)-(4.27) given in Chapter 4. We visualize the constraints in Figure 5.1 on the following page. Here $L_i = 4,\ L_j = 3$. Graph (i) corresponds to the constraint looking forward in time (4.25), (ii) to the one looking forward and backward (4.26) and (iii) to the one looking backward (4.27). The node in row i and column t corresponds to the binary variable y_i^t. Two variables whose nodes are adjacent cannot both be equal to one at the same time. The graphs (i)-(iii) thus show cliques arising from the nonops constraints for $\{i,j\} \in \mathcal{X}$ at time $t \in \mathcal{H}$. One could believe that the formulations using the cliques in (i) and (iii) may be stronger than (ii) as the clique between all the j nodes is already generated by (4.1). We shall test and compare how the three formulations perform in Chapter 7. For now, we wish to inspect if some of these constraints can be strengthened.

Consider the case when there are three operations $i,j,k \in \mathcal{I}$ such that $\{i,j\},\{j,k\},\{i,k\} \in \mathcal{X}$. Having all these pairs in $\mathcal{X}$ is equivalent to saying that none of the three operations $i,\ j$ or k can be performed simultaneously. Instead of looking at three pairs we could consider the set $\Delta = \{i,j,k\}$ and demand that within this set all operations must be non-simultaneous. This leads to the definition of a new set; let $\mathcal{X}^*$ be a set of sets of operations such that for each $\Delta \in \mathcal{X}^*$ all operations in Δ must be non-simultaneous, i.e., $\{i,j\} \in \mathcal{X}$

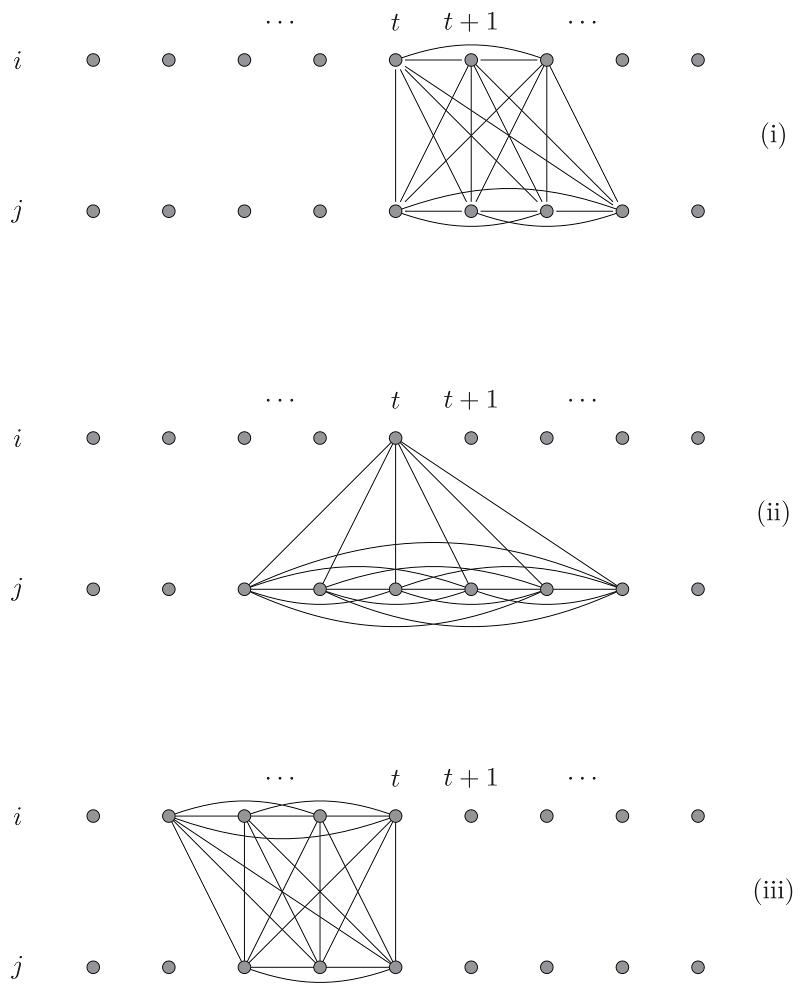


Figure 5.1: Cliques arising from non-simultaneous operations

for all $i \neq j \in \Delta$. We also demand that each Δ is maximal, i.e., if $\Delta \in \mathcal{X}^*$ and $s \notin \Delta$ we must have $\{i, s\} \notin \mathcal{X}$ for at least one $i \in \Delta$, or else we must have $s \in \Delta$. The inspiration for working with $\mathcal{X}^*$ rather than $\mathcal{X}$, as originally intended, came from a discussion with our industry partners [Ind22]. Usually, the nonops are stored as a set Δ consisting of two to five operations, and the number of such sets is small compared to the number of operations. When the nonops are stored as such sets, which we assume are maximal, we can access all $\Delta \in \mathcal{X}^*$ directly and use Δ in the nonops constraint. Hopefully, this will increase the size of the maximal cliques in Figure 5.1.

Consider first the forward-looking constraint (4.25) and let $\Delta = \{1, \ldots, k\}$

for some k. In the constraint for $\{i,j\} \in \mathcal{X}$ and $t \in \mathcal{H}$ the upper summation bounds depend on the duration of the other operation. When combining the cliques of type (i) for all pairs of operations in Δ for some fixed t some nodes will not be contained in the intersection of all the cliques. The nodes in the intersection will together form a maximal clique. The corresponding clique inequality is

$$\sum_{i\in\Delta} \sum_{s=t}^{t+L_{\underline{i}}-1} y_i^s \leq 1,$$

where $L_{\underline{i}} = \min_{j\in\Delta, j\neq i} L_j$.

Things are similar for the constraint (4.26) looking forward and backward. The upper summation bound also here depends on the duration of the other operation. The clique constraint arising from the maximal clique obtained when combining all constraints for some t is

$$\sum_{i=1}^{k-1} y_i^t + \sum_{s=t-L_k+1}^{t+L_{\underline{k}}-1} y_k^s \leq 1.$$

Both of the above clique inequalities are valid inequalities for the set X_n of feasible solutions to the RCPSP with only nonops constraints. However, in these inequalities some of the edges that were covered in the original constraints are uncovered due to the upper summation bound. When combining all the backward-looking constraints (4.27) we avoid this loss of covering as in those constraints the lower summation bound only depends on the duration of the relevant operation.

Proposition 5.2.1. *The following constraints are valid for* $\operatorname{conv}(X_n)$*:*

$$\sum_{i\in\Delta} \sum_{s=t-L_i+1}^{t} y_i^s \leq 1 \qquad\qquad \Delta \in \mathcal{X}^*,\ t\in\mathcal{H} \qquad (5.1)$$

For $|\Delta| > 2$ *the inequalities are stronger than* (4.27)*.*

Proof. Consider some $\Delta \in \mathcal{X}^*$ and $t \in \mathcal{H}$. For each $i \in \Delta$ let $x_i = \sum_{s=t-L_i+1}^{t} y_i^s$. Then x_i is binary due to (4.1) and equation (5.1) reads $\sum_{i\in\Delta} x_i \leq 1$. By using the Chvátal-Gomory procedure described in Section 2.3 on (4.27) we end up with (5.1). This inequality is then valid, and for $|\Delta| > 2$ it is stronger than (4.27). Using the new inequality instead of (4.27) cuts of, e.g., solutions where $x_i = \frac{1}{2}$ for each $i \in \Delta$. $\square$

This result shows that when the non-simultaneous operations are given as maximal sets of operations rather than pairs of operations we can exploit this and gain a stronger formulation for X_n. It follows as we then work with maximal cliques in a graph, not just pairs of nodes. A closer look at (5.1) shows that it is of the same form as the time-indexed resource capacity constraint (4.3). For each $\Delta \in \mathcal{X}^*$ we create a new fictive resource r_Δ which has a constant capacity of one, $C_{r_\Delta}^t = 1$ for all $t \in \mathcal{H}$. This new resource r_Δ is added to $\mathcal{R}$. Now, for each $i \in \Delta$ we create a demand of one for this resource, $R_{ir_\Delta} = 1$. All other operations $j \notin \Delta$ have $R_{jr_\Delta} = 0$. In this way we can view the non-simultaneous operations constraints as resource capacity constraints. This point of view did not become clear to us until after the three original time-indexed formulations were investigated and compared.

A new Big-M formulation

As seen above the time-indexed formulation (4.27) of the nonops constraints was strengthened by considering sets Δ of nonops rather than pairs. This result lead us to question ourselves: Can the same be done for the Big-M formulation? The Big-M formulation of the nonops constraints (4.20)-(4.22) uses the linear ordering variables z_{ij} and can be used for both continuous and time-indexed formulations. In the latter case we just include (4.2) or use (4.23)-(4.24).

Let $\Delta = \{1, \dots, k\}$ be a set of operations that should all be pairwise non-simultaneous and let Π_Δ be the set of all permutations of Δ. Such a permutation is denoted $\pi = (\pi(1), \dots, \pi(k)) \in \Pi_\Delta$. We introduce the binary variable $z_\pi = z_{\pi(1)\pi(2)\dots\pi(k)}$ which is equal to one if the operations in Δ are permuted like π, i.e., operation $\pi(1)$ is performed first, then $\pi(2)$ and so on, as the operations in Δ cannot overlap in the schedule. For each pair $i \neq j \in \Delta$ let Π_Δ^{ji} be the set of all permutations where j precedes i, so $\Pi_\Delta^{ji} = \{\pi \in \Pi_\Delta : \pi(j) < \pi(i)\}$.

Proposition 5.2.2. *The following constraints are valid for* $\operatorname{conv}(X_n)$ *and can replace* (4.20),(4.23)-(4.24)*:*

$$\sum_{\pi \in \Pi_\Delta} z_\pi = 1 \qquad \Delta \in \mathcal{X}^* \tag{5.2}$$

$$\sum_{t=0}^{H} t y_i^t \geq \sum_{t=0}^{H} t y_j^t + L_j - H\Big(1 - \sum_{\pi \in \Pi_\Delta^{ji}} z_\pi\Big) \qquad \Delta \in \mathcal{X}^*,\ i \neq j \in \Delta \tag{5.3}$$

Proof. For the case $|\Delta| = 2$ the constraints are equivalent to (4.20),(4.23)-(4.24), so consider the case $|\Delta| = k$ for some $k > 2$. Equation (5.2) says that exactly one permutation of Δ has to be chosen. This is valid for X_n. The Big-M constraint (5.3) is active for all permutations where j precedes i and inactive otherwise. When it is inactive the constraint for $j \neq i \in \Delta$ will be active. □

This proposition shows that (5.2)-(5.3) give an alternative formulation of the non-simultaneous operations constraints. We will call it the *permutation formulation* of the Big-M constraints to distinguish from the *pairwise formulation* with z_{ij} variables. If each pair of operations $i, j \in \mathcal{I}$ appears in at most one set of non-simultaneous operations $\Delta \in \mathcal{X}^*$ what we have done is simply to introduce the mapping $z_{ij} = \sum_{\pi \in \Pi_\Delta^{ij}} z_\pi$. Doing this comes with the cost of introducing more variables. For $|\Delta| = k$ we introduce $k!$ permutation variables z_π. This number grows extremely fast, but as $|\Delta|$ in our applications is limited to at most five the number of permutation variables is at most 120 for each Δ. The number of elements in $\mathcal{X}^*$ is also small compared to the number of operations. The hope is that, even though the number of variables increases, the permutation formulation may lead to better solver performance.

It could happen that a pair of operations $i, j \in \mathcal{I}$ is included in more than one set of non-simultaneous operations. For instance, they could both demand some heavy machinery that can process only one operation at a time due to safety regulations, and they may be processed in the same production area with limited capacity due to little floor space. In this case we can find new valid inequalities for $\text{conv}(X_n)$. Let P_n be the formulation for X_n using (5.2)-(5.3).

Corollary 5.2.3. *Let $\Delta_1, \Delta_2 \in \mathcal{X}^*$ be sets of operations that should all be pairwise non-simultaneous and assume $\Delta_1 \cap \Delta_2 = \{i, j\}$. The following constraints are P_n/X_n cutting planes:*

$$\sum_{\pi \in \Pi^{ij}_{\Delta_1}} z_\pi = \sum_{\pi \in \Pi^{ij}_{\Delta_2}} z_\pi \tag{5.4}$$

$$\sum_{\pi \in \Pi^{ji}_{\Delta_1}} z_\pi = \sum_{\pi \in \Pi^{ji}_{\Delta_2}} z_\pi \tag{5.5}$$

Example 5.2.4. *Let $\Delta_1 = \{1,2,3\},\ \Delta_2 = \{1,2,4\}$. Setting $i = 1,\ j = 2$ makes the first above equation $z_{123} + z_{132} + z_{312} = z_{124} + z_{142} + z_{412}$. This inequality expresses that if i precedes j in the chosen Δ_1 permutation it must also do this in the chosen Δ_2 permutation. The equality cuts off, e.g., some fractional solutions where $z_{213} = \frac{1}{H},\ z_{123} = \frac{H-1}{H}$ and $z_{124} = 1$ which are valid for P_n.*

We can find additional valid inequalities for $\text{conv}(X_n)$ when the *number of operations* in $\Delta_1 \cap \Delta_2$ is larger than two. Let $\delta = \Delta_1 \cap \Delta_2$ be the operations present in both sets of non-simultaneous operations and let Π_δ be the set of all permutations of δ. For each $\pi_\delta \in \Pi_\delta$ and $k = 1, 2$ let $\Pi^{\pi_\delta}_{\Delta_k}$ be the set of permutations of Δ_k that preserve the ordering of the operations in δ given in π_δ, i.e., $\Pi^{\pi_\delta}_{\Delta_k} = \{\pi \in \Pi_{\Delta_k} : \pi(i) > \pi(j)\ \forall\ i, j \in \delta,\ i \neq j \text{ s.t. } \pi_\delta(i) > \pi_\delta(j)\}$. Then the following inequality is valid for $\text{conv}(X_n)$:

$$\sum_{\pi \in \Pi^{\pi_\delta}_{\Delta_1}} z_\pi = \sum_{\pi \in \Pi^{\pi_\delta}_{\Delta_2}} z_\pi$$

Example 5.2.5. *For $\Delta_1 = \{1,2,3,4\},\ \Delta_2 = \{1,2,3,5\},\ \pi_\delta = (2,3,1)$ we get $z_{2314} + z_{2341} + z_{2431} + z_{4231} = z_{2315} + z_{2351} + z_{2531} + z_{5231}$.*

An extra set of valid inequalities for $\text{conv}(X_n)$ can be found when the *number of sets* $\Delta_1, \dots, \Delta_p$ of non-simultaneous operations that all include δ is larger than two. Using the above notation, let $\delta = \Delta_1 \cap \dots \cap \Delta_p$ for $p \geq 2$. Then the following inequality is valid for $\text{conv}(X_n)$ for all $1 \leq k < l \leq p$:

$$\sum_{\pi \in \Pi^{\pi_\delta}_{\Delta_k}} z_\pi = \sum_{\pi \in \Pi^{\pi_\delta}_{\Delta_l}} z_\pi$$

Example 5.2.6. *For $\Delta_1 = \{1,2,3,4\},\ \Delta_2 = \{1,2,3,5\},\ \Delta_3 = \{1,2,3,6\},\ \pi_\delta = (2,3,1)$ we get $z_{231a} + z_{23a1} + z_{2a31} + z_{a231} = z_{231b} + z_{23b1} + z_{2b31} + z_{b231}$ for each $a, b \in \{4,5,6\},\ a < b$.*

We implement some of these cuts and test how they affect solver performance in Chapter 7. There, we also compare how the permutation formulation of the Big-M constraints compares to the pairwise Big-M formulation

(4.20),(4.23)-(4.24) and the time-indexed resource formulation (5.1) for the nonops constraints.

As mentioned, this permutation formulation for the Big-M constraints is to our knowledge new in the literature. This holds for the world of RCPS problems, but to the best of our knowledge also for Big-M constraints in general. The idea of considering permutations of sets is obviously not new, consider e.g. the Permutation Flow-shop Scheduling Problem [NR10; ZIM21] or the Asymmetric Travelling Salesman Problem [RT12]. However, we have not found articles creating binary variables for each possible permutation and using these in the formulation as we have done here. This could be due to the exponential number of possible permutations and that there, in general, is no upper bound on the number of elements in the permutation other than some n. In our case the bound is $k = 5$. We have seen very few, but some IP formulations for permutation problems with a specified application using $k = 3$ indices, see e.g. [Dup22] and the relevant papers cited there. However, we have not seen any articles creating and using permutation variables together with Big-M constraints of the form (5.2)-(5.3).

5.3 Precedence Constraints

Precedence constraints (p) are given as start-to-start relations with a minimal time-lag of the predecessor's duration. We will study cuts for the precedence polyhedron $\mathrm{conv}(X_p)$ when using the time-indexed formulation (4.19) of the precedence constraints. Recall that the general inequality (4.1) then also holds, so we let P_p be the formulation of X_p with constraints (4.19) and (4.1).

Lemma 5.3.1. *The following inequalities are P_p/X_p cutting planes:*

$$y_j^t \leq \sum_{s=t+L_j}^{H-L_i} y_i^s \qquad i \in \mathcal{I},\ j \in \mathcal{P}_i,\ t \in \mathcal{H} \tag{5.6}$$

Proof. Let $i \in \mathcal{I},\ j \in \mathcal{P}_i$ and choose some $t \in \mathcal{H}$. We start by showing that the inequality is valid for $\mathrm{conv}(X_p)$. If $y_j^t = 0$ then (5.6) is clearly valid. If $y_j^t = 1$ (5.6) expresses that i must start sometime after $t + L_j$, which is valid by (4.19) and (4.1). To prove that (5.6) is a cutting plane we find some fractional solution in P_p that is violated by (5.6). Set $y_i^{t_i} = 1$ and let $y_j^{t_i-L_j-1} = y_j^{t_i-L_j+1} = \frac{1}{2}$. This satisfies (4.19) at equality, but violates (5.6) at the time $t = t_i - L_j + 1$. $\square$

The above inequalities are valid for X_p and thus for the feasible region $X = X_{rpndw}$ for the RCPSP as $X_{rpndw} \subseteq X_p$. In fact, they can be lifted to be even stronger cutting planes for time-indexed formulations for the RCPSP.

Proposition 5.3.2. *The following inequalities are P_p/X_p cutting planes:*

$$\sum_{s=t}^{H-L_j} y_j^s \leq \sum_{s=t+L_j}^{H-L_i} y_i^s \qquad i \in \mathcal{I},\ j \in \mathcal{P}_i,\ t \in \mathcal{H} \tag{5.7}$$

These cutting planes are stronger than (5.6).

Proof. We show this by sequentially uplifting the time-indexed variables that are not present in the valid inequalities (5.6). Choose some $i \in \mathcal{I},\ j \in \mathcal{P}_i$ and $t \in \mathcal{H}$. Note that (5.6) can be rewritten as $y_j^t + \sum_{s=0}^{t+L_j-1} y_i^s \leq 1$ due to (4.1), and that (4.19) can be written as a knapsack constraint $\sum_{s=0}^{H} s y_j^s - \sum_{s=0}^{H} s y_i^s \leq -L_j$. Let (j,s) be the index for the variable $y_j^s = y_{(j,s)}$. We order the indices for the unlifted variables as $U = ((i, t+L_j), \dots, (i, H-L_i), (j,0), \dots, (j, t-1), (j, t+1), \dots, (j, H-L_j)) = (u_1, \dots, u_p)$. In this way we start by lifting y_i^s variables before lifting y_j^s variables. When lifting y_{u_h} for $h = 1, \dots, p$ we iteratively solve the following IP:

$$
\begin{array}{lll}
\zeta_h = \max & \sum_{s=1}^{h-1} \alpha_s y_{u_s} + y_j^t + \sum_{s=0}^{t+L_j-1} y_i^s & \\
\text{subject to} & \sum_{s=1}^{h-1} a_s y_{u_s} + t y_j^t - \sum_{s=0}^{t+L_j-1} s y_i^s \leq -L_j - a_h, & \\
 & \sum_{s=0}^{H-L_k} y_k^s = 1 & k = i, j, \\
 & y_k^s \in \{0,1\} & k = i, j,\ s = 0, \dots, H - L_k,
\end{array}
$$

Set $\alpha_h = 1 - \zeta_h$.

Here, a_h is the index of y_{u_h} in the knapsack constraint. Note that this coefficient is positive for y_j^s variables and negative for y_i^s variables. The IPs can be solved by hand. When lifting y_{u_h} for $u_h = (i, t+L_j), \dots, (i, H-L_i)$ we can set $y_j^t = 1$. Then $\zeta_h = 1$ and $\alpha_h = 0$. For the h values such that u_h refers to $(j,0), \dots, (j, t-1)$ one can set $y_j^{t+L_j-1} = 1$ such that also these variables get $\alpha_h = 0$. For the last of indices $u_h = (j, t+1), \dots, (j, H-L_j)$ the only feasible solutions set $y_{u(s)} = 1$ for some variable with $\alpha_s = 0$, thereby giving an objective value of zero leading to $\alpha_h = 1$. These variables are therefore lifted. Corner cases are satisfied as only indices with $0 \leq t \leq H - L_j - L_i$ are relevant. The lifted inequality $\sum_{s=t}^{H-L_j} y_j^s + \sum_{s=0}^{t+L_j-1} y_i^s \leq 1$ is then valid, and thereby (5.7) holds. □

We have now found strong P/X cutting planes that can be used when solving the RCPSP. We will refer to inequalities of the form (5.7) as *precedence cuts*. In Chapter 6 we describe a separation algorithm and a heuristic for generating cutting planes of this type when given some fractional solution to the RCPSP formulated with the precedence constraints (4.19). We implement and run the heuristic in Chapter 7. For some specific scheduling problems the precedence cuts along with general constraints suffice to describe the convex hull of the feasible region to the problem. We shall look at such a problem now.

Complete description of a precedence polyhedron Consider a scheduling problem with one work order and a planning horizon of n. Denote the operations by $\mathcal{I} = \{1, \dots, k\}$ and a time index by j. For each operation $i \in \mathcal{I}$ we include the choice of whether or not to schedule i, i.e., we let $\sum_{j=1}^{n} y_i^j \leq 1$ instead of equal to one. Let there be precedence relations between each consecutive pair of operations $i, i+1 \in \mathcal{I}$ of the form $\sum_{j=1}^{n} j y_i^j - \sum_{j=1}^{n} j y_{i+1}^j \leq b_i$. If $b_i = -L_i$ where $L_i \geq 1$ is the duration of operation i then the relation is a finish-to-start relation, which is the type of precedence constraints contained in our RCPSP. The constraints then give rise to a chain as illustrated in Figure 3.2. If $b_i \geq 0$ then the relation is a maximal time-lag relation which expresses that

i cannot start later than b_i hours after $i+1$ starts. In both cases we assume $|b_i| + |b_{i+1}| < n$ for all $i = 1, \dots, k-2$ in order for the problem to be feasible.

Denote the feasible region of schedules for the k operations by X^k. This set can be expressed using the notation explained in the subsection on GUB inequalities in Section 2.3. There we considered a knapsack problem with two groups of variables along with GUB constraints. Consider now a case with not only two, but $k \geq 2$ groups of variables $I_1, \dots, I_k$. Each group $i = 1, \dots, k$ has only one set of indices, so $|I_i| = 1$ and $S_s = N_i$ for $s \in I_i$. For simplicity we set $N_i = N = \{1, \dots, n\}$ for all i, and we change the notation from x to y variables. Define $y = (y_1, \dots, y_k)$ where $y_i \in \{0,1\}^n$ for each $i = 1, \dots, k$. We consider the integer coefficients $a_j = j,\ j = 1, \dots, n$ for all groups and we let b be an integral vector. The set X in Section 2.3 had one knapsack constraint and two groups of variables. With k groups we introduce $k-1$ knapsack constraints, so let

$$\begin{aligned} X^k = \{y \in \{0,1\}^{kn} : \sum_{j=1}^{n} j y_i^j - \sum_{j=1}^{n} j y_{i+1}^j \leq b_i,\ i = 1, \dots, k-1, \\ \sum_{j=1}^{n} y_i^j \leq 1,\ i = 1, \dots, k\}. \end{aligned}$$

In the set X^k there are $k-1$ knapsack inequalities. By using Proposition 2.3.1 on each pair $i,\ i+1$ of groups we note that the non-trivial covers for the knapsack constraint are of the form $C = (C_i, C_{i+1}) = (\{j_i\}, \emptyset),\ C = (\{j_i\}, \{j_{i+1}\})$ or $C = (\emptyset, \{j_{i+1}\})$. Following [Wol90] and the procedure described in the proof of Proposition 5.4.4 the corresponding GUB cover inequalities give the polyhedron P^k.

$$\begin{aligned} P^k = \{y \in \mathbb{R}_+^{kn} : \sum_{j > n+b_i} y_i^j \leq 0 \text{ if } b_i < 0,\ i = 1, \dots, k-1, \\ \sum_{j \geq t} y_i^j - \sum_{j \geq t-b_i} y_{i+1}^j \leq 0,\ i = 1, \dots, k-1,\ t = b_i^+ + 1, \dots, n + b_i^-, \\ - \sum_{j \geq -b_i} y_{i+1}^j \leq -1,\ \text{if } b_i < 0,\ i = 1, \dots, k-1, \\ \sum_{j=1}^{n} y_i^j \leq 1,\ i = 1, \dots, k\} \end{aligned}$$

The inequalities in P^k are valid for X^k. The following proposition shows that if all $b_i \geq 0$ then the polyhedron P^k actually gives a complete linear description of the convex hull of X^k.

Theorem 5.3.3. *For $k \geq 2$, $\operatorname{conv}(X^k) = P^k$ when $b \geq 0$.*

Proof. Our goal is to show that the binary points in P^k are precisely X^k and that P^k is an integral polytope. The latter case is the difficult part.

As the inequalities in P^k are valid for X^k we know that $X^k \subseteq P^k \cap \{0,1\}^{kn}$. Let $y \in P^k \cap \{0,1\}^{kn}$ and suppose for contradiction that $y \notin X^k$. Let i be such that $\sum_{j=1}^n jy_i^j - \sum_{j=1}^n jy_{i+1}^j \geq b_i + 1$. There are two possible cases. If group i is chosen and group $i+1$ is not chosen we must have $\sum_{j=1}^n jy_i^j \geq b_i + 1$, but this violates $\sum_{j \geq t} y_i^j \leq 0$ for $t = b_i + 1$. If, on the other hand, both groups are chosen we let $t \geq b_i + 1$ be such that $y_i^t = 1$. Then $\sum_{j=1}^n jy_i^j - \sum_{j=1}^n jy_{i+1}^j \geq b_i + 1$ gives $\sum_{j=1}^n jy_{i+1}^j \leq t - b_i - 1$. This is equivalent to $\sum_{j \leq t - b_i - 1} y_{i+1}^j \geq 1$ which violates the constraint $\sum_{j \geq t} y_i^j - \sum_{j \geq t - b_i} y_{i+1}^j \leq 0$ in P^k as $\sum_{j \geq t} y_i^j = 1$.

We now show that P^k is an integral polytope. The case when $k = 2$ is proven in [Wol90]. We follow the same procedure given there to show that the result actually holds for any integer $k \geq 2$. Define the primal problem

$$\begin{array}{rll} \max & \sum_{i=1}^k \sum_{j=1}^n c_i^j y_i^j & \\ \text{subject to} & & \\ & \sum_{j \geq t + b_i} y_i^j - \sum_{j \geq t} y_{i+1}^j \leq 0 & i = 1, \ldots, k-1,\ t = 1, \ldots, n - b_i \\ & \sum_{j=1}^n y_i^j \leq 1 & i = 1, \ldots, k \\ & y_i^j \geq 0 & i = 1, \ldots, k,\ j = 1, \ldots, n, \end{array}$$

where $c_i^j \in \mathbb{R}$ for all j and i. In the first constraint we have shifted the summation index from what is given in P^k. Let z^k denote the optimal primal value. If we can find some integral y which gives the value z^k and a feasible solution to the dual problem which gives the value z_k we are finished by Proposition 2.2.3.

We start by finding the expression for z_k using recursion. Observe that if $k = 2$ the optimal value can be found by considering two cases; one where $\sum_{j=1}^n y_2^j = 0$ (group 2 is not chosen) and one where $\sum_{j=1}^n y_2^j = 1$ (group 2 is chosen). This is due to the following: If the sum were fractional and some $c_2^j > 0$ one would increase the sum to one. If all c_2^js are negative and all c_1^js are negative the sum would be zero, and if one or more $c_1^j > 0$ the optimal value can be obtained by setting the sum to either one or zero depending on how the coefficients' absolute values compare. The optimal value z_2 is then the maximum of the optimal values for the two cases, so we get

$$z_2 = \max \left\{ \max_{t_1 = 1, \ldots, b_1} (c_1^{t_1})^+ , \max_{t_2 = 1, \ldots, n} \left(c_2^{t_2} + \max_{t_1 = 1, \ldots, t_2 + b_1} (c_1^{t_1})^+ \right) \right\},$$

and at least one solution y that gives this value has to lie in X^2. Increasing k we see that as there are 2^k possible choices of whether or not to choose groups $1, \ldots, k$, the expression for z_k gets complicated. However, it can be calculated by the following recursion

$$\begin{cases} z_1^a = \max\limits_{t_1 = 1, \ldots, a} (c_1^{t_1})^+ & \text{for } a \geq 1, \\ z_i^a = \max \left\{ z_{i-1}^{b_{i-1}}, \max\limits_{t_i = 1, \ldots, a} f(t_i, i) \right\} & \text{for } i = 2, \ldots, k,\ a \geq 1, \\ z_k = z_k^n, & \end{cases}$$

where $f(j, i) = c_i^j + z_{i-1}^{b_{i-1} + j}$. We omit to prove this, but encourage the reader to test it by hand for a small k. There exists some binary vector y giving this value as one can set $y_i^j = 1$ if $f(j, i)$ was a part of the "path" to z_k. We now show that there exists a feasible dual solution with value z_k. The dual of the primal problem is

$$\begin{array}{rll}
\min & \sum_{i=1}^{k} \alpha_i & \\
\text{subject to} & & \\
\text{(i)} & \alpha_1 \geq c_1^j & j = 1, \dots, b_1 \\
\text{(ii)} & \alpha_1 + \sum_{s=1}^{j-b_1} u_1^s \geq c_1^j & j = b_1 + 1, \dots, n \\
\text{(iii)} & \alpha_i - \sum_{s=1}^{j} u_{i-1}^s \geq c_i^j & i = 2, \dots, k-1,\ j = 1, \dots, b_i \\
\text{(iv)} & \alpha_i - \sum_{s=1}^{j} u_{i-1}^s + \sum_{s=1}^{j-b_i} u_i^s \geq c_i^j & i = 2, \dots, k-1,\ j = b_i + 1, \dots, n \\
\text{(v)} & \alpha_k - \sum_{s=1}^{j} u_{k-1}^s \geq c_k^j & j = 1, \dots, n \\
& & \\
& \alpha_k, \alpha_i, u_i^s \geq 0 & i = 1, \dots, k-1,\ s = 1, \dots, n - b_i,
\end{array}$$

where we for all i define $u_i^s = 0$ for $s > n - b_i$. The dual problem can be found by writing out the coefficient matrix of the primal problem, marking relevant column and row indices and transposing the matrix. Consider the following non-negative solution for all $i = 2, \dots, k-1,\ s = 1, \dots, n - b_i$:

$$\begin{aligned}
\alpha_1 &= \max_{t_1=1,\dots,b_1} (c_1^{t_1})^+, \\
u_1^s &= \max_{t_1=1,\dots,b_1+s} (c_1^{t_1})^+ - \max_{t_1=1,\dots,b_1+s-1} (c_1^{t_1})^+, \\
\alpha_i &= \max_{t_i=1,\dots,b_i} f(t_i, i) - \sum_{s=1}^{i-1} \alpha_s, \\
u_i^s &= \max_{t_i=1,\dots,b_i+s} f(t_i, i) - \max_{t_i=1,\dots,b_i+s-1} f(t_i, i), \\
\alpha_k &= z_k - \sum_{s=1}^{k-1} \alpha_s.
\end{aligned}$$

From the value of α_k we see that this solution gives value z_k. We show the validity of the dual constraints by fixing an $i = 2, \dots, k-1$ and a j within the relevant range for each constraint. Constraint (i) is valid by construction. For the remaining constraints we will frequently observe the telescoping sum $\sum_{s=1}^{j} u_i^s = \max_{t_i=1,\dots,b_i+j} f(t_i, i) - \max_{t_i=1,\dots,b_i} f(t_i, i)$, and that $\sum_{s=1}^{i-1} \alpha_s = \max_{t_{i-1}=1,\dots,b_{i-1}} f(t_{i-1}, i-1)$.

$$\text{(ii)}\text{: } \alpha_1 + \sum_{s=1}^{j-b_1} u_1^s = \alpha_1 + \max_{t_1=1,\dots,j} (c_1^{t_1})^+ - \alpha_1 \geq c_1^j,$$

$$\begin{aligned}
\text{(iii)}\text{: } \alpha_i - \sum_{s=1}^{j} u_{i-1}^s &= \max_{t_i=1,\dots,b_i} f(t_i, i) - \sum_{s=1}^{i-1} \alpha_s - \max_{t_{i-1}=1,\dots,b_{i-1}+j} f(t_{i-1}, i-1) \\
&\quad + \max_{t_{i-1}=1,\dots,b_{i-1}} f(t_{i-1}, i-1) \\
&\geq c_i^j + z_{i-1}^{b_{i-1}+j} - \max_{t_{i-1}=1,\dots,b_{i-1}+j} f(t_{i-1}, i-1) \\
&\geq c_i^j,
\end{aligned}$$

$$\begin{aligned}
\text{(iv): } \alpha_i - \sum_{s=1}^{j} u_{i-1}^s + \sum_{s=1}^{j-b_i} u_i^s &= \max_{t_i=1,\dots,b_i} f(t_i,i) - \sum_{s=1}^{i-1}\alpha_s - \max_{t_{i-1}=1,\dots,b_{i-1}+j} f(t_{i-1},i-1)\\
&\quad + \max_{t_{i-1}=1,\dots,b_{i-1}} f(t_{i-1},i-1)\\
&\quad + \max_{t_i=1,\dots,j} f(t_i,i) - \max_{t_i=1,\dots,b_i} f(t_i,i)\\
&\geq c_i^j + z_{i-1}^{b_{i-1}+j} - \max_{t_{i-1}=1,\dots,b_{i-1}+j} f(t_{i-1},i-1)\\
&\geq c_i^j,
\end{aligned}$$

$$\begin{aligned}
\text{(v): } \alpha_k - \sum_{s=1}^{j} u_{k-1}^s &= z_k - \sum_{s=1}^{k-1}\alpha_s - \max_{t_{k-1}=1,\dots,b_{k-1}+j} f(t_{k-1},k-1)\\
&\quad + \max_{t_{k-1}=1,\dots,b_{k-1}} f(t_{k-1},k-1)\\
&= z_k - \max_{t_{k-1}=1,\dots,b_{k-1}+j} f(t_{k-1},k-1)\\
&\geq \max_{t_k=1,\dots,n} f(t_k,k) - \max_{t_{k-1}=1,\dots,b_{k-1}+j} f(t_{k-1},k-1)\\
&\geq c_k^j + z_{k-1}^{b_{k-1}+j} - \max_{t_{k-1}=1,\dots,b_{k-1}+j} f(t_{k-1},k-1)\\
&\geq c_k^j.
\end{aligned}$$

Since all dual constraints are satisfied we have found a feasible dual solution with the same value as the primal value obtained by an integral primal feasible solution. As this holds for any objective coefficients $c_i^j \in \mathbb{R}$ we must have a complete linear description of $\text{conv}(X^k)$.

□

Remark 1 (What about $b < 0$?)**.** *The above theorem is shown for the case $b \geq 0$. Without being able to formally prove it we believe that the theorem also holds for the case when $b < 0$. When trying to prove this case we followed the procedure in the above proof. If we let $L_i = -b_i > 0$ a primal problem is given by*

$$\begin{array}{rll}
\max & \sum_{i=1}^{k}\sum_{j=1}^{n} c_i^j y_i^j & \\
\text{subject to} & & \\
& \sum_{j\geq t} y_i^j - \sum_{j\geq t+L_i} y_{i+1}^j \leq 0 & i=1,\dots,k-1,\ t=1,\dots,n-L_i\\
& \sum_{j\geq n-L_i+1} y_i^j \leq 0 & i=1,\dots,k-1\\
& -\sum_{j\geq L_i} y_{i+1}^j \leq -1 & i=1,\dots,k-1\\
& \sum_{j=1}^{n} y_i^j \leq 1 & i=1,\dots,k\\
& y_i^j \geq 0 & i=1,\dots,k,\ j=1,\dots,n,
\end{array}$$

where $c_i^j \in \mathbb{R}$ for all j and i. Let $L_a^b = \sum_{i=a}^{b} L_i$ if $a \leq b$ and let $L_a^b = 0$ otherwise. If we for feasibility assume that $L_i \leq n - L_{i+1}^{k-1}$ for all $i = 2,\dots,k$, the optimal primal value $\zeta_k \in \mathbb{R}$ is given by the formula

$$\begin{cases} \zeta^a_{k+1,k} = 0 & \text{for } a \geq 1, \\ \zeta^a_{i,k} = \max\limits_{t_i = a,\dots,n-L_i^{k-1}} g(t_i, i) & \text{for } i = 1,\dots,k,\ a \geq 1, \\ \zeta_k = \max\left\{\zeta^{L_1}_{2,k}, \zeta^1_{1,k}\right\}, \end{cases}$$

where $g(j,i) = c_i^j + \zeta^{j+L_i}_{i+1,k}$. The dual problem is given by

$$\begin{array}{rll}
\min & \sum_{i=1}^k \alpha_i - \sum_{i=2}^k \gamma_i & \\
\text{subject to} & & \\
\text{(i)} & \alpha_1 + \sum_{s=1}^j u_1^s \geq c_1^j & j = 1,\dots,n-L_1, \\
\text{(ii)} & \alpha_1 + \beta_1 + \sum_{s=1}^j u_1^s \geq c_1^j & j = n-L_1+1,\dots,n, \\
\text{(iii)} & \alpha_i + \sum_{s=1}^j u_i^s \geq c_i^j & i = 2,\dots,k-1,\ j = 1,\dots,L_{i-1}-1, \\
\text{(iv)} & \alpha_i - \gamma_i - \sum_{s=1}^{j-L_{i-1}} u_{i-1}^s + \sum_{s=1}^j u_i^s \geq c_i^j & i = 2,\dots,k-1,\ j = L_{i-1},\dots,n-L_i, \\
\text{(v)} & \alpha_i + \beta_i - \gamma_i - \sum_{s=1}^{j-L_{i-1}} u_{i-1}^s + \sum_{s=1}^j u_i^s \geq c_i^j & i = 2,\dots,k-1,\ j = n-L_i+1,\dots,n \\
\text{(vi)} & \alpha_k \geq c_k^j & j = 1,\dots,L_{k-1}-1 \\
\text{(vii)} & \alpha_k - \gamma_k - \sum_{s=1}^{j-L_{k-1}} u_{k-1}^s \geq c_k^j & j = L_{k-1},\dots,n \\
& & \\
& \alpha_1, \alpha_i, \gamma_i \geq 0 & i = 2,\dots,k \\
& \beta_i, u_i^s \geq 0 & i = 1,\dots,k-1,\ s = 1,\dots,n-b_i,
\end{array}$$

where we for all i define $u_i^s = 0$ for $s > n - L_i$. Unfortunately, we were not able to find a non-negative dual feasible solution that gives the value ζ_k within the time we anticipated to spend on this issue. The quest of finding some valid solution with this value, or to conclude that no such exists remains at the top of our list of further work. We have not tried to give a proof of the theorem for the general case when $b \in \mathbb{Z}^k$ and remain curious about whether the theorem also holds in this case.

The above precedence polyhedron arose from a time-indexed formulation. In a continuous formulation with the start time variables t_i for each $i = 1,\dots,k$ the precedence constraints are of the form $t_i - t_{i+1} \leq b_i$ for $i = 1,\dots,k-1$. The coefficient matrix of these constraints is the transpose of the node-arc incidence matrix for a directed acyclic graph. Such matrices are totally unimodular, see e.g. [Sch86]. Integral box constraints for each t_i preserve this property, and the corresponding polyhedron is therefore integral.

We now return to the general RCPSP and finish this section by observing that several binary y_i^t variables may be fixed to zero when using a time-indexed formulation of the precedence constraints. Consider first a case with chained precedence relations between some (not necessarily all) operations.

Lemma 5.3.4. *Let $\{1,\dots,k\} \subseteq \mathcal{I}$ where $k \geq 2$ be a set of operations with precedence relations between each consecutive pair of operations. Let $L_a^b = \sum_{i=a}^b L_i$ and assume $L_1^k \leq H$. The following constraints are valid for* $\operatorname{conv}(X_p)$*:*

$$\sum_{t=L_1^{i-1}}^{H-L_i^k} y_i^t = 1 \qquad i = 1,\dots,k \tag{5.8}$$

Proof. Let $i \in \{1, \ldots, k\}$. Validity of the lower summation bound is seen by inserting (4.19) recursively for $j = i-1$ down to $j = 1$ and fixing $\sum_{t=0}^{H} t y_1^t = 0$. For the upper bound, insert (4.19) recursively from $j = i$ up to $j = k-1$ and fix $\sum_{t=0}^{H} y_k^t = H - L_k$. □

The valid inequality (5.8) is stronger than the general inequality (4.1) as it contains fewer variables on the left side. For this reason it can be used a priori to solving the problem to fix the y_i^s variables with $s < L_1^{i-1}$ or $s > H - L_i^k$ to zero. This may speed up the solution process.

Lemma 5.3.4 can be generalized to find valid inequalities when the precedence relations do not give rise to chains. In general an operation i may have more than one predecessor. The earliest start of i must then be after the maximum of the earliest finish times of the predecessors. If i has many successors the latest finish of i must be before the minimum of the latest starts of the successors.

Proposition 5.3.5. *Let $i \in \mathcal{I}$. Define $t_{MIN}(i) = \max_{j \in \mathcal{P}_i}(t_{MIN}(j) + L_j)$ if $|\mathcal{P}_i| > 0$ and $t_{MIN}(i) = 0$ otherwise. Define also $t_{MAX}(i) = \min_{j \in \mathcal{S}_i}(t_{MAX}(j) - L_i)$ if $|\mathcal{S}_i| > 0$ and $t_{MAX}(i) = H - L_i$ otherwise. The following constraint is valid for* $\text{conv}(X_p)$:

$$\sum_{t=t_{MIN}(i)}^{t_{MAX}(i)} y_i^t = 1 \tag{5.9}$$

Proof. Follows from the same reasoning as in the proof of Lemma 5.3.4. □

We note that results similar to the above corollary can be obtained in a continuous formulation with the start time variables. In this case we get $t_{MIN}(i) \leq t_i \leq t_{MAX}(i)$ where the lower bound is obtained by finding the longest path from a dummy start node to i in a directed acyclic activity-to-node graph, and the upper bound by the longest path from i to a dummy end node. See Application 1.4 in [Sch17] for a further explanation of this procedure.

5.4 Delay Constraints

This section describes valid inequalities and cutting planes for the feasible region for the RCPSP with only delay constraints X_d. We start by considering the time-indexed formulation P_d for X_d consisting of the general constraint (4.1) and the constraint (4.32) for the continuous delay variable along with non-negativity constraints. Recall the assumption that for all operations i the integer valued due date is at least as late as the integer valued duration, i.e., $T_i \geq L_i$.

Proposition 5.4.1. *The following inequalities are valid for* $\text{conv}(X_d)$:

$$d_i \geq \sum_{t=0}^{H-T_i} t y_i^{T_i - L_i + t} \qquad i \in \mathcal{I} \tag{5.10}$$

Let $i \in \mathcal{I}$. If $T_i > L_i$ the inequality is a P_d/X_d cutting plane.

Proof. Let $i \in \mathcal{I}$. For validity, observe that as (4.32) is valid the weaker inequality $d_i \geq \sum_{t=T_i-L_i}^{H-L_i} ty_i^t + L_i - T_i$ is also valid. By shifting the summation index this gives $d_i \geq \sum_{t=0}^{H-T_i}(T_i - L_i + t)y_i^{T_i-L_i+t} + L_i - T_i$. Combining this with $d_i \geq 0$ gives (5.10). To show that the inequality is a cutting plane fix $d_i = 0$ and consider the fractional solution $y_i^{T_i-L_i+1} = y_i^{T_i-L_i-1} = \frac{1}{2}$. This is valid for P_d as $d_i = 0 \geq 0$, but violates (5.10) as $0 < \frac{1}{2}$. □

The d_i variables are continuous and the y_i^t variables are binary. The delay constraint $d_i \geq \sum_{t=0}^{H} ty_i^t + L_i - T_i$ is called a *mixed knapsack inequality*. For such inequalities there exist a known family of cutting planes called *mixed integer rounding* (MIR) inequalities. As these general cutting planes often are generated by MIP solvers in the presolve phase they will not be inspected further here. We refer to [Wol06] for more information on such cutting planes.

Delay constraints and nonops Valid inequalities for the set of feasible solutions to the RCPSP may be generated when we combine the delay constraints (4.32) with other constraints. Consider the set X_{nd} of feasible schedules satisfying the delay constraints and non-simultaneous operations constraints. We let P_{nd} be the time-indexed formulation consisting of (4.1), (4.32) and (5.1).

Lemma 5.4.2. *Let $\Delta = \{1, \ldots, k\} \subseteq \mathcal{I}$ be a set of operations that should all be pairwise non-simultaneous, and assume $L_i = L$ and $T_i = T$ for all $i \in \Delta$. The following inequality is valid for* $\operatorname{conv}(X_{nd})$*:*

$$\sum_{i \in \Delta} d_i \geq kL - T. \tag{5.11}$$

Assume $kL \leq H$. If $\frac{k+1}{2}L \leq T < kL$ the inequality is a P_{nd}/X_{nd} cutting plane.

Proof. For validity, observe that as the operations in Δ must be spread in the schedule the earliest makespan for the processing of these operations is kL. For $T < kL$ some operations must then be delayed, causing the minimum total delay to be $kL - T$. To observe that the inequality is a cut we find some fractional solution in P_{nd} which is violated by (5.11). For every $i \in \Delta$ fix $d_i = 0$ and set $y_i^t = \frac{1}{k}$ for each $t \in \{0, L, \ldots, (k-1)L\}$. If $0 < kL - T$ this solution violates (5.11). To show that the solution is valid for P_{nd} observe that the general constraints (4.1) and nonops constraint (5.1) are satisfied. The delay constraint (4.32) gives $0 = d_i \geq \sum_{s=0}^{k-1} \frac{sL}{k} + L - T = \frac{L}{k}\frac{k(k-1)}{2} + L - T = \frac{k+1}{2}L - T$ which is valid when $T \geq \frac{k+1}{2}L$. □

As the non-simultaneous operations must be spread in the schedule this will lead to lower bounds on the (weighted) sum of the d_i variables. The above cut concerned the simple case when all nonops had the same duration and due date. We inspect how the result can be generalized when this is not the case.

Proposition 5.4.3. *Let $\Delta = \{1, \ldots, k\} \subseteq \mathcal{I}$ be a set of operations that should all be pairwise non-simultaneous, and let $T_{MAX} = \max_{i \in \Delta} T_i$. The following inequality is valid for* $\operatorname{conv}(X_{nd})$:

$$\sum_{i\in\Delta} d_i \geq \sum_{i\in\Delta} L_i - T_{MAX}. \tag{5.12}$$

Assume $kL_{MAX} \leq H$. If $\frac{k+1}{2}L_{MIN} \leq T_{MAX} < \sum_{i\in\Delta} L_i$ the inequality is a P_{nd}/X_{nd} cutting plane.

Proof. For validity, observe that if $T_{MAX} \geq \sum_{i\in\Delta} L_i$ the inequality holds as every $d_i \geq 0$. Otherwise, assume for simplicity that the operations in Δ are ordered as $(1,\ldots,k)$. Operation k is then delayed by $d_i \geq \sum_{i\in\Delta} L_i - T_k \geq \sum_{i\in\Delta} L_i - T_{MAX}$, so (5.12) must hold. To observe that the inequality is a cut, follow the same procedure as in the proof of Lemma 5.4.2 by fixing $d_i = 0$ and setting $y_i^t = \frac{1}{k}$ for each $t \in \{0, L_i, \ldots, (k-1)L_i\}$ for each $i \in \Delta$.

□

The above results show that cuts can be found when the delay constraints are combined with nonops constraints. However, we have only found fractional solutions which are cut off for some restricted values of L_i and T_i and the planning horizon H. Trying to generalize the results to hold for any parameter values became hard. The same was the case when trying to find valid inequalities for a weighted sum of delay variables $\sum_{i\in S} w_i d_i \geq LB$ of some subset $S \subseteq \mathcal{I}$ and generalized or variable lower bound LB. Cuts of this form were returned by cut generation software as strong cutting planes. We believe that such cuts arise from finding infeasible solutions to small sub-problems of the original problem, i.e., we only have some subset of the operations $\mathcal{I}' \subset \mathcal{I}$, time points $\mathcal{H}' \subset \mathcal{H}$ and constraints. As all problem instances differ in constraints and parameter values it can be hard to write such constraints on a general form. In the future we hope to investigate relevant sub-problems further to find more valid inequalities and cutting planes for conv(X_d).

Binarizing the delay variable We now consider the formulation P_d for X_d using the binary time-indexed variables w_i^t and constraints (4.33)-(4.34), as usual along with the general constraint (4.1) and relevant non-negativity constraints. Let $n = (H+1)|\mathcal{I}|$. We then get $X_d = P_d \cap (\{0,1\}^n \times \{0,1\}^n)$ where

$$\begin{aligned} P_d = \{(y,w) \in \mathbb{R}_+^n \times \mathbb{R}_+^n : \sum_{t=0}^{H} t w_i^t \geq \sum_{t=0}^{H} t y_i^t + L_i - T_i \qquad & i \in \mathcal{I}, \\ \sum_{i=0}^{H} y_i^t = 1,\ \sum_{t=0}^{H} w_i^t = 1 \qquad & i \in \mathcal{I}, \\ y_i^t = 0 \qquad i \in \mathcal{I},\ H - L_i < t \leq H&, \\ w_i^t = 0 \qquad i \in \mathcal{I},\ H - T_i < t \leq H&\}. \end{aligned}$$

Note that with this formulation the RCPSP is a fully binary programming problem. The next proposition shows that with this formulation we can find cutting planes similar to the precedence cuts (5.7).

Proposition 5.4.4. *The following inequalities are P_d/X_d cutting planes:*

$$\sum_{s=t}^{H-L_i} y_i^s \leq \sum_{s=t+L_i-T_i}^{H-T_i} w_i^s \qquad i \in \mathcal{I},\ t \in \mathcal{H} \tag{5.13}$$

Proof. Let $i \in \mathcal{I}$. We will use Proposition 2.3.1, though another approach would be to lift inequalities like in the proof of Proposition 5.3.2. We start by showing that the delay constraint (4.34) can be written as the knapsack constraint in the set X in Proposition 2.3.1. Let group 1 refer to the y_i^s variables and group 2 to the w_i^t variables. So, we set $x_j = y_i^j$ for $j \in N_1,\ x_j = w_i^j$ for $j \in N_2$, and then fix $N_1 = N_2 = \{0, \ldots, H\}$. This gives $|I_1| = |I_2| = 1$. Setting $a_j = j$ for all j for both groups and $b = T_i - L_i \geq 0$ then leads to the knapsack constraint in X. We note that including the variables with coefficient $a_j = 0$ does not change this. Nor does changing the GUB inequalities from upper bounds to equalities, i.e., from ≤ 1 to $= 1$.

We look for GUB covers of the knapsack constraint (4.34). The covers $C = (C_1, C_2)$ are

$$\begin{array}{ll} (\{\alpha\}, \emptyset) & \alpha = T_i - L_i + 1, \ldots, H, \\ (\{T_i - L_i + \beta\}, \{\gamma\}) & \beta = 1, \ldots, H - T_i + L_i,\ \gamma = 0, \ldots, \beta - 1, \\ (\emptyset, \{\delta\}) & \delta = 0, \ldots, L_i - T_i - 1. \end{array}$$

Due to the fact that some variables are fixed to zero and that $T_i - L_i \geq 0$ the only non-redundant, non-dominated valid inequalities arising from Proposition 2.3.1 with these covers are $\sum_{s=t}^{H} y_i^s \leq \sum_{s=t+L_i-T_i}^{H} w_i^s,\ t \in [T_i - L_i + 1, H]$, which equals (5.13) when we fix variables to zero like in P_d.

As Proposition 2.3.1 only shows validity and we have not considered any unlifted weaker cutting planes it remains to show that the inequalities (5.13) actually are cutting planes. This can be done by observing that solutions with $w_i^s = 1$ for $s \geq 1$ and the fractional values $y_i^{s-L_i+T_i-1} = y_i^{s-L_i+T_i+1} = \frac{1}{2}$ are cut off by (5.13) at $t = s - L_i + T_i + 1$.

□

5.5 Resource Constraints

We finish this chapter by investigating the resource capacity constraints. Recall the time-indexed constraint (4.3):

$$\sum_{i \in \mathcal{I}} \sum_{s=t-L_i+1}^{t} R_{ir} y_i^s \leq C_r^t \qquad t \in \mathcal{H},\ r \in \mathcal{R}$$

In the following we look at one such constraint, so fix $r \in \mathcal{R}$ and $t \in \mathcal{H}$. Let $|\mathcal{I}| = n,\ N = \{1, \ldots, n\},\ x_j = \sum_{s=t-L_j+1}^{t} y_j^s \in \{0, 1\}$ and $a_j = R_{jr} \in \mathbb{N}_+$ for $j \in N$. Let also $b = C_r^t \in \mathbb{N}_+$. Then (4.3) is of the form

$$\sum_{j \in N} a_j x_j \leq b,$$

which is a knapsack constraint. In our case many of the a_j coefficients will be zero when considering resources that few operations demand. It could also happen that $a_j > b$ as the resource availability may be low some times in the schedule. We ignore the latter case and consider knapsack constraints where $0 \leq a_j \leq b$ for all $j \in N$.

It is beyond the scope of this thesis to try to find new valid inequalities for special single 0-1 knapsack polytopes P relevant to our RCPSP. Without any further discussion of the content we refer to some results from the literature that can be of relevance:

- If $a_j = 1$ or $a_j \in [\lfloor \frac{b}{3} \rfloor + 1, \lfloor \frac{b}{2} \rfloor]$ for all $j \in N$ a complete linear description of P is known. The same holds when $a_j = 1$ or $a_j \in [\lfloor \frac{b}{2} \rfloor + 1, \lfloor b \rfloor]$ for all $j \in N$ [Wei96].
- If $a_j \in \{1, p\}$ for all $j \in N$ a complete linear description of P is known [DF03].

For more results see the survey of Hojny et al. [Hoj+20]. We note also the important result that when given a minimal cover C, any lifted cover inequality $\sum_{j \in C} x_j + \sum_{j \in N \setminus C} \alpha_j x_j \leq |C| - 1$ that defines a facet of P can be obtained by sequential lifting if and only if the coefficients of the unlifted variables are integral valued [Zem89]. Integrality of all a_j is the case for our RCPSP.

We observed that several cutting planes returned from Xpress were clique inequalities involving variables for operations with demand for the same resource. By using simple integer rounding we can obtain a CG cut arising from one knapsack constraint only. Let $P = \{x \in \{0,1\}^n : \sum_{j \in N} a_j x_j \leq b\}$.

Proposition 5.5.1. *Let $\lambda = \lfloor \frac{b}{2} \rfloor + 1$. Assume that $\lambda \leq \max_{j \in N}(a_j)$ and that $|\{j : a_j \geq \lambda\}| > 1$. The following inequality is valid for P:*

$$\sum_{j \in N} \left\lfloor \frac{a_j}{\lambda} \right\rfloor x_j \leq 1 \tag{5.14}$$

Example 5.5.2. *Consider the knapsack constraint*

$$x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 + 5x_6 \leq 7.$$

Then $\lambda = 4$ and $x_4 + x_5 + x_6 \leq 1$ is valid for P.

We note that the above clique cut is an extended cover inequality for a cover C with $|C| = 2$. However, not all ECIs for a cover of size two can be generated this way. In the example the ECI $x_3 + x_5 + x_6 \leq 1$ arising from $C = \{3, 5\}$ is not generated. However, when b is not too big compared to the coefficients, (5.14) finds one valid (unlifted) ECI quickly by simply dividing and rounding coefficients.

Resource constraints and nonops We inspect valid inequalities that arise when the resource capacity constraints are combined with non-simultaneous operations constraints. Let P_{rn} be a formulation of the set X_{rn} of feasible schedules satisfying the general constraint (4.1), time-indexed resource constraint (4.3) and nonops constraint (5.1), along with relevant variable bounds.

Let $\Delta \subseteq \mathcal{I}$ be a set of non-simultaneous operations. It could happen that all these operations have demand for the same (non-fictive) resource, e.g., workers who are qualified to handle the machines which due to safety regulations only can process one operation at a time. In this case we can find some P_{rn}/X_{rn} cutting planes. Let $r \in \mathcal{R}$, $t \in \mathcal{H}$ and express the corresponding resource capacity constraint on knapsack form as above. Let $J = \{j \in N : 0 < a_j \leq b\}$, $J_p = \{j \in J : a_j \geq p\}$.

Lemma 5.5.3. *Let $\Delta \subseteq J$ be a set of operations that should all be pairwise non-simultaneous. The following inequality is a P_{rn}/X_{rn} cutting plane:*

$$\sum_{j \in J_b \cup \Delta} x_j \leq 1 \tag{5.15}$$

Proof. Each pair $i, j \in J_b,\ i \neq j$ is a minimal cover for the resource capacity constraint by the definition of the set J_b. So, for $C = \{i, j\}$ we must have $\sum_{j \in C} x_j \leq 1$. The extended cover inequality $\sum_{j \in E(C)} x_j \leq 1$ is then valid for $E(C) = J_b$. By lifting a variable $k \in \Delta \setminus J_b$ we obtain $\sum_{j \in J_b} x_j + x_k \leq 1$. Non-simultaneity of the operations in Δ gives $\sum_{j \in \Delta} x_j \leq 1$, which causes the lifting coefficient of any unlifted $k \in \Delta$ to be one regardless of the lifting order. The proposition then holds. $\square$

Example 5.5.4. *Let $r \in \mathcal{R}$ and consider the knapsack inequality*

$$x_1 + 2x_2 + 3x_3 + 4x_4 + 4x_5 + a_6x_6 + a_7x_7 + a_8x_8 \leq 4,$$

which is one of the resource capacity constraints for this resource. Let $\Delta = \{6, 7, 8\}$ be non-simultaneous operations. Then (5.15) *gives*

$$x_4 + x_5 + x_6 + x_7 + x_8 \leq 1.$$

The next proposition shows that (5.15) in some cases can be strengthened. Let $a_\Delta = \min_{i \in \Delta} a_i \geq 1$, and define $\bar{b} = \max\left\{\lfloor \frac{b}{2} \rfloor + 1,\ b - a_\Delta + 1\right\}$.

Proposition 5.5.5. *Let $\Delta \subseteq J$ be a set of operations that should all be pairwise non-simultaneous. The following inequality is a P_{rn}/X_{rn} cutting plane:*

$$\sum_{j \in J_{\bar{b}} \cup \Delta} x_j \leq 1 \tag{5.16}$$

Proof. Assume for contradiction that the inequality is not valid for $\text{conv}(X_{rn})$. Let $i, j \in J_{\bar{b}} \cup \Delta$ be such that $x_i = x_j = 1$. We consider three possible cases which all will lead to a contradiction. If $i, j \in \Delta$ this violates the nonops constraint (5.1). If $i \in \Delta,\ j \notin \Delta$ then $a_i + a_j \geq a_\Delta + (b - a_\Delta + 1) > b$, which violates the knapsack constraint. Finally, if $i, j \notin \Delta$ then $a_i + a_j \geq 2(\lfloor \frac{b}{2} \rfloor + 1) > b$, which again violates the knapsack constraint. The inequality (5.16) must then be valid for $\text{conv}(X_{rn})$, and as it contains at least as many elements on the left side as (5.15) it is a cutting plane. $\square$

Example 5.5.4 (Continued)**.** *Let $a_6 = a_7 = 2,\ a_8 = 3$. Then $\bar{b} = \max\{3, 3\}$ and* (5.16) *gives*

$$x_3 + x_4 + x_5 + x_6 + x_7 + x_8 \leq 1.$$

We will implement the cutting planes (5.16) and test how they perform in Chapter 7. It can already be observed that, for these cuts to be strong, the values of the coefficients and b need to be restricted. Adding (5.16) to a formulation will probably not improve solver performance much when b is large compared to the coefficients or when $a_{\Delta} = 1$. We believe the cuts will be more efficient when Δ is large, in bottlenecks when b is low or when the demands a_j are high.

As in the above section we experienced that cut generation software generated more cuts which probably arise from a combination of resource constraints and other constraints in smaller subproblems. Although we tried, we found it hard to derive these cuts on a general form by hand.

CHAPTER 6

Separation Algorithms

Many real-life instances of hard combinatorial optimization problems are too large to be solved using exact methods with natural IP or MIP formulations of the set of feasible solutions to the problem. One way to tackle this is to search for stronger formulations of the set of feasible solutions in hope that this may reduce the time needed by the solver algorithm to find an optimal solution. Adding cutting planes to the initial formulation is one way of obtaining a stronger formulation, but to do this one needs a suitable separation algorithm to see if any inequalities are violated by the current fractional solution, and if so, decide which of these to add to the formulation. In this chapter we shall inspect how we can study a set of inequalities that we believe may work as cutting planes for formulations for the RCPSP. We will explain how the cutting planes found in Chapter 5 can be analyzed computationally and how separation algorithms for these may look.

The chapter is structured as follows: In Section 6.1 we explain a naive separation algorithm that can be used for checking if a set $\mathcal{Z}$ of valid inequalities for a set X contains any cutting planes that can be added to a current formulation P of X. This algorithm is based on full enumeration. We explain why we will need such an algorithm for testing the cutting planes found in Chapter 5. In Section 6.2 we find an exact separation algorithm for a subset of the precedence cuts found in Section 5.3. Then, in Section 6.3 we introduce an empirically based separation heuristic for the full set of these precedence cuts. This algorithm is not guaranteed to find a violated inequality, and if it finds one it does not necessarily find the most violated inequality. For this reason we call it a separation *heuristic*. When using the term *separation* in this chapter we are only interested in separating fractional points in P from $\text{conv}(X)$ using inequalities in the given set $\mathcal{Z}$.

6.1 Naive General Separation Algorithm

Consider some polyhedron $P = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$ which is a formulation for the set $X = P \cap \mathbb{Z}^n$ of integral solutions to the general IP problem (IP). Let $\mathcal{Z}$ be a set in which the elements $(a, \alpha) \in \mathcal{Z}$ refer to inequalities $a^T x \leq \alpha$ that are valid for $\text{conv}(X)$. We assume all inequalities in $\mathcal{Z}$ are of the same form, e.g., deduced from the same set of constraints. Let x^* be an optimal solution to the LP relaxation (LP) of (IP). Given $\mathcal{Z}$ we can ask ourselves:

1. Are any inequalities $(a, \alpha) \in \mathcal{Z}$ violated by x^*?
2. If so, how does adding these cutting planes to P affect the performance of the optimization solver when solving (IP)?
3. Should a separation algorithm be made for the inequalities in $\mathcal{Z}$?

These three questions are precisely the questions that we ask ourselves about the cutting planes we found in Chapter 5. Could they actually be useful when solving RCPSP instances, or are they just nice theoretical results? In order to answer question 1. we will use a naive separation algorithm which we describe in this section. We implement and use this in Chapter 7 to answer questions 2. and 3.

Definition 6.1.1 (Full enumeration). *Solving a separation problem by* full enumeration *means looping over every $(a, \alpha) \in \mathcal{Z}$ and checking if the fractional solution x^* violates (a, α).*

Solving a separation problem by full enumeration is what we call a *naive separation algorithm*. It is naive as the number of inequalities in $\mathcal{Z}$ can be large, maybe exponentially many. Using full enumeration on the set of *all* possible cutting planes is in general not a good idea, however, it can be used to obtain knowledge about the set $\mathcal{Z}$ of inequalities, and full enumeration on a small set of inequalities may be done fast. As our wish is to obtain more knowledge about the cuts found in Chapter 5 we will use this naive separation algorithm to generate a set $\mathcal{F} \subseteq \mathcal{Z}$ of cutting planes which are violated by one or more fractional solutions x^* to (LP). We do this with a cut-and-branch approach with respect to the inequalities in $\mathcal{Z}$: First, we solve the LP problem (LP) over the initial formulation P. Let x^* denote the optimal solution returned by the solver and let $\mathcal{F}$ be the set of all inequalities in $\mathcal{Z}$ that are violated by x^*. We add the inequalities in $\mathcal{F}$ to P and solve (LP) again over the new formulation. We continue to add cutting planes and solve (LP) iteratively until a solution x^* which violates no inequalities in $\mathcal{Z}$ is found. Then, we solve (IP) with the final formulation P. The algorithm is summarized in Algorithm 1. We call it a $\mathcal{Z}$ *cut-and-branch* algorithm as the cutting planes in $\mathcal{Z}$ are only added to the root node of the branching tree.

With the $\mathcal{Z}$ cut-and-branch algorithm we generate the set $\mathcal{F}$ and can inspect how adding the inequalities in $\mathcal{F}$ to P affects the solver when solving (IP) over P. If the solver time is reduced or bounds are improved when the cuts are added this could indicate that it would be smart to make a non-naive separation algorithm for the set $\mathcal{Z}$. This algorithm, which either heuristically or exactly should return an inequality in $\mathcal{Z}$ which is violated by the fractional solution x^*, can be used to add cutting planes from $\mathcal{Z}$ to the root node or at nodes as the branching tree grows. With the Gurobi Optimizer the latter can be achieved by using the separation algorithm in a *callback*. In this way Gurobi will solve the separation problem and add some violated inequality $(a, \alpha) \in \mathcal{Z}$ to P *if* and *when* the solver finds it necessary during the branching process. We believe this possibility can be beneficial to use when solving real-life instances of the RCPSP that are influenced by special types of constraints for which we know cutting planes exist. The precedence constraints are one such type of constraints.

Algorithm 1 $\mathcal{Z}$ Cut-and-Branch Algorithm

```
Require:
  (IP), (LP) = An IP and its LP relaxation
  $P$ = formulation of the set of feasible solutions $X$ to (IP)
  $\mathcal{Z}$ = set of inequalities valid for conv($X$)
Ensure:
  $\mathcal{F}$ = set of $P/X$ cutting planes
  $x^*$ = optimal solution to (IP)

  $\mathcal{F} \leftarrow \emptyset$
  $cuts \leftarrow$ TRUE
  while $cuts$ do
    $x^* \leftarrow$ optimal solution to (LP) over $P$
    $cuts \leftarrow$ FALSE
    for all $(a, \alpha) \in \mathcal{Z}$ do                ▷ Naive separation algorithm
      if $a^T x^* > \alpha$ then
        add $(a, \alpha)$ to $\mathcal{F}$
        $cuts \leftarrow$ TRUE
      end if
    end for
    $P \leftarrow P$ intersected with halfspaces in $\mathcal{F}$
  end while
  Use a branching based solver to solve (IP) over $P$
  $x^* \leftarrow$ optimal solution to (IP)
```

6.2 Exact Separation Algorithm for Precedence Cuts

In the following sections we consider the RCPSP with precedence constraints. Let X be the feasible region of (RCPSP) and let P be some formulation of X using the time-indexed formulation (4.19) for the precedence constraints. We ignore the formulation for the rest of the constraint types and simply denote a formulation for their set of feasible solutions by P'. For simplicity we use dimension n and assume that X is a binary set, and for convenience we now change the notation and denote a feasible solution by y instead of x. Thus, $X = P \cap \{0,1\}^n$ where

$$
\begin{aligned}
P = \{y \in \mathbb{R}^n_+ : \sum_{t=0}^{H} t y_i^t \geq \sum_{t=0}^{H} t y_j^t + L_j \qquad & i \in \mathcal{I},\ j \in \mathcal{P}_i, \\
\sum_{i=0}^{H} y_i^t = 1, \qquad & i \in \mathcal{I}, \\
y_i^t = 0 \qquad & i \in \mathcal{I},\ H - L_i < t \leq H, \\
y \in P'\}. &
\end{aligned}
$$

By Proposition 5.3.2 the inequalities (5.7) are valid for conv(X) and cut of fractional points valid for (4.19). Let $\mathcal{Z}$ be a set where the elements are triplets referring to these inequalities. By $(j, i, t) \in \mathcal{Z}$ we refer to the inequality

$$\sum_{s=t}^{H-L_j} y_j^s \leq \sum_{s=t+L_j}^{H-L_i} y_i^s.$$

We shall see in Chapter 7 that the number of such precedence cuts (j, i, t) generated by the naive separation algorithm often was large, and that the solver time could increase considerably when adding all violated inequalities to the initial formulation. Therefore, an alternative approach is to only add *one* violated inequality $(j, i, t) \in \mathcal{Z}$ to P. In general it is a classic approach to find and add the *most violated inequality* to the formulation, if any are violated. This problem can be formulated and solved as an integer programming problem.

Let $\mathcal{P}$ be the set of all precedence relations in the RCPSP, i.e., for each operation $i \in \mathcal{I}$ we let $(j, i) \in \mathcal{P}$ if j is a predecessor to i. Given this pair and a fractional solution y^* to the LP relaxation of the RCPSP we can find the inequality $(j, i, t) \in \mathcal{Z}$ which is most violated by y^* or conclude that no such inequality exists. Consider the following IP:

$$\begin{aligned}
\max \quad & \zeta = \textstyle\sum_{t=0}^{H} \left(\sum_{s=t}^{H} y_j^{s*}\right) z_j^t + \sum_{t=0}^{H} \left(\sum_{s=0}^{t} y_i^{s*}\right) z_i^t \\
\text{subject to} \quad & \textstyle\sum_{t=0}^{H} t z_j^t - \sum_{t=0}^{H} t z_i^t \geq -L_j + 1 \\
& \textstyle\sum_{t=0}^{H} z_j^t \leq 1, \quad \sum_{t=0}^{H} z_i^t \leq 1 \\
& z \in \{0,1\}^n
\end{aligned}$$

Recall that the (j, i, t) inequalities are a special type of the inequalities in Proposition 2.3.1. By this proposition they can be derived from a GUB cover $C = (C_1, C_2)$ which covers the knapsack inequality (4.19). In the above IP the variable z_j^t is equal to one if $t \in C_1$ and zero otherwise, and $z_i^t = 1$ if $t \in C_2$ and zero otherwise.

Proposition 6.2.1. *Let $(j, i) \in \mathcal{P}$ and y^* be a fractional point in P. If $\zeta > \sum_{t=0}^{H} y_i^{s*}$ and $z(C)$ is optimal, then the GUB cover associated with $z(C)$ gives the most violated (j, i, t) inequality. If $\zeta \leq \sum_{t=0}^{H} y_i^{s*}$ then no (j, i, t) inequality is violated by y^*.*

Proof. Consider the inequality (j, i, t) arising from the GUB cover $C = (\{t\}, \{t + L_j - 1\})$. As $\sum_{s=t+L_j}^{H-L_i} y_i^{s*} = \sum_{s=0}^{H} y_i^{s*} - \sum_{s=0}^{t+L_j-1} y_i^{s*}$ when some variables are fixed to zero in P, the inequality (j, i, t) is equivalent to

$$\sum_{s=0}^{t+L_j-1} y_i^{s*} \leq \sum_{s=0}^{H} y_i^{s*} - \sum_{s=t}^{H} y_j^{s*}.$$

Let $z(C)$ be the vector associated with C. As C is a cover it must hold that $\sum_{t=0}^{H} t z_j^t(C) - \sum_{t=0}^{H} t z_i^t(C) > -L_j$. For (j, i, t) the cover gives $z_j^t(C) = z_i^{t+L_j-1}(C) = 1$. Using this vector the above inequality is equivalent to

$$\sum_{t=0}^{H} \left(\sum_{s=0}^{t} y_i^{s*}\right) z_i^t(C) \leq \sum_{s=0}^{H} y_i^{s*} - \sum_{t=0}^{H} \left(\sum_{s=t}^{H} y_j^{s*}\right) z_j^t(C).$$

From this inequality we see that y^* violates (j,i,t) if $\sum_{t=0}^{H}\left(\sum_{s=t}^{H} y_j^{s*}\right) z_j^t(C) + \sum_{t=0}^{H}\left(\sum_{s=0}^{t} y_i^{s*}\right) z_i^t(C) > \sum_{s=0}^{H} y_i^{s*}$. Maximizing the left side over all possible covers C then returns the most violated inequality. $\square$

The above separation problem and the proof are deeply inspired by the separation problem for the general GUB cover inequalities in Proposition 2.3.1 described in Proposition 2.2 in [Wol90]. Given a pair $(j,i) \in \mathcal{P}$ of operations with a precedence relation between them the above separation problem finds the inequality (j,i,t) which is most violated by a fractional solution y^*. However, in our case we are also interested in finding the pair $(j,i) \in \mathcal{P}$ which ensures the largest violation. The maximization problem must then be expanded so that it also includes the choice of such a pair, if not, the IP must be solved for every $(j,i) \in \mathcal{P}$, which could be time-consuming. We have not expanded the separation problem to include the choice of (j,i) here, but note it as further work. Instead, we designed our own separation heuristic to find cuts $(j,i,t) \in \mathcal{Z}$ that can be added to P.

6.3 Separation Heuristic for Precedence Cuts

Consider again the set $\mathcal{Z}$ of all precedence cuts (5.7) valid for conv(X) as described above. In order to gain knowledge about the characteristics of the cuts $(j,i,t) \in \mathcal{Z}$ that were normally violated by fractional solutions y^* to LP relaxations of RCPSP instances we ran the $\mathcal{Z}$ cut-and-branch algorithm described in Section 6.1. We used several of the test instances from Appendix A.3. The number of violated inequalities was, as mentioned, normally very large, so to find the most violated ones we increased the threshold for violation. Through some trying and failing to set this threshold for the different instances we began to see a pattern in the most violated cuts $(j,i,t) \in \mathcal{Z}$. Several instances contained chained precedence relations between the operations $1,\dots,k$ in some of the work orders $w \in \mathcal{WO}$ in the problem. In this case operation i is followed by $i+1$ and we can denote the relevant precedence cuts by $(i,i+1,t)$ for $t \in \mathcal{H}$. We observed that several of the most violated inequalities were of this form and that they had

- operation i or $i+1$ included in some non-simultaneous operations constraint,
- a small gap between operation i's relative position in the work order, $\frac{i}{k}$, and t's relative position in the horizon, $\frac{t}{H}$.

These frequent observations lead to the idea of a simple separation heuristic; a heuristic that uses the above observations and the naive separation algorithm in hope of quickly finding a violated precedence cut $(j,i,t) \in \mathcal{Z}$ which should be added to the formulation P. This heuristic is not guaranteed to find any violated inequality even if one exists, and the returned inequalities are not guaranteed to be the most violated ones. The heuristic is also limited as it is inspired by results on instances where the precedence relations are chained and within one work order. Although or models for the RCPSP support precedence relations between any pair of operations, the chained precedence relations within the same work order were presented as the ones that frequently will appear in real-life instances of the RCPSP [Ind22]. For this reason we found it suitable to use this information when designing the test instances and the heuristic.

Let $v(j,i,t,y^*)$ be the violation of (j,i,t) in the current solution y^*, so

$$v(j,i,t,y^*) = \sum_{s=t}^{H-L_j} y_j^{s*} - \sum_{s=t+L_j}^{H-L_i} y_i^{s*}.$$

If $v(j,i,t,y^*) > 0$ then $(j,i,t) \in \mathcal{Z}$ is an inequality which is violated by y^*. Based on this and the observations above we design the separation heuristic given in Algorithm 2. The heuristic only considers precedence relations where the predecessor or successor is included in a non-simultaneous operations constraint. If there are no non-simultaneous operations constraints in the problem we ignore this criterion. For each precedence relation $(j,i) \in \mathcal{P}$ we let k be the number of operations in the work order that j belongs to. We add to the set $\mathcal{G}$ all inequalities (j,i,t) such that t's relative position in the horizon $\frac{t}{H}$ is no more than 10 % different from j's position in the work order, denoted $\frac{j}{k}$ for simplicity. We let v^* be the maximum violation obtained by the inequalities in $\mathcal{G}$. If $v^* > 0$ then some inequalities are violated and we add to the set $\mathcal{F}$ all inequalities $(j,i,t) \in \mathcal{G}$ that are violated by the amount v^*. To cope with numerical round-off errors we frequently use a tolerance of 10^{-6}, which is the default feasibility tolerance used by the Gurobi Optimizer. We will implement this separation heuristic and test how it affects the solver performance in Chapter 7.

Algorithm 2 Separation Heuristic for Precedence Cuts

```
Require:
  $P$ = formulation of the set of feasible solutions $X$ to (RCPSP)
  $y^*$ = solution to current LP relaxation
  $\mathcal{Z}$ = set of precedence inequalities $(j,i,t)$
Ensure:
  $\mathcal{F}$ = set of some inequalities $(j,i,t)$ violated by $y^*$

  $\mathcal{F},\ \mathcal{G} \leftarrow \emptyset$
  for $(j,i) \in \mathcal{P}$ do
      if neither $j$ or $i$ have non-simultaneous operations then continue
      end if
      $k \leftarrow$ number of operations in $j$'s work order
      $T \leftarrow \{\max(0, \lceil H(\frac{j}{k} - 0.1)\rceil), \dots, \min(H, \lfloor H(\frac{j}{k} + 0.1)\rfloor)\}$
      for $t \in T$ do
          add $(j,i,t)$ to $\mathcal{G}$
      end for
  end for
  $v^* \leftarrow \max_{(j,i,t)\in\mathcal{G}}(v(j,i,t,y^*))$
  if $v^* > 0$ then
      Add $(j,i,t)$ to $\mathcal{F}$ for all $(j,i,t) \in \mathcal{G}$ with $v(j,i,t,y^*) = v^*$
  end if
```

CHAPTER 7

Computational Results

In the previous chapters we have introduced the RCPSP and different IP and MIP formulations for the set of feasible schedules given different types of constraints. In Chapter 5 we looked at how these formulations can be strengthened by adding valid cutting planes, and in Chapter 6 we found relevant separation algorithms that can be used when solving the RCPSP. This work has led up to the main goal of this thesis, which is to inspect and compare how the choice of formulation affects the performance of the Gurobi optimization solver. Our aim is to find the formulation that overall ensures stability, high quality on lower bounds and low running time when used to solve real-life instances of the RCPSP for scheduling and rescheduling purposes.

The chapter is structured as follows: In Section 7.1 we give information on the test instances that are designed for testing and comparing the different formulations with respect to computational performance. The results are presented and discussed in Section 7.2. In Section 7.3 we present the results when solving the rescheduling problem from Section 4.6. We inspect how cutting planes affect the solver performance in Section 7.4 and discuss the results.

As this thesis is a contribution to a real project we used the optimization solver which eventually will be used by our industry partners. This is the Gurobi Optimizer version 9.5.2 [Gur22]. Some observations were also made using version 9.1.1. The solver ran a cloud license and all tests were done on an Intel Xeon Gold 6240 CPU with 2.60 GHz and 7 Cores on a Microsoft Windows Server with 20 GB RAM. All code is implemented in C# version 10.0. using the .NET 6.0 framework in Visual Studio 2022. Unless otherwise specified Gurobi used one thread and default solver settings, and the maximum solution time was set to 900 seconds.

7.1 Test Instances

From the beginning our emphasis has been on testing the different MIP models on real data from our industry partners in order to find formulations that are suitable to be implemented and used in scheduling software for consumers. From a research perspective we could have studied other, more specified instances. These could be extremely large instances that are relevant for worst-case studies, or small instances where finding an initial feasible schedule is computationally hard. We could also have tested our models on benchmark instances for

Type	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{R}\rvert$	H	# Instances
S	0–100	1–6	0–200	12
M	100–500	23	200-400	5
L	500–1000	23	400-600	8
XL	1000–1500	23	600-800	3

Table 7.1: Information on the test instances for the RCPSP

scheduling problems and the standard RCPSP. This has not been done in this thesis, but is noted as further work which will be done in the OptiPlan project.

When starting the work with this thesis we were given the impression that several real-life instances were to be handed to us for testing purposes. Unfortunately, this did not happen due to delays in the OptiPlan project. However, we were given a pseudo-random instance inspired by real data from our relevant partners. With some alterations to ensure feasibility this is the base test instance denoted as $L1$. From this instance we generated a total of 28 feasible scheduling problem instances with different sizes and constraint characteristics. See Table 7.1 for information on these instances. Recall that the symbols are explained in Appendix A.1. The test instances are categorized as either small (S), medium (M), large (L) or extra large (XL) instances. This choice is affected by the number of operations $|\mathcal{I}|$, resource types $|\mathcal{R}|$ and planning horizon H in hours. The smallest instances have a horizon of some days, while the largest instances have a horizon of up to five months. For more information on the data and constraints in each problem instance please see Table A.2 in Appendix A. There we also mention why small instances can be harder to solve than large instances.

In our applications each operation is categorized as either a general operation, production critical operation or safety critical operation. In every test instance we set the delay cost B_i of operation i to 1 if it was a general operation, $B_i = 100$ if it was production critical and $B_i = 1000$ if it was safety critical.

7.2 Comparing Formulations for the RCPSP

We start by comparing different models for (RCPSP) introduced in Chapter 4. Recall that by a *model* for the RCPSP we mean the choice of how to state the weighted delay *objective* and *formulation* of the set of feasible solutions. When comparing models there exist several ways of measuring performance. We measured the performance and quality of a model by analyzing the following four measures in prioritized order:

1. **Solution time:** What is the solution time spent by the Gurobi Optimizer? Or, if the maximum solver time is reached, what is the current duality gap? We search for a model which overall shows the tendency of ensuring low running times for the optimization software when used on real-life instances of the RCPSP.

2. **Stability:** Does the formulation perform well and as expected on all test instances, or does the solver behavior seem alternating and random when

this formulation is used? We search for a model which can be trusted to have a stable high performance on all problem instances, and which can tackle a variety of problem sizes and constraint characteristics.

3. **Lower bounds and incumbent solutions:** We search for a model which leads to a high lower bound when solving the LP relaxation of the problem. We inspect how fast an incumbent solution is found by the solver and how the gap between this upper bound and the lower bound reduces.

4. **Size of the formulation:** Should the above measures give approximately the same results for different models we prefer the one with the least number of variables and constraints as some problem instances may be very large.

Our emphasis has been on comparing time-indexed formulations for the RCPSP. When comparing different formulations for one constraint type, e.g., the non-simultaneous operations constraints, we needed default formulations for the rest of the constraints. These are given in the base model.

Definition 7.2.1 (Base model)**.** *The* base model *is the default model used in our test runs. The base model in this thesis is*

$$(o(C), Prpnw(TI)d(CTI)).$$

The above model was chosen as base model as it turned out to perform well on initial test runs. The model is a time-indexed model extended with the continuous delay variable d_i for each operation $i \in \mathcal{I}$. See Table A.1 for a reference to the mathematical constraints. In the tables in this chapter we will under the columns named "Model" only state the formulation for the constraint(s) of interest, and use the base model for the rest of the formulation and the objective.

Including the start time variables

Every time-indexed formulation for the set X of feasible schedules to the RCPSP will include the binary y_i^t variables, where y_i^t is equal to one if operation $i \in \mathcal{I}$ starts at time $t \in \mathcal{H}$ and zero otherwise. The starting time of i is given by $\sum_{t=0}^{H} ty_i^t$. Using (4.2) we can extend the formulations with the starting time variables t_i. This makes the formulations more intuitive and easier to implement, but comes with the cost of introducing new variables. The general effect this extension has on model performance is not clear, so we started by comparing how the base model performed with and without the t_i variables included. The formulations are then $Prpnw(TI)d(CTI)$ and $Prnw(TI)pd(C)$, respectively.

We ran these models on the test instances $S1$–$S4$, $M1$–$M3$ and $L1$–$L3$. Both models found an optimal solution within the time limit on all but one instance. The lower bounds were equal for both models, as well as the behavior of the solver with respect to the presolve and branching phase. The only differences we experienced were with the solution times. The base model was always fastest, and on average it was 94.2 % faster than the extended base model, causing the solver time to be cut in half. This result was and is unexpected as we initially believed that including the t_i variables would be beneficial for the solver.

Model	% Win
$o()$	67
$o(C)$	25
$o(TI)$	8

Table 7.2: Comparing formulations for the weighted delay objective

Changing the t_i variables from Gurobi continuous to Gurobi integer did not change the results. The extended base model also had more unstable behavior with respect to running time than the base model. At this stage we are not able to dive deeper into the reasons for this poor performance as that would require knowledge about the Gurobi solver which is unavailable to us. We note that the results may very well be opposite for other scheduling problem instances, and for this reason we do not wish to conclude something on a general level. However, due to the results in our tests we find it better not to include the t_i variables in our time-indexed RCPSP formulations.

Result 1. *The base model should not be extended with start time variables.*

Later in this section we extend other time-indexed models besides the base model with the start time variables. As we shall see, this did on average not improve the performance of these models either.

Objective and delay constraints

For the RCPSP we consider the weighted delay objective, also called weighted tardiness. See Section 4.4. We compare how the choice of objective formulation affects the performance of the base model. Thus, we let the formulation for X be as in the base model, and consider the objectives $o(C)$, $o(TI)$ and $o()$. This choice also affects the delay constraints, which are either $d(CTI)$ or $d(TI)$ along with non-negativity constraints, or none $d()$, respectively.

We ran the three models on the instances $S1$–$S4$, $M1$–$M3$, $L1$–$L3$ and $XL1$–$XL2$. The results are given in Table 7.2. Here, "% Win" is the number of test instances given in percent in which the models performed best with respect to running time or gap returned within the time limit. Here we see that $o()$ was fastest in 67% of the test runs. When the $o()$ model was faster than $o(C)$ the differences in running time were larger than the times where $o(C)$ was faster than $o()$. The $o()$ objective reduces the number of variables and constraints as no delay variables are included. All models performed equally on the quality of lower bounds. The $o(TI)$ model was slowest in all except one instance, and the number of constraints and variables increased by a factor of $\mathcal{O}(H|\mathcal{I}|)$ compared to the $o()$ model. Without adding cutting planes of the form (5.13) we do not recommend binarizing the delay variables and using $o(TI)$. And, as long as the d_i variables are not needed for other purposes or cutting planes we recommend using the $o()$ objective and to not introduce delay variables or constraints.

Result 2. *Time-indexed models should use the $o()$ objective and not include delay variables or constraints.*

Early on in the testing process we tried including the schedule makespan $m \geq 0$ in the objective by adding the fraction $\frac{m}{H+1}$ which is strictly between zero

Model	% BLB	% Win
$n(forward)$	100	0
$n(middle)$	87.5	12.5
$n(backward)$	100	50
$n(BMTI)$	75	37.5

Table 7.3: Comparing pairwise formulations for the nonops constraints

and one. In this way the solver would prioritize the schedule with the earliest makespan among the solutions with optimal integer valued weighted delay. Adding this fraction to the objective increased the running times considerably, and as an early makespan was not important for our industry partners we decided to not include makespan as a part of the objective.

Non-simultaneous operations constraints

The non-simultaneous operations constraints express that two given operations $i, j \in \mathcal{I}$ cannot overlap in the schedule. As we have seen in Chapter 4 and Chapter 5 these constraints can be formulated in different ways. See these formulations in Table A.1.

We started by comparing the formulations when the nonops are given as sets of pairs $\{i, j\} \in \mathcal{X}$. We ran the formulations $n(forward)$, $n(middle)$, $n(backward)$ and $n(BMTI)$ on the 15 instances $S1$–$S4$, $S8n$–$S10n$, $S12r$, $M1$–$M3$, $M4n$ and $L1$–$L3$. The results are given in Table 7.3. Here ”% BLB” is the number of test instances given in percent where the models returned the highest value of the LP relaxations among all the models, and ”% Win” is as above. Overall we experienced little difference between the first three models. The $n(middle)$ model returned a poorer lower bound than $n(forward)$ and $n(backward)$ on one instance, causing the formulation to be weaker, and the model was on average a bit slower than the other models. This confirmed our belief in Section 5.2. With respect to running time the Big-M model $n(BMTI)$ and the $n(backward)$ model alternated at being best. As we know that formulations using $n(backward)$ can be strengthened using (5.1) we will test how this formulation compares to $n(BMTI)$ next. When it is the case that the nonops are given as pairs $\{i, j\} \in \mathcal{X}$ rather than as larger sets Δ, we recommend using the Big-M formulation as this overall performed well. In real-life schedules it may be a small probability that i and j actually do overlap. Delayed row and column generation may then be used to create the z_{ij} variables and to add the Big-M constraints to the formulation only if they are violated. We did not try this, but note it as further work.

We now consider the case when we know the nonops are given as maximal sets $\Delta \in \mathcal{X}^*$ with sizes between two and five. We ran the pairwise Big-M formulation $n(BMTI)$, our permutation formulation $n(BMP)$ and the strongest time-indexed formulation $n(TI)$ where the nonops are represented as operations demanding some dedicated resource. We ran the instances $S3$, $S8n$–$S9n$, $M3$, $M4n$, $L3$, $L5$, $L6n$-$L7n$ and $XL2$. The results are given in Table 7.4. Here we state the problem instance ”Instance”, the model ”Model”, and we give the number of constraints and variables in the formulations. We report the number

Instance	Model	Cons	Vars	Nodes	Sol
$S3$	$n(BMTI)$	569	1100	1	0.07s
	$n(BMP)$	513	1100	1	0.19s
	$n(TI)$	1238	950	1	0.05s
$S8n$	$n(BMTI)$	569	1100	5976	6s
	$n(BMP)$	513	1226	153015	157s
	$n(TI)$	1238	950	1	0.80s
$S9n$	$n(BMTI)$	389	992	13	0.17s
	$n(BMP)$	373	1016	1	0.20s
	$n(TI)$	566	950	1	0.06s
$M3$	$n(BMTI)$	9580	181200	1	9s
	$n(BMP)$	9499	181444	1	6s
	$n(TI)$	16120	181000	1	7s
$M4n$	$n(BMTI)$	9591	177330	1	39s
	$n(BMP)$	9449	178362	1	52s
	$n(TI)$	17192	177000	1	64s
$L3$	$n(BMTI)$	16140	720388	1	44s
	$n(BMP)$	16059	720632	1	65s
	$n(TI)$	26784	720188	1	38s
$L5$	$n(BMTI)$	12333	434198	18	-
	$n(BMP)$	12253	434518	26	4.6%
	$n(TI)$	22267	434000	20	6.9%
$L6n$	$n(BMTI)$	12450	434276	22	6.3%
	$n(BMP)$	12344	434596	22	4.0%
	$n(TI)$	25860	434000	31	4.2%
$L7n$	$n(BMTI)$	12450	434276	98	14%
	$n(BMP)$	12344	434596	11	8.8%
	$n(TI)$	25860	434000	105	9.6%
$XL2$	$n(BMTI)$	20928	1167196	1	98s
	$n(BMP)$	20852	1167340	1	100s
	$n(TI)$	37706	1167000	1	116s

Table 7.4: Comparing the resource formulation of nonops constraints with the Big-M formulations

of nodes visited in the branching tree, and in the column ”Sol” the time spent to obtain an optimal solution is given. Very minor differences in running times are considered irrelevant. If no optimal solution was found within 900 seconds we report the duality gap, i.e., the gap between the incumbent solution and the best lower bound obtained so far. A dash is written if no solution was found.

In the table we see that the time-indexed formulation overall performs well. The solution times and gaps returned are satisfactory despite the large number of constraints on large instances. We experienced a very stable performance with this formulation. In the branching phase an incumbent solution was found quickly and the duality gap decreased evenly. For our new permutation formulation, however, it took a long time to find an incumbent solution and it was through ”lucky heuristics” that the gap decreased by large amounts

Model	% Opt	% Inc	% BLB	% Win
Base	98	100	100	100
$rnw(F)pd(C)$	33	66	66	0
$rnw(IF)pd(C)$	8	25	42	0

Table 7.5: Comparing formulations for the resource constraints

at a time. This lead to unstable performance. We are pleasantly surprised by the permutation formulation's good performance in the column "Sol" as it outperformed the time-indexed formulation on some instances. But, we believe the permutation formulation has to be further investigated and tested before being used for non-research purposes. Suitable algorithmic and decomposition aspects should be investigated, and it is noted as further work to study how the performance is affected if the permutation constraints are added in a delayed fashion, as mentioned above. In Section 7.4 we test how the permutation performance changes when the cutting planes found in Chapter 5 are added. This improves the performance, but not enough for us to recommend using the permutation formulation for industrial purposes (yet). In general, as the non-simultaneous operations may appear in sets of cardinality larger than two, we will recommend using the time-indexed resource formulation for these constraints.

Result 3. *Non-simultaneous operations should be represented as operations demanding some dedicated resource.*

Resource constraints

The resource constraints for our RCPSP express that at any point in the schedule the resource demands of the operations in progress cannot exceed the availability. We modeled these constraints in three ways; the time-indexed knapsack constraints $r(TI)$, the interval flow formulation $r(IF)$ and the continuous flow formulation $r(F)$ as explained in Chapter 4. We test how these formulations compare when used on the base model on the instances $S1$–$S4$, $S11r$–$S12r$, $M1$–$M3$ and $L1$–$L3$. By Result 3 we modeled the non-simultaneous operations constraints as resource constraints. In the continuous formulations we did this also for the windows of opportunity constraints, whereas in the time-indexed model we used (4.28) for these constraints. The results are given in Table 7.5. Here "% Opt" ("% Inc") is the number of instances in percent where an optimal (incumbent) solution was found, and "% BLB" and "% Win" are as they were above.

The results show that the base model with $r(TI)$ clearly outperformed both flow models. This model found an optimal solution in all but one instance and was best with respect to stability and quality of lower bounds. The interval flow model was the model which led to poorest performance. With this model the solver used much time in the presolve phase and thereby long time to find a lower bound. Both the flow models had very poor lower bounds; the LP relaxation value was often zero even though the optimal value was much larger. The interval-free flow model led the solver to spend much time on heuristics to improve the poor lower bound, not the incumbent solution. We ran the flow models on the extra large instances $XL1$–$XL3$, but no incumbent solutions were found here. This was unexpected, as we believed the flow models would perform at least somewhat satisfactorily on these instances. We believe one reason for the poor performance of the flow models is the large number of constraints and variables. Recall that the number of constraints is $\mathcal{O}(|\mathcal{I}||\mathcal{R}|)$ in $r(F)$ models and $\mathcal{O}(H|\mathcal{R}|)$ in $r(TI)$ models. In all our test instances, inspired by real data, the number of operations $|\mathcal{I}|$ was larger than the planning horizon H in hours. For L instances the number of constraints in the flow model was up to 25 times the number of constraints in the time-indexed model. This number reduced as the size of the instances reduced. On average the number of variables in the time-indexed model was roughly twice the number of variables in the flow model. Nevertheless, the time-indexed model performed best on instances of all sizes, S–XL. For this reason we cannot recommend using the continuous flow models unless they are improved.

Result 4. *Time-indexed models should be used for solving real-life instances of the RCPSP that compare to the test instances in size.*

7.3 Rescheduling Results

We now report how different models performed when solving the rescheduling problem (4.36) introduced in Section 4.6. Information on the rescheduling test instances is given in Table 7.6. For each scheduling problem instance $S2$, $S3$, $L1$ and $XL3$ we give the schedule start ”Start” and schedule end ”End” with dates, and we make two rescheduling problem instances denoted with an extra A or B. The difference between these instances is the date ”ResStart” which is the date from which we reschedule. Let $t' \in \mathcal{H}$ be the time index of this point in time, referring to 07:00 the given date if no hour is given. The rescheduling problem is solved as follows:

1. Let the current schedule be an optimal solution to the corresponding scheduling problem.
2. Remove operations that are finished before t' from the set $\mathcal{I}$ and reduce the duration of the operations that are in progress at t' to their remaining processing time. Remove constraints for $t < t'$ and for the operations that are removed, and do potential updates on parameter values.
3. Set the MIP Start value for the start time of i to be i's start time in the current schedule. This value will help guide Gurobi's search for a feasible schedule.
4. Set ”Start” to ”ResStart” and update the time indices. Add the set $\mathcal{I}_{NEW}$ to $\mathcal{I}$ and introduce new variables.
5. Solve (4.36).

For each of the instances in Table 7.6 we let $\mathcal{I}_{NEW}$ be a set of five safety critical operations. These operations have resource demands, but do not introduce any other new constraints. The parameter values for the constraints in the original scheduling problem remain unchanged. For the weights (W_i^+, W_i^-)

Instance	Start	ResStart	End
$S2A$	28.04.22	02.05.22	27.05.22
$S2B$		16.05.22	
$S3A$	06.05.22	11.05.22	16.05.22
$L1A$	03.04.22	07.04.22	12.07.22
$L1B$		01.06.22	
$XL3A$	01.04.22	05.05.22	15.08.22
$XL3B$		05.06.22	

Table 7.6: Information about test instances for the rescheduling problem

of moving operation i backward or forward in time we make it three times more expensive to move it forward in time compared to postponing it, so $W_i^- = 3W_i^+$. The costs are set to $(3, 9)$ if the operation was originally scheduled to be processed on the "ResStart" date, $(2, 6)$ if scheduled the next day and $(1, 3)$ if scheduled some day later.

We tested and compared how different time-indexed models performed on the rescheduling instances. We do not report the results from the flow models due to their poor performance here and in the above section. As a measure of performance we only report the time spent to find an optimal solution. The reason for this is that quick rescheduling is the major desire from our industry partners. The solution times are listed in Table 7.7. Here we give the objective formulation in "Obj", and in "Formulation" we only state the formulations that are different from the formulation in the base model $(o(C), Prpnw(TI)d(CTI))$. Overall, the results show that there is little difference in running time between the different models that are not extended with the starting time variables. Even for the instances that include nonops constraints the differences in running times are insignificantly small. The tendencies in the results remained the same also when we changed parameter values like durations and resource demands for the operations in $\mathcal{I}_{NEW}$ in order to make some instances harder to solve. Note especially that weekly rescheduling like in $S3A$ is done incredibly fast. As before, the objective $o()$ gives the best result. We recommend using a non-extended time-indexed model for rescheduling purposes. Until we are given real test data

Obj	Formulation	Instance						
		$S2A$	$S2B$	$S3A$	$L1A$	$L1B$	$XL3A$	$XL3B$
$o(C)$	Base						21s	2.2s
	$n(BMTI)$	0.73s			52s	1.3s	20s	2.3s
	$n(BMP)$		0.75s				19s	2.1s
	$pd(C)$	0.76s		< 50ms	208s	1.7s	67s	3s
	$pd(C)n(BMC)$						64s	2.8s
$o(TI)$	$d(TI)$	0.80s	0.88s		47s	1.6s	21s	2.4s
$o()$	$d()$	0.76s	0.68s		37s	1.2s	16s	1.8s

Table 7.7: Comparing formulations for the rescheduling problem

which gives a clearer picture of how hard the rescheduling problems are to solve we do not wish to point out one formulation as strictly better than the other ones.

Result 5. *Time-indexed models are well suited for rescheduling problems.*

7.4 Effectiveness of Cutting Planes

In this section we report how the solver performance is affected when different cutting planes from Chapter 5 are added to the formulations in different models for the RCPSP. We inspect how the values of the LP relaxations to different instances increase when cuts are added and how the overall solution time of the solver potentially changes. We will also observe and report if the cutting planes are actually generated or if they are usually satisfied by the fractional solutions to the LP relaxations and thereby seem redundant. If a cut performs well on these measures we will say that the cut is *effective*, i.e., adding it to the formulation improves the solver performance. We therefore study the *effectiveness* of cutting planes.

We consider the resource cuts (5.14), resource and nonops cuts (5.16), the Big-M permutation cuts (5.4)-(5.5), continuous delay cuts (5.10), nonops and delay cuts (5.12), binary delay cuts (5.13) and the precedence cuts (5.7). For each of these sets $\mathcal{Z}$ of valid inequalities we ran the $\mathcal{Z}$ cut-and-branch algorithm described in Section 6.1. That is, for each family of cuts $\mathcal{Z}$ we used full enumeration to add all inequalities $(a, \alpha) \in \mathcal{Z}$ violated by the fractional solution y^* to the LP relaxations of RCPSP instances to the initial formulations P, and we did this iteratively until a fractional solution was found which violated no inequalities in $\mathcal{Z}$. We only tested other ways of separation for the precedence cuts and based our methods here on the heuristic in Section 6.3. Unless otherwise specified we used the base model for all test runs and default solver settings for Gurobi. In the tables in this section we will add a star * behind the model to indicate that cuts have been added using the $\mathcal{Z}$ cut-and-branch algorithm. For such models the reported solution time is the time spent to solve the RCPSP when using the final formulation where all the generated cuts are added, i.e., the time spent after the while-loop in Algorithm 1.

Resource cuts

We start by running the $\mathcal{Z}$ cut-and-branch algorithm on the resource cuts (5.14) from Section 5.5. Recall that these cuts are extended cover inequalities when the resource constraints are formulated as time-indexed knapsack constraints. The results comparing the base model with and without these cuts added by the $\mathcal{Z}$ cut-and-branch algorithm are given in Table 7.8. In this table we report the time spent to find the optimal solution in the row ”Time”, or we write the duality gap if no solution was found within the time limit. In the row ”LP/Cutval” we give the value of the LP relaxation to the left and the best lower bound found by Gurobi at the root node before the branching starts to the right. If these values were equal we only write it once. The optimal objective value is given in ”ObjVal”. Finally, we report the number of cuts generated by the $\mathcal{Z}$ cut-and-branch algorithm in each iteration in the row ”Cuts”. Each

Measure	Model	Instance					
		$S2$	$S11r$	$S12r$	$L1$	$L4$	$XL2$
Time	Base	1.5s	0.16s	30ms	141s	99%	117s
	Base*	0.44s	0.12s	37ms	98s	76%	213s
LP/CutVal	Base	100	1113/1800	533/600	0	4104	4091
	Base*	219/700		600			
ObjVal		700	1800	600	0		4091
Cuts	Base*	(20,4,3)	(9)	(15,6)	(131,40,5)	(157,33, 16,11,1)	(151,61, 29,11,5)

Table 7.8: Base model performance with and without the resource cuts (5.14) added by the $\mathscr{Z}$ cut-and-branch algorithm

column corresponds to a test instance. We tested seven instances, but give results only for the six in which cuts were generated.

The table shows that adding the resource cuts increased the LP value in some instances. The solution time was reduced in most cases, but not for the extra large instance, which we believe is due to the large number of cuts added to the formulation in this case. The Gurobi Optimizer does generate cover cuts as a part of the solution process, but we can not say if and how it generates lifted or unlifted extended cover inequalities. We wish to inspect further how our resource cuts can be generalized to other or stronger clique cuts or extended cover inequalities and believe a separation algorithm or heuristic should be made for such cuts. By our measures described above we do consider the resource cuts (5.14) as *effective*.

In the same matter as above we tested the effectiveness of the resource and nonops cuts (5.16) from the same subsection. As mentioned there, these cuts are only generated for specific values of the resource demands and availabilities. We ran the $\mathscr{Z}$ cut-and-branch algorithm for these cuts on several instances, among them the instances $S11r$, $M5rn$ and $L8rn$ which were designed specifically with these cuts in mind. We experienced that the cuts were generated, but that they did not improve the lower bounds or the running time on any instance. This was unexpected, as cuts of this form were often returned by the Xpress solver. The cuts did improve the solution time when added to the base model extended with start time variables, but it could not compare to the base model without cuts added. We do not wish to stamp these cuts as ineffective based on our limited test runs. If it is the case for real-life instances of the RCPSP that nonops often do demand the same resources we believe these cuts should be further investigated and tested.

Permutation formulation cuts

We now consider our permutation formulation (5.2)-(5.3) denoted $n(BMP)$ of the non-simultaneous operations constraints. We consider this with and without the permutation cuts (5.4)-(5.5) added by the $\mathscr{Z}$ cut-and-branch algorithm, and compare these models to the base model. We only searched for sets overlapping non-simultaneous operations of size two, i.e., $\delta = \{i, j\}$ for some pair of operations $i,\ j$ included in at least two sets $\Delta_1, \dots, \Delta_p$ of nonops. We compared the models $n(BMP),\ n(BMP)^*$ and Base on a total of four instances

Measure	Model	Instance	
		$S3$	$S8n$
Time	$n(BMP)$	192ms	157s
	$n(BMP)^*$	84ms	73s
	Base	57ms	0.81s
LP/CutVal	$n(BMP)$		0
	$n(BMP)^*$	500	
	Base		600/700
ObjVal		500	700
Cuts	$n(BMP)^*$	(6,2)	(7,1)

Table 7.9: Comparing formulations when the permutation cuts (5.4)-(5.5) are added to the permutation formulation of the nonops constraints

in which we knew sets of overlapping nonops existed, but we only report the two in which cuts were generated. The results are given in Table 7.9.

As only four instances included overlapping sets of nonops it is hard to conclude on the effectiveness of these cuts at this stage. However, we see that the solution time needed by the permutation model decreased considerably when the cuts were added, which is pleasantly surprising for us. The cuts did not affect the LP value. We have already established that the permutation model does not outperform the base model on small instances, and we see that this remains the case also after the cuts are added. We believe a separation algorithm should be made for the permutation cuts or their generalizations (Section 5.2), and that this should be implemented when sets of overlapping nonops are contained in large and extra large instances of the RCPSP.

Result 6. *Further studies of the permutation formulation of the Big-M constraints should include separation with the permutation cuts* (5.4)-(5.5) *or their generalizations.*

Delay cuts

The base model for the RCPSP includes a continuous delay variable $d_i \geq 0$ for each operation $i \in \mathcal{I}$. We compared the performance of this model to the cases where $\mathcal{Z}$ was either the family of continuous delay cuts (5.10) or the nonops delay cuts (5.12), and cuts were added by the $\mathcal{Z}$ cut-and-branch algorithm. We ran the three models on the instances $S2$–$S4$, $L3$, $M3$ and $M4n$ and only comment the results. We experienced that the nonops delay cuts were never generated on any instance. This was disappointing, as we believed they would be generated for at least some instances. However, we suspected these cuts to be weak as we only managed to prove that they were actually P_{nd}/X_{nd} cutting planes for some restricted parameter values. We still believe that the nonops and delay constraints can be combined to find new valid inequalities, but think that the variables then need to be weighted, which makes the inequalities harder to derive by hand. As of now the nonops delay cuts (5.12) are *ineffective.*

The continuous delay cuts (5.10) were often generated. They never increased the LP value and had alternating effect on the running times. We believe the inequalities should be added not as cuts, but as variable lower bounds constraints to the variable d_i. It should be tested whether or not this improves the performance of the base model.

We also tested how the fully time-indexed model with objective $o(TI)$ and binary delay variables w_i^t was affected when the cuts (5.13) were added by the $\mathcal{Z}$ cut-and-branch algorithm. We used the same instances as above. A large number of cuts were generated at each iteration for each instance. This caused the solver time to increase by a large amount, as expected. Surprisingly, the LP value did not increase in any instance after the cuts were added. For this reason we did not investigate these cuts any further, and we believe the base model with either the $o(C)$ or the $o()$ objective remains the best to use for RCPSP instances.

Precedence cuts

We finish off by discussing the effectiveness of the precedence cuts (5.7) when these were added to the time-indexed formulation in the base model. First, we tried *replacing* the precedence constraints (4.19) by all the precedence cuts. It was quickly experienced that the number of constraints then grew very large, and that this led to much of the solver time being spent on presolve and solving the LP relaxation. This was as expected, and confirms that these inequalities should only be added as cuts. Using the $\mathcal{Z}$ cut-and-branch algorithm also turned out to be inefficient with respect to running time as the number of inequalities $(j, i, t) \in \mathcal{Z}$ that were violated usually was very large. However, we experienced that the quality of the lower bound increased considerably, which is what lead us to derive the separation heuristic in Section 6.3.

We implemented the separation heuristic for precedence cuts as explained in Algorithm 2. The set $\mathcal{G} \subset \mathcal{Z}$ of possible violated inequalities was generated when the model was initialized a priori to solving the problem. In order to add inequalities throughout the branching tree the heuristic was implemented in a Gurobi callback function, see Section 6.1. The separation problem solved at different nodes in the branching tree was to calculate v^* by full enumeration of the set $\mathcal{G}$, and if $v^* > 0$ to add all elements (j, i, t) with this violation to $\mathcal{F}$. In order to use callbacks in this way we had to use an academic license for the Gurobi Optimizer. The results reported in this subsection were made using Gurobi Version 10 on an Intel Core i7-8650U CPU 1.90GHz with 4 cores running on a Windows 10 Enterprise with 64GB RAM. All settings remained the same as above, but the Gurobi PreCrush parameter was set to one when the separation heuristic was used in order to not ignore our user-added cuts.

We compared the base model performance with and without using the separation heuristic on the instances $S1$, $S5p$, $S7p$, $L1$, $L4$ and $XL1$. The results are given in Table 7.10. Here, two stars ** are added behind the model name to indicate that cuts have been added throughout the branching tree. The row ”Time” shows total solver time or the current gap if no optimal solution was found. The row ”SepTime” shows the amount of time devoted to the separation heuristic in percentage of the total solver time. In the table we see that the separation heuristic improved performance on some instances. In these instances there were more chained precedence relations within work orders than in the instances where the solver did not improve the performance, which is as expected by the design of the heuristic. Unfortunately, our heuristic did

Measure	Model	Instance					
		$S1$	$S5p$	$S7p$	$L1$	$L4$	$XL1$
Time	Base	22%	37s	0.69s	141s	99%	-
	Base**	28%	18s	0.85s	-		817s
SepTime	Base**	6%	16%	7%	59%	0.1%	0.6%
LP/CutVal	Base	900/24331	2758/6032	20786/23000	0	4104/4105	4143
	Base**	900/24047	2758/5859				
ObjVal		43600	10600	23000	0		4143

Table 7.10: Base model performance with and without the precedence cuts (5.7) added by the separation heuristic

not improve the lower bounds on any instance, which was unexpected as the bounds increased a lot when the $\mathcal{Z}$ cut-and-branch algorithm was used. Further work should therefore inspect the characteristics of the cuts $(j, i, t) \in \mathcal{Z}$ which do improve the LP value. The number of nodes visited in the branching tree was normally reduced when the heuristic was used. The instance $L1$ was solved at the root node by the base model, but an incumbent solution was not found when using the heuristic, which used over half of the available solver time in this case. Further study of this heuristic should limit the maximum time the solver can devote to it in a single node to avoid such problems. It should also be inspected how the solver can use the precedence cuts beneficially without affecting the generation of other solver cuts. The results from $S1$ can imply that the addition of our precedence cuts was done at the cost of not including other Gurobi cuts or using heuristics that could improve the lower bounds and the duality gap.

We find it necessary to test this separation heuristic on real test data before concluding whether or not it should be a part of the model that is to be used for industrial purposes. We know that chained precedence relations are common in such scheduling problems, which is why we remain hopeful that a suitable separation algorithm for the precedence cuts will be beneficial to use and that the cuts will be *effective*.

Result 7. *Suitable separation algorithms and heuristics for the precedence cuts* (5.7) *should be prioritized to be tested on time-indexed models for real-life instances of the RCPSP.*

CHAPTER 8

Concluding Remarks

Inspired by a scheduling problem in the industry this thesis has studied a Resource Constrained Project Scheduling Problem (RCPSP). A solution to this $\mathcal{NP}$-hard problem schedules all operations within the planning horizon in a way that satisfies resource, precedence, non-simultaneous operations and delay constraints and minimizes the weighted delay objective. We have found different time-indexed and continuous formulations for the set of feasible schedules and inspected how these can be strengthened by adding additional valid inequalities. Separation algorithms have been developed and used to test how these inequalities affect the solver performance. Through experimental testing on instances inspired by real data we have tried to get a clearer picture of which model that ensures a stable high solver performance.

$$\text{Winning model: } (o(), Prpnw(TI)d())$$

This model for the RCPSP has a binary, time-indexed formulation with no continuous variable for the delay of each operation. Its simplicity and generality caused it to perform stably well on a variety of test instances, also for rescheduling. For this reason we find it well suited for real applications. Our result is expected and aligns with the literature, but we were surprised that the model outperformed continuous models also on extra large instances.

There are weaknesses and limitations to the winning model result and the results mentioned in Chapter 7, and some of these are discussed in that chapter. The amount and quality of test instances are limited, especially on the very large and hard end. Our result's actual transferability to real scenarios is still unknown. Several valid inequalities from Chapter 5 had limited effect on solver performance, and some were proven to be cutting planes only in specialized cases believed to be realistic. Our separation heuristic for the precedence cuts was designed for data on a particular form, and it did not perform to the anticipated and desired extent. The permutation formulation seems only suitable for restricted parameter values, and its performance was unstable. Nevertheless, we claim that we have reached our goal for this thesis: A winning model has been found, and we believe our results are of applicable interest to the OptiPlan project. Some results are of theoretical interest for the field of maintenance scheduling, but also for resource constrained scheduling and scheduling in general.

All work in this thesis is intended and believed to have been done in an ethically justifiable way, using working procedures common for MILP and RCPSP research. Given all that is described and commented we see no further aspects or approaches potentially able to raise ethical discussions, nor have we any personal interest in being untruthful. The known applications of our results raise no immediate concerns, on the contrary; exciting possibilities.

Further Work

We list some relevant future work and research possibilities:

- It should be investigated if Theorem 5.3.3 also holds for $b < 0$ and for a general $b \in \mathbb{Z}^k$.
- Our permutation formulation needs further study: Why did it perform well on some instances? Can delayed row or column generation be beneficial? A suitable separation algorithm for the permutation cuts and their generalizations should be developed.
- The separation algorithm in Section 6.2 should be expanded to include the optimal choice of operations $(j, i) \in \mathcal{P}$. More tests should be run to inspect how the separation heuristic in Section 6.3 can increase the value of lower bounds and possibly avoid enumeration.
- Several cutting planes from Chapter 5 need to be generalized and strengthened. Our test runs showed that cutting planes arising from sub-problems with several constraint types were strong. Small instances could be made to inspect how such inequalities can be derived by hand.
- The models for our RCPSP should be tested on benchmark instances from PSBLIB or MIPLIB. The continuous flow models should be tested on instances where the planning horizon is larger than the number of operations. Do their performance increase in these cases?
- Our RCPSP can be extended to better fit reality, see Section 4.7. This will lead to new extensions to the standard RCPSP.

Appendices

APPENDIX A

Information On Models And Test Data

This thesis finds and compares different formulations for the set X of feasible schedules to a Resource Constrained Project Scheduling Problem (RCPSP). In Appendix A.1 we list all the symbols used for modelling the RCPSP. Additional symbols used in the rescheduling problem are given in Table 4.2. Appendix A.2 gives information on how the different models for RCPSP are denoted, and in Appendix A.3 we give information on the test instances used for comparing these models.

A.1 List of Symbols

Sets		
$\mathcal{H}$	set of time points	$\{0, 1, \ldots, H\}$
$\mathcal{I}$	set of operations	
$\mathcal{K}_r$	set of interval indices for resource r	
$\mathcal{P}$	set of precedence related pairs of operations	
$\mathcal{P}_i$	set of predecessors to operation i	$\subset \mathcal{I}$
$\mathcal{R}$	set of resource types	
$\mathcal{S}_i$	set of successors to operation i	$\subset \mathcal{I}$
$\mathcal{T}_r$	set of availability changing times for resource r	$\subseteq \mathcal{H}$
$\mathcal{W}_i$	set of windows of opportunity for operation i	$\subseteq \mathcal{H}$
$\mathcal{WO}$	set of work orders	
$\mathcal{X}$	set of pairs of nonops	
$\mathcal{X}^*$	set of maximal sets of nonops	
Δ	maximal set of nonops	$\in \mathcal{X}^*$

Problem parameters

B_i	cost per unit time delay of operation i	$\in \mathbb{R}_+$
C_r^t	amount of resource type r available at time t	$\in \mathbb{N}$
$C_r^{[k]}$	amount of resource type r available in time interval k	$\in \mathbb{N}$
C_r	maximum amount available of resource type r	$\in \mathbb{N}$
H	planning horizon	$\in \mathbb{N}$
L_i	duration of operation i	$\in \mathbb{N}$
R_{ir}	amount of resource type r required by operation i	$\in \mathbb{N}$
T_i	deadline for operation i	$\in \mathbb{N}$

Decision variables

d_i	delay of operation i	$\in \mathbb{R}_+$
f_{ij}^r	flow of units of resource r from operation i to j	$\in \mathbb{R}_+$
t_i	start time of operation i	$\in \mathbb{R}_+$
y_i^t	operation i starts at time t	$\in \{0,1\}$
w_i^t	operation i is delayed t time points	$\in \{0,1\}$
z_{ij}	operation i is scheduled before operation j	$\in \{0,1\}$
z_π	operations in π are permuted like π	$\in \{0,1\}$

Help variables

s_r^k, e_r^k	source and sink nodes for resource r in time interval k
s_r, e_r	source and sink nodes for resource r

A.2 Abbreviations for Model Names

By a *model* for the RCPSP we mean the choice of objective and formulation of X. In Section 4.5 we explain that such a model is denoted $(o(), Pr()p()n()d()w())$ where o stands for objective, r for resource, p for precedence, n for non-simultaneous operations, d for delay and w for windows off opportunity. A reference to the formulation used for these constraints is given inside the parenthesis. In Table A.1 we give abbreviations that are used when comparing the formulations in Chapter 7. Here, TI stands for time-indexed, IF for interval flow, F for flow, BM for Big-M, C for continuous and P for permutation. These letters can be combined. As an example, by $n(BMC)$ we mean using a Big-M formulation of the nonops constraints with the extended continuous start time variable. The motivation for introducing the abbreviations is to make it easier to understand which formulations we are comparing when running tests in Chapter 7.

Abbreviation	Formulation
$r(IF)$	r(4.4)-(4.12)
$r(F)$	r(4.4), (4.6), (4.13)-(4.17)
$r(TI)$	r(4.3)
$p(C)$	p(4.18)
$p(TI)$	p(4.19)
$n(forward)$	n(4.25)
$n(middle)$	n(4.26)
$n(backward)$	n(4.27)
$n(BMC)$	n(4.20)-(4.22)
$n(BMTI)$	n(4.20), (4.23)-(4.24)
$n(BMP)$	n(5.2)-(5.3)
$n(IF)$	n(4.4)-(4.12)
$n(F)$	n(4.4), (4.6), (4.13)-(4.17)
$n(TI)$	n(5.1)
$d()$	none
$d(C)$	d(4.31)
$d(CTI)$	d(4.32)
$d(TI)$	d(4.33)-(4.34)
$w(IF)$	w(4.4)-(4.12)
$w(F)$	w(4.4), (4.6), (4.13)-(4.17)
$w(TI)$	w(4.28)
$o()$	o(4.29)
$o(C)$	o(4.30)
$o(TI)$	o(4.35)

Table A.1: Abbreviations for constraint and objective formulations

A.3 Test Instance Data

As explained in Section 7.1 we made 28 feasible instances to the RCPSP inspired by an instance generated by our industry partners to simulate real-life data. Information on these tests is given in Table A.2. We have categorized the instances into small (S), medium (M), large (L) and extra large (XL) instances depending on the number of operations $|\mathcal{I}|$, number of resources $|\mathcal{R}|$ and the planning horizon H. A problem "Prob" is named as ANc, where $A \in \{S, M, L, XL\}$, $B \in \mathbb{N}$ and $c \in \{\ , p, r, n\}$. Including some character for c indicates that the problem instance was made with a specified intention of testing that type of constraint. As an example, the problem instance $L6n$ is the sixth large instance, and it was made to especially test formulations for non-simultaneous operations. The three last columns give the number of precedence constraints in the problem instance ($\#P$), nonops constraints ($\#N$) and number of operations which have specified windows of opportunity ($\#W$). When two instances of the same size have an equal or almost equal number of some constraint type then the variables in the constraints are different, e.g., the precedence relations are chained in one instance and more random in the other. In the $\#N$ column we write 0 if there were no nonops. If there were, we write a/b where a is the number of pairs of nonops and b is the number of maximal sets of nonops. Using the symbols in Appendix A.1 this gives $a/b = |\mathcal{X}|/|\mathcal{X}^*|$. The resource capacities and demands are given for each time index, giving a total number of $(H + 1)|\mathcal{R}|$ constraints. We also made seven instances for the rescheduling problem (4.36). Information on these instances is given in Section 7.3.

In Chapter 7 it is seen that extra large instances may be easier to solve than some small instances with respect to running time. We list some of the characteristics that in our experience complicated the instances:

- Low resource capacities
- Long chains of precedence relations
- Narrow windows of opportunity

Prob	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{WO}\rvert$	$\lvert\mathcal{R}\rvert$	H	$\#P$	$\#N$	$\#W$
$S1$	100	10	6	176	90	0	0
$S2$	100	10	6	176	20	0	20
$S3$	19	2	6	48	0	84/19	0
$S4$	19	2	6	48	19	9/2	5
$S5p$	44	4	6	88	40	0	0
$S6p$	46	8	6	88	38	0	0
$S7p$	34	2	6	88	32	4/4	0
$S8n$	19	2	6	48	0	84/19	0
$S9n$	19	2	6	48	0	21/5	0
$S10n$	19	2	6	48	0	15/5	0
$S11r$	19	2	6	48	0	6/2	0
$S12r$	8	8	1	64	0	0	0
$M1$	500	322	23	360	30	21/5	30
$M2$	500	322	23	360	50	52/11	0
$M3$	500	322	23	360	0	100/19	0
$M4n$	500	322	23	360	0	165/23	0
$M5rn$	500	322	23	360	0	115/23	0
$L1$	1246	1068	23	576	178	0	0
$L2$	1246	1068	23	576	0	0	0
$L3$	1246	1068	23	576	0	100/19	100
$L4$	1000	822	23	432	50	50/13	50
$L5$	1000	822	23	432	100	100/19	0
$L6n$	1000	822	23	432	0	138/33	0
$L7n$	1000	822	23	432	0	138/32	0
$L8rn$	1246	1068	23	576	0	130/25	100
$XL1$	1500	1090	23	776	302	49/11	52
$XL2$	1500	1090	23	776	52	98/22	52
$XL3$	1500	1090	23	776	52	98/22	52

Table A.2: Full information on the test instances for the RCPSP

APPENDIX B

Block Decomposition Heuristic

The literature on algorithms for solving the standard Resource Constrained Project Scheduling Problem (RCPSP) and its extensions involves both exact approaches and heuristics. In this thesis our emphasis has been on the modelling side; we wanted to find formulations for the set of feasible schedules which can be used in exact IP based algorithms. This was done as we found it of most applicable value for the OptiPlan project, we believed exact algorithms would perform well on real-life instances and we found the search for and comparison of formulations of most personal and mathematical interest.

As a part of the early work with this thesis we designed a simple heuristic for constructing a feasible solution to the RCPSP. We call it the Block Decomposition Heuristic (BDH). This was made before we knew that our problem is actually an extension to the standard RCPSP, and is for that reason uninspired by classic construction heuristics for this problem, see [ADN08]. Due to the heuristic's various performance and limited fit to test data instances we left the heuristic and prioritized to examine and improve the exact formulations. For the simple sake of showing what we have done we include the description of the BDH in this appendix. We hope to improve it further at a later stage in the OptiPlan project when real data is received.

Motivation and assumptions

The idea for the BDH came after learning how scheduling is typically done by our industry partners [Ind22]. The key lessons were:

- The schedules normally consist of work orders $w_1, \ldots, w_W \in \mathcal{WO}$ in which there often are chained precedence relations between the operations in each work order.
- Scheduling is done weekly, and the schedule for weeks close in time is denser than for weeks far away as new work orders arrive regularly.
- The long horizon schedule is tentative and rescheduling is done frequently.

We used these experiences to design the Block Decomposition Heuristic; a heuristic which divides the long planning horizon $\mathcal{H} = \{0, 1, \ldots, H\}$ into disjoint blocks $B_1, \ldots, B_{NB}$ such that $\bigcup_{j=1,\ldots,NB} B_j = \mathcal{H}$ and solves a restricted RCPSP for each block. Let us take a closer look at how this is done.

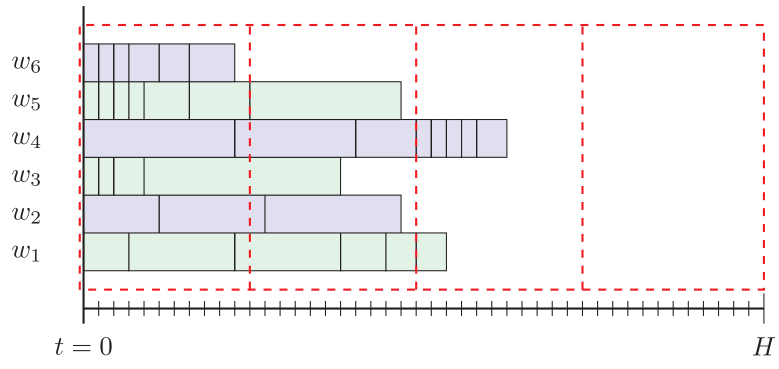


Figure B.1: Initial Block Decomposition schedule with uniform blocks

Dividing the horizon into blocks

Let $NB \in \mathbb{N}$ be the number of blocks which we wish to divide the horizon $\mathcal{H}$ into. We considered two ways of making the blocks:

1. *Uniform blocks.* Let $\alpha = \lfloor \frac{H}{NB} \rfloor$. We set $B_j = \{(j-1)\alpha, \ldots, j\alpha - 1\}$ for $j = 1, \ldots, NB - 1$ and $B_{NB} = \{(NB-1)\alpha, \ldots, H\}$.
2. *Exponential blocks.* Let $\beta_j = \lfloor \frac{H}{2^{NB-j}} \rfloor$. We set $B_1 = \{0, \ldots, \beta_1 - 1\}$, $B_j = \{\beta_{j-1}, \ldots, \beta_j - 1\}$ for $j = 2, \ldots, NB-1$ and $B_{NB} = \{\lfloor \frac{H}{2} \rfloor, \ldots, H\}$.

Both of these block decomposition rules have their weaknesses which will be mentioned shortly. A healthy block decomposition rule should combine the two above rules and also take into account the number of operations in each block. This number is determined by the initial schedule.

Creating an initial schedule

The problem (RCPSP) consists of different constraint types. As we have seen in this thesis that windows of opportunity and non-simultaneous operations may be represented as resource constraints, and that the delay variable constraints may be dropped, we only consider the precedence constraints p and resource constraints r in this appendix. We assume that the precedence relations are chained within each work order.

A schedule is called c *feasible* if it satisfies all constraints of type $c \in \{p, r\}$. The BDH starts by creating a precedence feasible schedule in a way that minimizes the completion time of each work order. See Figure B.1 for a Gantt chart illustration. Here, $W = 6$ and each row corresponds to a work order. The operations in each work order are pushed to the left as much as possible in a way that satisfies the chained precedence relations. We also illustrate the uniform blocks for $NB = 4$ in the figure.

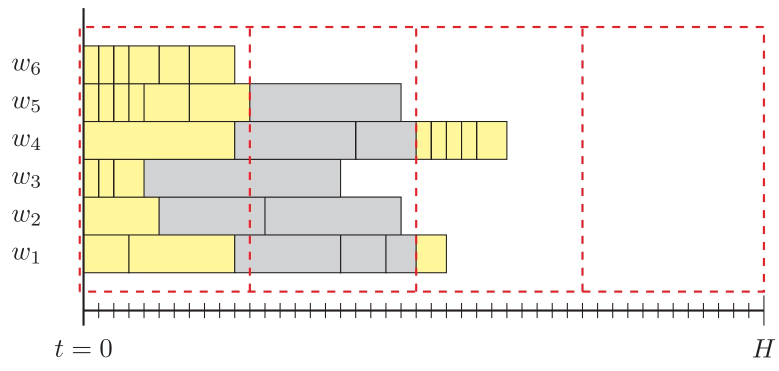


Figure B.2: Operations divided by blocks

Solving the restricted RCPSP for each block

We let $\mathcal{I}_j$ be the set of all operations which are completed in block j in the initial schedule, i.e., we let $\mathcal{I}_j = \{i \in \mathcal{I} : Q_i + L_i - 1 \in B_j\}$ where Q_i is the start time of operation i in the initial schedule and L_i is the duration. Define $\mathcal{I}_{1:j} = \bigcup_{k=1}^{j} \mathcal{I}_k$ and let $\mathcal{C}_{1:j}$ to be a set of all constraints which include the operations in $\mathcal{I}_{1:j}$. Those are the precedence constraints $(j, i) \in \mathcal{P}$ where both $j, i \in \mathcal{I}_{1:j}$ and all resource constraints, but we ignore the demands of each operation $i \notin \mathcal{I}_{1:j}$. Finally, let $NB^* = |\{j : |\mathcal{I}_j| > 0\}|$ be the number of blocks in which there are operations ending in the original schedule. Should it happen that $|\mathcal{I}_j| = 0$ for some $j \leq NB^*$ then this block is collapsed with the following block. In Figure B.2 we see how the operations from Figure B.1 are divided into $NB^* = 3$ sets of operations.

The BDH attempts to find a good feasible solution to the RCPSP by solving NB^* restricted IPs where the values in the previous solutions are fixed when moving on to the next IP. This is done as follows: After creating the initial schedule and finding the above-mentioned sets the BDH solves the problem IP$_1$. This problem contains the operations in $\mathcal{I}_1$, constraints in $\mathcal{C}_1$ and uses the full horizon $\mathcal{H}$. The objective is to minimize the unweighted sum of start times $\sum_{i \in \mathcal{I}_1} t_i$. After solving IP$_1$ the BDH continues to solve IP$_2$; a problem which contains the operations $\mathcal{I}_{1:2}$ and constraints $\mathcal{C}_{1:2}$, but where the start times of the previously scheduled operations $i \in \mathcal{I}_1$ are fixed to their optimal start time from IP$_1$. This procedure continues iteratively until either IP$_{NB^*}$ is solved to optimality or some IP$_j$ is infeasible. If the latter is the case the start times of the operations in $\mathcal{I}_{1:j-1}$ are returned and the start times for $\mathcal{I}_{j:NB^*}$ are ignored. This solution is *not* valid for the RCPSP. If IP$_{NB^*}$ is solved then the returned solution will be feasible for the RCPSP. The full BDH is summarized in Algorithm 3 on page 91. There, $pred(i)$ is the predecessor to operation i and t_i is the start time decision variable.

Results, challenges and improvements

The idea for the BDH seems simple and intuitive: By dividing the full RCPSP into smaller problems the heuristic iteratively solves IPs that are small in number of operations and fixes some solution values to the next iteration. However, there are several challenges with the heuristic design. We mention some of them and point to ideas for how the BDH can be improved.

- *Assumptions.* The BDH assumes chained precedence relations in each work order in order to create the initial schedule. In general, we cannot assume that this is always the case, as there may exist work orders with no precedence relations between any of the operations.
- *Objective.* In order to push operations "to the left" so that the later operations can fit in the schedule the objective must in some way attempt to minimize start times or work order completion times. However, the objective in our RCPSP is to minimize the weighted delay costs; a goal which does not necessarily need the operations to be left-pushed.
- *Operations in blocks.* If H is small or NB is large it may happen for both the block decomposition rules that there are no or few operations finishing in the first block. This was experienced in test runs. We believe a beneficial BD rule should ensure a suitable number of operations in each block, as well as choosing NB appropriate to H and $|\mathcal{I}|$.
- *Solving many IPs.* The heuristic involves solving NB^* IPs which are as hard as the original RCPSP in complexity, though smaller in number of operations. Solving each to optimality can be time-consuming, so one could consider either setting an optimality gap for the restricted IPs or solving them heuristically. However, this will have a domino effect which can cause the final objective value to be poor.
- *Infeasible* IP$_j$. A restricted IP may be infeasible, making the heuristic unable to return a feasible solution. An improved BDH should have a low risk of not returning a feasible solution. However, for a large H a partial solution for the first blocks may be enough for the operation planners in practice. The rest of the schedule may be created by rescheduling or solving a new scheduling problem by updating the parameter values and increasing the horizon H.

These experiences were some of the reasons why we prioritized and still prioritize to use exact algorithms when solving the RCPSP. To the test instances which do include chained precedence relations the heuristic found satisfyingly good solutions in some cases, despite an unstable behavior. These are given in Table B.1. We compare the BDH performance with the exact base model performance (see Chapter 7) with a time limit of 900 seconds. The "Time" reported for the BDH includes the full time spent running Algorithm 3. The "BestVal" row shows the optimal value or the value of the incumbent solution at the time limit. The results indicate that for instances with chained precedence relations the BDH may be suitable to use for finding a MIP start value for the exact solver. We believe the BDH should be further developed and improved on the above-mentioned challenges before using this approach.

Measure	Solver	Instance	
		$S1$	$S5p$
Time	Base	22%	37s
	BDH	123s	2.9s
BestVal	Base	43600	10600
	BDH	51600	11600
ObjVal		43600	10600
Rule/NB	BDH	Exp/4	Uniform/4

Table B.1: Best results of the Block Decomposition Heuristic

Algorithm 3 Block Decomposition Heuristic

```
Require: $NB$ = Number of blocks
Ensure: Solution to IP$_{NB^*}$ or partial solution to some IP$_j$

  for $w \in \mathcal{WO}$ do                                    ▷ Make initial schedule
      $C_i \leftarrow 0$ for first operation $i$ in $w$
      for $i \in w$ do
          $C_i \leftarrow C_{pred(i)} + L_{pred(i)}$
      end for
  end for

  Partition $\mathcal{H} = \{0, \dots, H\}$ into $NB$ blocks $B_1, \dots, B_{NB}$
  for $j \leftarrow 1, NB$ do
      $\mathcal{I}_j \leftarrow \{i \in \mathcal{I} : C_i + L_i - 1 \in B_j\}$
  end for

  $NB^* = |\{j : |\mathcal{I}_j| > 0\}|$
  for $j \leftarrow 1, NB^*$ do                                   ▷ Initialize IP$_j$
      Operations $\leftarrow \mathcal{I}_{1:j}$
      Constraints $\leftarrow \mathcal{C}_{1:j}$
      Objective $\leftarrow$ min $\sum_{i \in \mathcal{I}_{1:j}} t_i$
      for $i \in \mathcal{I}_{1:j-1}$ do fix $t_i$ to optimal value from IP$_{j-1}$
      end for
      Solve IP$_j$
      if IP$_j$ infeasible then return solution from IP$_{j-1}$
      end if
  end for
  return solution from IP$_{NB^*}$
```

Bibliography

[ADN08] Artigues, C., Demassey, S. and Néron, E., eds. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications.* Control Systems, Robotics and Manufacturing Series. John Wiley & Sons, 2008.

[BMR88] Bartusch, M., Möhring, R. H. and Radermacher, F. J. 'Scheduling project networks with resource constraints and time windows'. In: *Annals of operations research* vol. 16, no. 1 (1988), pp. 199–240.

[CCZ14] Conforti, M., Cornuéjols, G. and Zambelli, G. *Integer Programming.* Cham, 2014.

[Chv73] Chvátal, V. 'Edmonds polytopes and a hierarchy of combinatorial problems'. In: *Discrete Mathematics* vol. 4, no. 4 (1973), pp. 305–337.

[CL22] Christof, T. and Löebel, A. *PORTA.* https://porta.zib.de. Version 1.4.1. 7th Oct. 2022.

[Dah10] Dahl, G. *An introduction to convexity.* University of Oslo, 2010.

[DF03] Dahl, G. and Foldnes, N. 'Complete description of a class of knapsack polytopes'. In: *Operations research letters* vol. 31, no. 5 (2003), pp. 335–340.

[DM12] Dahl, G. and Mannino, C. *Notes on combinatorial optimization.* 2012.

[Dre+00] Drexl, A. et al. 'ProGen/ πx – An instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions'. In: vol. 125, no. 1 (2000), pp. 59–72.

[Dup22] Dupin, N. 'Integer Linear Programming Reformulations for the Linear Ordering Problem'. In: *Optimization and Learning.* Ed. by Dorronsoro, B. et al. Vol. 1684. Communications in Computer and Information Science. Switzerland: Springer International Publishing AG, 2022, pp. 74–86.

[Fai22] Fair Isaac Corporation. *FICO Xpress Solver.* https://www.fico.com/fico-xpress-optimization/docs/latest/overview.html. Version 9.0. 14th Oct. 2022.

[GM02] Gabrel, V. and Minoux, M. ‘A scheme for exact separation of extended cover inequalities and application to multidimensional knapsack problems’. In: *Operations Research Letters* vol. 30, no. 4 (2002), pp. 252–264.

[Gur22] Gurobi Optimization, LLC. *Gurobi Optimizer*. https://www.gurobi.com/. Version 9.5.2. 7th Aug. 2022.

[HB10] Hartmann, S. and Briskorn, D. ‘A survey of variants and extensions of the resource-constrained project scheduling problem’. In: *European Journal of Operational Research* vol. 207, no. 1 (2010), pp. 1–14.

[Hoj+20] Hojny, C. et al. ‘Knapsack polytopes: a survey’. In: *Annals of operations research* vol. 292, no. 1 (2020), pp. 469–517.

[Ind22] Industry partners. Private communication. Oct. 2022.

[Kar+21] Kardos, D. et al. ‘Numerical experiments with LP formulations of the maximum clique problem’. In: *Central European Journal of Operations Research* vol. 30 (2021).

[Kha79] Khachiyan, L. G. ‘A polynomial algorithm in linear programming’. In: *Soviet Mathematics Doklady*. Vol. 20. 1979, pp. 191–194.

[KKF09] Keha, A., Khowala, K. and Fowler, J. ‘Mixed integer programming formulations for single machine scheduling problems’. In: *Computers & Industrial Engineering* vol. 56 (2009), pp. 357–367.

[KL08] Kaparis, K. and Letchford, A. N. ‘Local and global lifted cover inequalities for the 0–1 multidimensional knapsack problem’. In: *European Journal of Operational Research* vol. 186, no. 1 (2008), pp. 91–103.

[Kon+11] Koné, O. et al. ‘Event-based MILP models for resource-constrained project scheduling problems’. In: *Computers & operations research* vol. 38, no. 1 (2011), pp. 3–13.

[Law77] Lawler, E. L. ‘A “Pseudopolynomial” Algorithm for Sequencing Jobs to Minimize Total Tardiness’. In: *Studies in Integer Programming*. Ed. by Hammer, P. et al. Vol. 1. Annals of Discrete Mathematics. Elsevier, 1977, pp. 331–342.

[Law78] Lawler, E. ‘Sequencing Jobs to Minimize Total Weighted Completion Time Subject to Precedence Constraints’. In: *Algorithmic Aspects of Combinatorics*. Ed. by Alspach, B., Hell, P. and Miller, D. Vol. 2. Annals of Discrete Mathematics. Elsevier, 1978, pp. 75–90.

[Mar11] Maruoka, A. *Concise Guide to Computation Theory*. 1st ed. London: Springer, 2011.

[Ner19] Neri, F. ‘An Introduction to Computational Complexity’. In: *Linear Algebra for Computational Sciences and Engineering*. Cham: Springer, 2019, pp. 419–432.

[NR10] Naderi, B. and Ruiz, R. ‘The distributed permutation flowshop scheduling problem’. In: *Computers & Operations Research* vol. 37, no. 4 (2010), pp. 754–768.

[PB18] Prot, D. and Bellenguez-Morineau, O. ‘A survey on how the structure of precedence constraints may change the complexity class of scheduling problems’. In: *Journal of scheduling* vol. 21, no. 1 (2018), pp. 3–16.

[Pin22] Pinedo, M. L. *Scheduling : Theory, Algorithms, and Systems.* 6th ed. Cham: Springer, 2022.

[Pul89] Pulleyblank, W. R. ‘Chapter V polyhedral combinatorics’. In: *Handbooks in operations research and management science* vol. 1 (1989), pp. 371–446.

[PW06] Pochet, Y. and Wolsey, L. A. ‘Production Planning by Mixed Integer Programming’. In: *Springer Series in Operations Research and Financial Engineering* (2006).

[PWW69] Pritsker, A. A. B., Watters, L. J. and Wolfe, P. M. ‘Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach’. In: *Management Science* vol. 16, no. 1 (1969), pp. 93–108.

[RM12] Riise, A. and Mannino, C. *The surgery scheduling problem-a general model.* 2012.

[RT12] Roberti, R. and Toth, P. ‘Models and algorithms for the Asymmetric Traveling Salesman Problem: an experimental comparison’. In: *EURO Journal on Transportation and Logistics* vol. 1, no. 1 (2012), pp. 113–133.

[Sch17] Schrijver, A. *A Course in Combinatorial Optimization.* 2017.

[Sch86] Schrijver, A. *Theory of linear and integer programming.* Wiley-Interscience series in discrete mathematics. Chichester: John Wiley & Sons, 1986.

[Smi56] Smith, W. E. ‘Various optimizers for single-stage production’. In: *Naval Research Logistics Quarterly* vol. 3 (1956), pp. 59–66.

[Vit12] Vitányi, P. ‘Turing Machines and Understanding Computational Complexity’. In: *CoRR* (2012).

[Wei96] Weismantel, R. ‘Hilbert Bases and the Facets of Special Knapsack Polytopes’. In: *Mathematics of operations research* vol. 21, no. 4 (1996), pp. 886–904.

[WN88] Wolsey, L. A. and Nemhauser, G. L. *Integer and Combinatorial Optimization.* John Wiley & Sons, 1988.

[Wol06] Wolter, K. ‘Implementation of cutting plane separators for mixed integer programs’. PhD thesis. Technische Universität Berlin, 2006.

[Wol90] Wolsey, L. A. ‘Valid inequalities for 0–1 knapsacks and mips with generalised upper bound constraints’. In: *Discrete Applied Mathematics* vol. 29, no. 2-3 (1990), pp. 251–261.

[Węg+11] Węglarz, J. et al. ‘Project scheduling with finite or infinite number of activity processing modes – A survey’. In: *European Journal of Operational Research* vol. 208, no. 3 (2011), pp. 177–205.

[Zem89] Zemel, E. ‘Easily Computable Facets of the Knapsack Polytope’. In: *Mathematics of Operations Research* vol. 14, no. 4 (1989), pp. 760–764.

[ZIM21] Zaied, A. N. H., Ismail, M. M. and Mohamed, S. S. ‘Permutation flow shop scheduling problem with makespan criterion: literature review’. In: *J. Theor. Appl. Inf. Technol* vol. 99, no. 4 (2021), pp. 830–848.