CHALMERS

New valid inequalities for a time-indexed formulation of the flexible job shop scheduling problem

Master's Thesis in Engineering Mathematics and Computational Science

DAVID YONG-MIN LEFFLER

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2015
Master's Thesis 2015:NN

Master's thesis 2015:NN

New valid inequalities for a time-indexed formulation of the flexible job shop scheduling problem

David Yong-min Leffler

CHALMERS

Department of Mathematical Sciences
Division of Mathematics
Chalmers University of Technology
Gothenburg, Sweden 2015

New valid inequalities for a time-indexed formulation of the flexible job shop scheduling problem

David Yong-min Leffler

Supervisor: Karin Thörnblad
Examiner: Michael Patriksson

Master's Thesis 2015:NN
Department of Mathematical Sciences
Chalmers University of Technology
SE–412 96 Gothenburg
Telephone +46 (0)31 772 1000

Printed in Gothenburg, Sweden 2015

New valid inequalities for a time-indexed formulation of the flexible job shop scheduling problem

David Yong-min Leffler

Department of Mathematical Sciences
Chalmers University of Technology

Abstract

In this thesis a family of valid inequalities originally formulated for single- and parallel machine scheduling problems are extended to a time-indexed, mixed integer linear programming formulation of the flexible job shop scheduling problem. The model of the flexible job shop scheduling problem that is used for this purpose was originally formulated in the licentiate thesis [22], and is based on the practical case of scheduling the production of aircraft engine components at a multitask production cell in Trollhättan, Sweden. The strength of the valid inequalities when applied to this model is assessed by means of computational testing. This has been performed using a cutting-plane method on Fattahi test instances, both with the objective of minimizing makespan as well as minimizing tardiness. For both objectives the new valid inequalities have yielded improved lower bounds, however with varying effect. Computational results are reported along with suggestions for implementation of the valid inequalities that are likely to grant the best results.

Keywords: *multipurpose machine, flexible job shop scheduling problem, MILP, mathematical optimization, time-indexed decision variables, valid inequalities, polyhedral methods, cutting-plane methods, makespan-tardiness*

Acknowledgements

This master's thesis marks the end of my studies here at Chalmers which is a milestone of sorts. With this in mind, I feel the urge to express my gratitude not only towards those that have helped me throughout the course of this project, but also to those that have supported and accompanied me in this chapter of my life.

First, I would like to express my deepest thanks to my supervisor Karin Thörnblad for her invaluable mentorship, dedication and encouragement throughout the course of this project. Finishing this thesis would, without a doubt, not have been possible without you. Furthermore, I would like to thank my examiner Michael Patriksson, for his thorough scrutiny of my written work, difficult questions and feedback that helped me learn and improve. Thank you both for the opportunity to work on this project, and for guiding me to its completion.

I would also like to thank my family and friends. I feel both fortunate and grateful to have parents like Un-ae and Hakon Leffler, for their unconditional love, support and patience with me throughout my life and at Chalmers, through thick and thin. I would also like to express my sincere appreciation to my sister Märta. I owe you more than you know and I want to let you know how much I truly appreciate having you in my life without sounding too cheesy or cliché. It's tricky though, and words are certainly not enough, but hopefully you still get what I mean. I would also like to thank my brother Carl. Although we do not meet as often as I would like, you are always present and a constant inspiration to me.

Among my friends I would in particular like to thank Carl, Emil and Erik, who have both shared and enriched my life at Chalmers since the beginning, and who I will always associate with my time here. I would also like to thank my friend Ufuk for the motivating discussions on research in applied mathematics and for proofreading my final report. Last but not least, I would also like to thank my lovely girlfriend Georgia, for being the amazing and caring person that you are, and for being there for me throughout the course of this project.

David Yong-min Leffler
Gothenburg, February 2015

Contents

1

Introduction

Polyhedral methods have yielded substantial results with regards to finding the solution to many important NP-hard optimization problems. The development of polyhedral methods for machine scheduling problems, however, is still in its early stages. The flexible job shop scheduling problem (FJSP), that is studied in this thesis, is a generalization of the so-called job shop scheduling problem (JSP), one of the most thoroughly researched classes of machine scheduling problems. Due to the inherent complexity of the JSP, the majority of existing research has been on the JSP in its simplest form; the single-machine scheduling problem (SMSP). The idea behind this approach is to hopefully extend knowledge of the structural properties for this simpler group of problems to more complex, multi-machine formulations.

In this thesis a family of valid inequalities originally derived for a time-indexed mixed integer linear programming (MILP) model of the SMSP (see [21]) is extended to a time-indexed MILP model of the FJSP. This model of the FJSP has its roots in application, stemming from the problem of finding more efficient schedules at a multitask production cell for aircraft engine components in Trollhättan, Sweden (more on this in the following section). The planning and control of such a multitask cell can be formulated as a FJSP, which was the research topic of a licentiate thesis [22] in which three MILP models for solving the FJSP were implemented and compared. Of the three models tested, a time-indexed formulation performed exceptionally well, showing promise both with regards to the quality of the solution found, as well as solving the FJSP within an amount of time that allowed for practical use. It is this time-indexed formulation that we hope to strengthen with the aforementioned family of valid inequalities.

1.1 Background

GKN Aerospace (previously Volvo Aero and purchased by GKN in 2012) has invested in a multitask production cell consisting of a set of ten production resources, five of which are multipurpose machines. The purpose of this multitask cell is to, relative to an ordinary job shop at the production site, perform a larger variety of jobs with an increased degree of machine utilization. Furthermore, the aim is to shorten lead times, decrease product costs, and increase delivery precision [23].

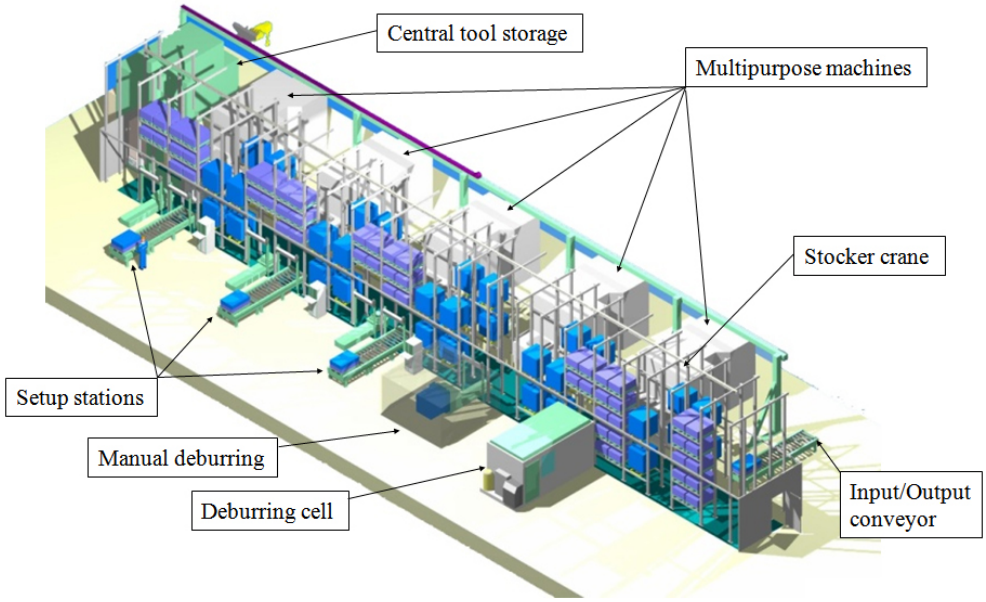


Figure 1.1: Overview of the multitask cell

A built-in scheduling algorithm based on a simple priority function was delivered with the multitask cell, to aid in the planning of the cell's resources when it was first bought. As the topic of a master's thesis in 2006 (see [13]), the schedules produced using this default function were evaluated, revealing that use of the default function may not be well adapted to the production of complex components (e.g., aero engine compressor rear frames in the case of GKN Aerospace), and would often result in a great deal of unused potential. This result is one of the reasons why the scheduling of the multitask cell is currently performed manually (for a more detailed description on the current planning of the multitask cell see [25]). An additional conclusion from the thesis [13] was that implementing a more refined scheduling algorithm had great potential in improving the efficiency of production.

The results of the aforementioned master's thesis inspired the topic of a licentiate thesis (see [22]), where mathematical optimization was proposed as a method to improve the

scheduling of the multitask cell. Optimizing the scheduling of jobs in a set of multi-purpose machines (such as the multitask production cell at GKN Aerospace) is known to be a complex combinatorial optimization problem that is identified within the field of operations research as a *flexible job shop scheduling problem* (FJSP) (see section 2.1 for more on theoretical scheduling problems) and in the licentiate thesis [22], a number of different mathematical formulations of the FJSP were implemented and compared. One formulation of the FJSP in this study, modeled as a time-indexed MILP, performed exceptionally well with regards to both computation time and sizes of instances. To our knowledge there are no other existing implementations of such a model to date.

The main advantage of using a time-indexed formulation is that they often provide very strong lower bounds (via the solution to their LP relaxation) relative to other MILP formulations (see [26]). Sadly, the trade-off to this is that time-indexed formulations also tend to be very large in size, where even relatively small instances can result in an intractable number of constraints and variables (for more on time-indexed formulations see section 2.1.3). The memory required to store an instance, as well as the time required to solve the LP relaxation of even a smaller time-indexed model, can thus become too large for practical purposes. As a result, time-indexed models have historically only been applicable to simpler scheduling problems, and smaller instances. Although prospects for solving larger models are continously improving with advancements in the computational capabilities of computers, it is still important to find ways to reduce memory requirements, as well as the solution time of the LP relaxation, if time-indexed MILP formulations are to have significant impact on the solution of machine scheduling problems.

In polyhedral theory, it is well-known that valid inequalities (in particular so-called facet-inducing valid inequalities) can be used to improve the lower bounds provided by the LP relaxation and in so doing potentially improve the time to solve the unrelaxed problem as well (for more information on polyhedral theory see sections 2.2.2 and 2.3). In a Ph.D. thesis (see [2]), three classes of valid inequalities (VIs) were developed for a time-indexed MILP formulation of yet another category of machine scheduling problems: the *parallel-machine scheduling problem* (PMSP). The strength of the new classes of VIs was tested via a cutting-plane method, resulting in improved lower bounds when applying the VIs both individually and in combination. One of the three classes of VIs tested is based on a family of VIs that were previously formulated for a time-indexed model of the most basic type of machine scheduling problem, the *single-machine scheduling problem* (SMSP) (see [21]). Sousa and Wolsey also established that, if the time horizon is large enough, the VIs are facet-inducing for their time-indexed model of the SMSP [21]. Since the SMSP is a sub-problem of the FJSP (if one reduces the number of machines in the FJSP to one), it is possible that this family of VIs can be extended to the time-indexed FJSP model formulated in [22] as well. With this in mind, as well as the success in strengthening the model of the PMSP in [2] using an extension of these VIs, we are motivated to test the validity and effects of implementing this class of VIs to the time-indexed model of the FJSP developed in [22].

Interface between cutting-plane and an iterative solution procedure

As previously mentioned, the main drawback of using a time-indexed model is that the number of decision variables and constraints grows quickly with problem size. Since the number of variables and constraints is highly dependent on the number of time steps chosen to discretize the total planning period, one approach to circumventing the issue of model size is to try and find a solution using as few time steps as possible. This is the basic idea behind a new solution procedure developed in [23], where the time-indexed model that is used in this thesis is solved for iteratively smaller time steps. Here, the best schedule found in one iteration is transformed into a feasible starting solution for the next iteration. The resulting makespan is also used to choose a suitable size for the planning period in the next iteration. The aim is thus to keep the number of time steps as small as possible while progressively increasing the accuracy of the solution with each iteration (for a more detailed description of this solution procedure see [23] or [24]).

A potential application of the new VIs (and partly what inspired the subject of this thesis) is to run a cutting-plane algorithm parallell to the aforementioned iterative solution method. One of the termination criterion of the iterative solution procedure is that, at the end of each iteration, the *mipgap*[1] is sufficiently small. By 'sufficiently small' we mean that the *mipgap* is smaller than some predefined number and is hence either optimal (if LB $= z$) for the current iteration, or is close enough to optimality for practical purposes. In other words, the *mipgap* is used to measure the quality of the solution found so far and the algorithm terminates if the *mipgap* check at the end of each iteration qualifies the solution as optimal or near optimal within some predefined margin.

[1] In CPLEX version 12, the following definition of the *mipgap* is used: $mipgap = \frac{z-LB}{10^{-10}+z} \cdot 100\%$, where z denotes the objective value of the best solution to the unrelaxed problem found so far, and LB denotes the best lower bound found so far.

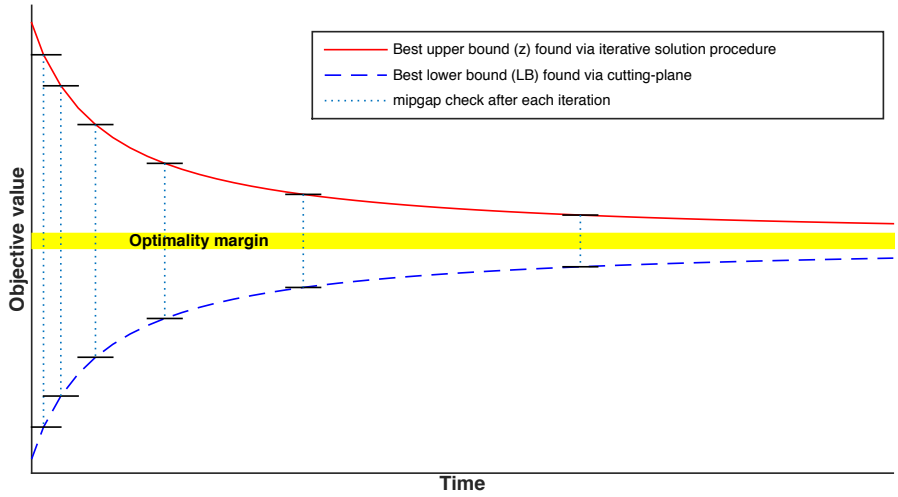


Figure 1.2: Narrowing the *mipgap* by running a cutting-plane algorithm parallel to the iterative solution procedure in [23]. The highlighted yellow region marked "Optimality margin" denotes a predifined margin in which the objective value of a solution is considered optimal or near optimal. Note that this figure is only a visualization of this concept, and is not based on real data.

By raising the LB using a cutting-plane algorithm with the new VIs, for use in the *mipgap* check at the end of each iteration of the iterative solution procedure, one can potentially qualify the optimality (or near optimality) of a solution at an earlier iteration. The potential time saved, however, must of course be weighed against the extra computation time required to calculate the LBs with the new VIs included.

1.2 Purpose

The purpose of this thesis is to further develop the time-indexed MILP model of the FJSP developed in [22], with the goal of producing a stronger formulation with improved computation times, and in so doing further the search for strong MILP descriptions of the FJSP. This will also be done with the potential of applying this model to a real production in mind. To do this a class of valid inequalities originally formulated for the SMSP will be extended to the FJSP. Since this class of valid inequalities is quite large they will be evaluated via a cutting-plane method. Hopefully this will also provide some further insight into extending knowledge from single- to multiple machine scheduling. On a more general level, the aim of this study is also to improve on tools for finding optimal, or near-optimal, schedules for multitask cells similar to that of GKN Aerospace.

1.3 Research questions

In this thesis we aim to answer the following research questions:

- What are the effects of implementing the aforementioned class of valid inqualities

originally formulated for a time-indexed model of the SMSP [21], and previously extended to a time-indexed model of the PMSP [2], to the time-indexed MILP model of the FJSP developed in [22]?

  - What are the effects on the initial optimality gap for different types of problem instances?
  - What are the effects on the time to solve the LP-relaxation?

- Can a cutting-plane algorithm using these valid inequalities improve the performance of the iterative solution procedure developed in [23]?

1.4 Limitations

To evaluate the efficacy of the new VIs, a generic MILP solver (CPLEX) will be used to obtain solutions to the LP relaxation of our model in each iteration of our cutting-plane algorithm. The choices that the software makes when solving the LP relaxation will not be discussed (for details on this we refer to the CPLEX manual [11]).

In most literature the FJSP consists of a sequence of operations to be processed in a given order, just as in the original JSP. For FJSPs appearing in industry, however, it is fairly common that jobs will have operations that have other precedence conditions. For example: 'assembly' sequences where two mutually independent sequences eventually merge into one, 'disassembly' sequences where a single sequence splits into two mutual independent sequences etc.. The extension of the model that we are working with to include these other types of precedence relations will not be included in this thesis (see [3] for an example of a FJSP formulation that includes these conditions). However, these other types of precedence conditions can be handled by formulating them as precedence relations between jobs (see [22]). The test instances that will be used will also be limited to the 10 largest Fattahi instances (for more on the choice of these test instances see section 4.1).

Another potentially interesting topic of study is to determine the dimension of the polyhedra associated with the time-indexed model that we will be using. Doing so can help determine whether or not the VIs tested are so-called facet inducing (for theory on this see section 2.2.2). This will be left as a topic for future research, and we will have to be content with testing the strength of the new VIs empirically.

The majority of research on the FJSP has previously been focused on heuristics, due to the computational complexity of the problem. Many of these heuristics could potentially be used to complement the model that we are using, since by providing a good feasible starting solution one can potentially cut down on the number of nodes in the branch-and-bound tree and thus speed up the solution process. This will also be left as a topic of future research.

1.5 Outline

In chapter 2 we will introduce the subjects and preceding research that make up the context of this study. This will consist of an introduction to theoretical scheduling problems in section 2.1 with a particular focus on the FJSP and the use of mathematical programming to solve this class of problems. Furthermore, we will introduce some basic notation and theory related to mixed integer linear programming in section 2.2, and polyhedral theory in section 2.2.2. The notation and concepts introduced in these sections will then be used to introduce the reader to the theory of valid inequalities in section 2.3 as well as cutting-plane algorithms in section 2.4. In chapter 3 we will introduce the time-indexed MILP model of the FJSP that will be used in this thesis. To simplify implementation some small alterations have been made to the original model that was developed in [22], which will be the topic of section 3.2.3. A brief description of the original models development along with a discussion regarding the choice of objective function will be the topic of section 3.1. In the final section of chapter 3 (section 3.3), we will present the family of VIs in their original form for the SMSP followed by their extension to the FJSP. Chapter 4 will be dedicated to presenting our cutting-plane algorithm implementation along with computational results. The Fattahi test instances that are used as input data are also described in this chapter, under section 4.1. Finally, in chapter 5, we will present our conclusions, a discussion on our study in retrospect as well ideas for future research.

2

Subject orientation

The following chapter consists of a basic review of the concepts required to interpret the results of this thesis, as well as a summary of some of the preceding research within related fields of study. An introduction to theoretical scheduling problems with a particular focus on the job-shop scheduling problem (JSP) and its extension to the FJSP is given in section 2.1. To frame this section to fit within the scope of this thesis, we will be focusing on the solution of scheduling problems by means of mathematical optimization, in particular through use of time-indexed mathematical models such as the one used in this thesis (for an overview of various approaches to solving the JSP or the FJSP see [12] and [7]). Next, relevant theory on mixed integer linear programming is presented in section 2.2 as well as a brief description of the main motivation underlying the application of this methodology to the solution of scheduling problems. The assessment of MILP formulations also requires some basic results and concepts from polyhedral theory which will be presented in section 2.2.2 and 2.3. Finally, an introduction to cutting-plane methods is given in section 2.4.

2.1 Scheduling theory

The flexible job shop problem (FJSP), single-machine scheduling problem (SMSP) and parallell-machine scheduling problem (PMSP) that were mentioned in chapter 1 are just a few classes among a broad range of theoretical scheduling problems that are studied in a field of research known as *scheduling theory*. The first scheduling algorithms were formulated for solving models of industrial production processes in the mid-fifties, and in the seventies computer scientists found these to be a useful tool for improving the performance of computer systems. Over time scheduling theory has grown into a multidisciplinary, cross-disciplinary field (e.g., manufacturing, computer design, logistics, communication etc.) with techniques ranging from simple dispatching rules to highly sophisticated algorithms and heuristics [6, 12].

A general scheduling problem consists of finding optimal processing sequences (with respect to some performance measuring function, commonly referred to as the *objective function* or *cost function*) of a given set of jobs on a designated set of resources (machines in the case of GKN aerospace). In the simplest case, the SMSP, the number of resources is one, and we are simply left with the problem of finding the optimal order in which to process the jobs on a single resource. Due to the inherent complexity of most scheduling problems, research in scheduling theory often starts with the SMSP with the intention of extending results to more complex problems. The vast majority of published results within scheduling theory are therefore related to the SMSP.

2.1.1 Job Shop Scheduling Problem

One of the most extensively studied models within scheduling theory is the *job shop scheduling problem* (JSP). The JSP belongs to a branch of scheduling problems referred to as *shop* scheduling problems along with two other classes: the flow shop- and open shop scheduling problems (for more information on what distinguishes these problem classes see [6]). The popularity of the JSP as a topic of study is due to that it is generally considered to be a good representation of the overall field, as well as a useful umbrella term encompassing a class of problems in computational complexity theory that are notoriously difficult to solve.

The JSP belongs to a class of problems known as NP-hard [10], meaning it is impossible to solve an arbitrary instance of the JSP to optimality in polynomial time (see [6, 9] for more on complexity theory). There are a wide spectrum of strategies for attacking the JSP (see [12] for a review of many of these), however, these approaches can be divided into two general categories: *exact/optimizing methods* and *approximative methods*. Exact/optimizing methods are, as their name suggests, methods that aim to find an optimal solution exactly. Due to the NP-hardness of the problem, the time requirement for solving the JSP using exact methods increases exponentially with problem size. Consequently, most traditional optimization methods for finding an exact solution can only be used for smaller scale instances if one is to find a solution within a reasonable amount of time. Approximative methods consist of finding a near optimal solution instead within a moderate amount of time. The bulk of research for solving larger, more complex, instances of the JSP has therefore been focused on approximative methods.

What characterizes the class of *shop* problems is that they are *multi-operation* models, meaning that associated with each job to be scheduled are a set of operations that each need to be completed to complete the job. Moreover, each operation is also associated with a single resource that can process it (dedicated machine). In the case of the JSP the sequence of operations for each job must also be processed in a specific order. The constraints indicating that one operation must precede another are called *precedence constraints*. Furthermore, the problem is subject to *capacity constraints* (or *disjunctive constraints*), that enforce that the resources are only able to process one operation

at a time, and that each operation can only be processed by one resource at a time. Unless otherwise stated, *preemption* is also not allowed meaning each operation must be processed uninterrupted until completion once it has started. To formulate the JSP more precisely we will start by introducing the following notation:

$\mathcal{J}$ the set of n jobs; $j \in \mathcal{J} := \{1, \ldots, n\}$,
$\mathcal{N}_j$ the set of n_j operations of job j; $i \in \mathcal{N}_j := \{1, \ldots, n_j\}$,
$\mathcal{K}$ the set of m resources; $k \in \mathcal{K} := \{1, \ldots, m\}$,
μ_{ij} resource that can process operation i of job j, ($\mu_{ij} \in \mathcal{K}$),

along with precedence relations of the form:

$O_{1j} \rightarrow O_{2j} \rightarrow \cdots \rightarrow O_{n_j j}$, for $j \in \mathcal{J}$,

where O_{ij} = operation i of job j, for $j \in \mathcal{J}$, $i \in \mathcal{N}_j$.

The JSP can now be expressed as finding an optimal processing sequence, with respect to a chosen objective function, of jobs $j \in \mathcal{J}$ (consisting of operations $i \in \mathcal{N}_j$) on resources $\mu_{ij} \in \mathcal{K}$. This processing sequence must also satisfy the aforementioned precedence constraints. The processing time of an operation i of job j is denoted p_{ij}. Depending on the problem, release dates r_j (i.e., the earliest point at which a job j can be processed) as well as due dates d_j (i.e., a desired completion time for a job j), may also be defined.

2.1.2 Flexible Job Shop Scheduling Problem

The FJSP (also referred to as the multipurpose machine problem) is a generalization of the JSP in the sense that a given operation may be processed not only by a single resource, but on any one of a given set of, not necessarily identical, available resources. For each operation in the FJSP we thus replace the μ_{ij} (as defined for the JSP above) with a given set of resources on which that operation can be processed:

$\mathcal{M}_{ij}$ the set of resources that can process operation i of job j, ($\mathcal{M}_{ij} \subseteq \mathcal{K}$).

The size of the set $\mathcal{M}_{ij}$ models the varying degree of flexibility of operations i of job j for a problem instance. If $\mathcal{M}_{ij} \subset \mathcal{K}$ for an operation in an instance of the FJSP, then this operation is said to have *partial flexibility*. If $\mathcal{M}_{ij} = \mathcal{K}$ for an operation then it is said to have *total flexibility*. Since the processing time of an operation can now vary depending on which resource that it is being processed on we will need an additional index to keep track of this. For the FJSP the processing time of operation i of job j on resource k is therefore denoted p_{ijk}. The FJSP is at least as difficult to solve as the JSP since in addition to finding an optimal sequence of operations on resources, it is also necessary to choose which resource will process each operation [14]. Since the FJSP is a generalization of the JSP, it also belongs to the class of NP-hard problems.

2.1.3 Decision variables

Although exact mathematical optimization models of scheduling problems have been formulated since the late 1950s, the computational strength of computers has historically only allowed for the solution of simpler problems and smaller instances, which are often not relevant for real application. Like the JSP, the NP-hard structure of the FJSP has caused the majority of research on this problem to primarily focus on approximative methods (see for example [5, 18, 27]), as opposed to methods in which an optimal solution is calculated exactly. Approximative methods generally do not provide solutions that, with respect to their objective function, provide values that have a guaranteed distance from the optimum. They can, however, be sufficiently effective for many problem instances, or for specific applications, and may also serve as a complement to exact methods by providing good feasible starting solutions.

What distinguishes one mathematical formulation of a scheduling problem from another typically stems from the type of binary decision variables that are chosen to represent the sequencing of operations on resources. This choice effects not only the structure of the solution polyhedron (more on this in section 2.2.2) but also how objective functions and additional constraints can be modeled by means of linear equations and inequalities. In the case of modeling the FJSP, the choice of decision variables has much to do with how one chooses to model the dimension of time, i.e., how one keeps track of when operations start and finish. For example, in the papers [3, 8, 17] different forms of MILPs are proposed for solving the FJSP. The decision variables used in these models are based on those originally formulated by Manne in 1960 (see [15]), for an integer programming model of the SMSP. In [15] the decision variables are defined as:

$$y_{jq} = \begin{cases} 1, & \text{if job } j \text{ precedes job } q \\ 0, & \text{otherwise.} \end{cases}$$

The basic idea with this type of decision variable is that by keeping track of the ordering of jobs (or operations) in an optimal schedule one can deduce the start and finish times of the jobs by combining their ordering and processing times. Models using this type of decision variable are referred to as *Manne family* models and are widespread within the field of operations research.

The model of the FJSP that is used in this thesis handles the dimension of time in a different way. In this case the planning period is divided into an integer number of time periods of equal length. The decision variables are then defined as:

$$x_{ijku} = \begin{cases} 1, & \text{if operation } i \text{ of job } j \text{ is processed on resource } k \text{ at time } u \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

Models that make use of this type of decision variable, that are indexed by both job and time period, are commonly referred to as *time-indexed* (TI) models. TI models have

long been used both in exact methods as well as approximative algorithms for a variety of scheduling problems. The first TI integer programming model of a JSP can be attributed to Bowman in 1959 (see [4], referred to in this paper as the *schedule-sequencing problem*). In Chapter 1 we also mentioned a TI formulation of the SMSP (see [21]), from which the family of VIs that are being adapted to the TI model in this paper originate. In the Ph.D. thesis [2] a TI formulation also outperformed three other MILP models for the PMSP. TI formulations for solving the FJSP exactly seem to be less common however. The model of the FJSP (originally developed in [22]) that is used in this thesis is, to our knowledge, the first of its kind.

To compare different models arising from different choices of variables one often compares the quality of the lower bounds obtained from their corresponding LP relaxations, as well as the number of variables and constraints that they require [20]. The most common reason for using a TI model is that they tend to provide very strong lower bounds relative to other MILP formulations. Their main drawback, however, is that they also tend to induce very large models [20, 21]. Even relatively small instances can generate a huge number of constraints and variables. We can see this just by observing the indices of (2.1) above, where the number of decision variables adds up to (operations) $\times$ (jobs) $\times$ (resources) $\times$ (time periods). The resulting number of constraints also becomes very large (more on this in chapter 3). As a result, the memory required to store a problem instance as well as the time required to solve even the LP relaxation of a problem may become intractably large. A large portion of research on time-indexed models has therefore been dedicated to finding ways to cut down on their size, as well as methods for solving them faster. Since the number of variables and constraints tend to depend heavily on the number of time periods chosen in the discretization of the planning period, one approach to reducing memory and time requirements is to try and keep this number as small as possible while still attempting to maintain a quality solution. This is, for example, the purpose of the iterative solution procedure developed in [24] (also briefly described in section 1.1). Another approach is to search for VIs, in particular facet-inducing VIs, that strengthen a given TI model in the sense that they improve the LBs provided by the TI model's LP relaxation. This can potentially improve the time to solve the model as well. We will go into more detail on this approach in section 2.2.2.

2.1.4 Objective functions

As previously mentioned, the optimality of a feasible solution to a scheduling problem is determined by whether or not it minimizes (or maximizes) a chosen objective function. The most commonly used objective in research is the minimization of so-called *makespan*, i.e., the time difference between the starting time of the earliest scheduled job and the time in which all jobs have been completed. According to the survey [12], the main reason why the makespan objective is so popular is that it is easily modelled and was also one of the first objectives used to study scheduling problems in the 1950s. To formulate the makespan objective more precisely, let C_j denote the completion time of job j and assume that the starting time of the earliest scheduled job is 0. Then the

makespan ($C_{\max}$) is defined as:

$$C_{\max} := \max\{C_i : i = 1, \ldots, n\}.$$

Other frequently used objectives are related to the earliness/tardiness of jobs, where the goal is for example to minimize some cost (or maximize profit) associated with processing a job to completion earlier then, or after, its due date (i.e., desired completion time). To clarify, if we have due date d_j of job j, the earliness (E_j) of job j can be defined as:

$$E_j := \max\{0, d_j - C_j\},$$

and tardiness (T_j) of job j as:

$$T_j := \max\{0, C_j - d_j\}.$$

A job j is thus early (respectively, tardy) if $E_j > 0$ (respectively, $T_j > 0$). There are several other types of objectives that are used depending on the purpose of the model. The objective function may also be comprised of both single or multiple optimality criteria. For a more comprehensive introduction to standard objective functions see [6]. A discussion regarding the choice of objectives that are used in this thesis can be found in section 3.1.

2.2 Mixed Integer Linear Programming

In this paper the FJSP is formulated as a mathematical optimization problem called a *mixed integer linear program* (MILP). To model a scheduling problem in this way some basic concepts and definitions are required. A brief introduction to these is provided in the following section. For a more in-depth treatment of the topics presented here we refer to [16], the main resource that we will be using for this section.

2.2.1 Integer and combinatorial optimization

Integer and combinatorial optimization refers to the problem of maximizing or minimizing an objective function consisting of one or several variables. These variables are subject to inequality and equality constraints, as well as integrality restrictions on some or all of the variables. In the case of a MILP, the objective function as well as the inequality and equality constraints are all linear. Furthermore, some, but not all, of the decision variables of a MILP are also restricted to integer values. These integrality restrictions are what allow MILPs to capture the discrete nature of some decisions. For example, many models utilize binary decision variables, whose values are restricted to 0 or 1, to represent whether or not an action is taken. In the case of a scheduling problem for instance, a binary decision variable may represent the choice of whether or not one allocates a machine to a job at a given time period. A MILP can be defined more formally as finding z where:

$$z = \max\{cx + hy \ : \ Ax + Gy \leq b, x \in \mathbb{Z}_+^n, y \in \mathbb{R}_+^p\}. \tag{2.2}$$

The $z \in \mathbb{R}^1$ in this definition is commonly referred to as the *objective value* (of a solution (x,y)) that we wish to maximize given an *objective function* $cx+hy$. Note that minimizing the objective function is equivalent to maximizing the negative of the same function, so this definition encompasses both minimization as well as maximization problems. It is also common to assume that the variables are non-negative, which can also be observed in this definition. To include the aforementioned binary decision variables one can, for example, replace $x \in \mathbb{Z}_+^n$ with $x \in B^n$, where B^n is the set of n-dimensional binary vectors.

From the definition of a MILP (2.2) we can see that a MILP instance is specified by data (c,h,A,G,b) where c is a $1 \times n$ vector, h is a $1 \times p$ vector, A an $m \times n$ matrix, G is an $m \times p$ matrix, and b an $m \times 1$ vector. The set of all viable solutions $S = \{x \in \mathbb{Z}_+^n, y \in \mathbb{R}_+^p \ : \ Ax + Gy \leq b\}$ is often referred to as the *feasible region* of an instance of (2.2) and a problem instance is said to be *feasible* if $S \neq \emptyset$. A feasible point (x^0,y^0) is called an *optimal solution* if it maximizes the objective function, that is, if

$$cx^0 + hy^0 \geq cx + hy \ \text{ for all } (x,y) \in S.$$

Not all feasible problem instances have optimal solutions however. For example if the objective function of a feasible maximization instance can be increased infinitely (i.e., $\exists (x,y) \in S$ such that $cx + hy > \omega$, $\forall \omega \in \mathbb{R}^1$) then the instance is called *unbounded*. If the data set of a MILP instance is rational (which it is for most practical cases), and is also feasible, then the MILP will either have an optimal solution or is unbounded (see [16] for a proof of this). Consequently, solving a rational MILP equates to either finding an optimal solution to the problem, or showing that it is unbounded or infeasible. Throughout this thesis we will, unless otherwise stated, assume that our data is rational.

A *linear integer program* (IP) can be described as a special case of a MILP for which *all* of the variables are restricted to be integer instead of just some. For an IP we instead have

$$z = \max\{cx \ : \ Ax \leq b, x \in \mathbb{Z}_+^n\},$$

with feasible region $S = \{x \in \mathbb{Z}_+^n : Ax \leq b\}$. If there are instead *no integrality restrictions* on any of the variables then the resulting model is called a *linear program* (LP). The LP can thus be defined as finding z where

$$z = \max\{hy \ : \ Gy \leq b, y \in \mathbb{R}_+^p\},$$

with feasible region $P = \{y \in \mathbb{R}_+^p : Gy \leq b\}$. The LP subproblem resulting from relaxing all integrality constraints of a MILP or IP is known as the *LP relaxation* of the MILP or IP. For example, the LP relaxation of the IP as defined above has feasible region $S_{LP} = \{x \in \mathbb{R}_+^n \ : \ Ax \leq b\}$.

2.2.2 Polyhedral theory

It is often useful to regard the set of feasible solutions to a LP as a *polyhedron* defined by the linear relations between the continuous variables. In the following subsection we will

provide a brief summary of the polyhedral theory associated with optimization problems that make up part of the research context of this thesis. We will start by introducing some basic definitions that will then lead us into some of the more fundamental results on this topic. Just as in the previous section we will be using notation from [16]. For omitted proofs and more elaborate descriptions of the topics in this section we refer to [16, 19]. For a paper on how polyhedral theory is applied to machine scheduling problems see [20].

Definition of outer representation, polytope, dimension and implicit equality

One of the most common ways of defining a *polyhedron* $P \subseteq \mathbb{R}^n$ is as the intersection between the solution sets of a finite number of linear inequalities (such as the feasible region of a LP):

$$P = \{x \in \mathbb{R}^n \ : \ Ax \leq b\}, \text{ where } (A,b) \text{ is a } m \times (n+1) \text{ matrix.}$$

This way of defining a polyhedron P is referred to as the *outer representation* of P. A polyhedron is said to be *rational* if (A,b) is a rational matrix, i.e., that P is the solution set to the rational linear system $Ax \leq b$. In this thesis we will only be considering rational polyhedra and will thus assume that this is the case throughout this section. Furthermore, a polyhedron P is called *bounded* if

$$\exists\ \omega \in \mathbb{R}^1_+ \text{ s.t. } P \subseteq \{x \in \mathbb{R}^n \ : \ -\omega \leq x_j \leq \omega \text{ for } j = 1,\ldots,n\}.$$

A *bounded polyhedron* is also called a *polytope*.

A key property of a polyhedron P is its *dimension*, denoted $\dim(P)$, which is defined as the number of affinely independent points in P minus one (i.e., $\dim(P) = k$ if the number of affinely independent points in P is $k+1$). A polyhedron P in $\mathbb{R}^n$ is called *full-dimensional* if $\dim(P) = n$. If we let $I = \{1,\ldots,m\}$ be the index set of the inequalities $Ax \leq b$, we can express $Ax \leq b$ as $a_i x \leq b_i$ for $i \in I$. An inequality $a_i x \leq b_i$ in a linear system of inequalities $Ax \leq b$ is called an *implicit equality* if any solution x of $Ax \leq b$ also satisfies $a_i x = b_i$. Note that if a polyhedron P contains an implicit equality in its description, then P lies in the hyperplane that is defined by this implicit equality and hence cannot be full-dimensional.

Definition of valid inequality, face, facet and vertex

Given a polyhedron $P = \{x \in \mathbb{R}^n \ : \ Ax \leq b\}$, an important question to adress is whether or not any of the inequalities $a^i x \leq b_i$ of $Ax \leq b$ can be dropped, and which are necessary in the description of P. An inequality that can be dropped without changing the feasible region of P is called *redundant*. An inequality $\pi x \leq \pi_0$, or (π,π_0), is called a *valid inequality* (VI) for P if it is satisfied by all points $x \in P$. If (π,π_0) is a VI for polyhedron P and $F = \{x \in P \ : \ \pi x = \pi_0\}$, then F is called a *face* of P that is *represented by* (π,π_0). A face F is said to be *trivial* if $F = \emptyset$ or $F = P$. All other faces

are called *proper*. Moreover, when a face F of P represented by (π,π_0) is nonempty (i.e., $\max\{\pi x \ : \ x \in P\} = \pi_0$) we say that ($\pi$,$\pi_0$) *supports* P. Note that any inequalities $a^i x \leq b_i$ that are not supports of P are redundant and can hence be discarded.

A face F of a polyhedron P is itself a polyhedron with $\dim(F) \leq \dim(P)$. If F is not empty or the whole of P (i.e., proper), then $0 \leq \dim(F) < \dim(P)$. Often, within mathematical programming, the most interesting faces to find are those of the highest and lowest dimensions. A proper face F of P of the highest dimension (i.e., $\dim(F) = \dim(P) - 1$) is called a *facet* of P. A proper face F of P of the lowest dimension (i.e., $\dim(F) = 0$) is called a *vertex* of P.

Facet inducing valid inequalities

Using the definitions stated above the following important theorem will be stated without proof (see [16]).

Theorem 2.2.1. *A full-dimensional polyhedron P has a unique (to within scalar multiplication) minimal representation by a finite set of linear inequalities. In particular, for each facet F_i of P there is an inequality $a^i x \leq b_i$ (unique to within scalar multiplication) representing F_i and $P = \{x \in \mathbb{R}^n \ : \ a^i x \leq b_i \text{ for } i = 1, \ldots, t, \ t \in \mathbb{N}\}$.*

In other words, if $Ax \leq b$ is a minimal representation of a full-dimensional polyhedron P (in the sense that no inequality of $Ax \leq b$ is redundant), then each inequality of $Ax \leq b$ induces a facet of P and each facet of P is induced by exactly one of these inequalities. This means that the facets of a polyhedron are both sufficient and necessary for its description and hence the best type of inequality we can look for if we wish to describe a polyhedron minimally in terms of linear inequalities are those that are facet inducing. If the polyhedron P is, however, not full-dimensional then its minimal representation will also contain implicit equations, i.e., $Bx = d \ \forall \ x \in P$. In this case $P = \{x \in \mathbb{R}^n \ : \ Ax \leq b, Bx = d\}$, where each inequality of A still corresponds to a facet inducing inequality and no equation in $Bx = d$ is implied by any of the other equations in $Ax \leq b$ or $Bx = d$.

Definition of inner representation, extreme points and extreme rays

Equivalent to the outer representation of a polyhedron as stated above (i.e., as the intersection of a finite number of closed halfspaces), a polyhedron may also be defined in terms of its extreme points and extreme rays. This way of defining a polyhedron P is commonly known as the *inner representation* of a P. Although both the outer- and inner definitions of polyhedra are equivalent, they both have their own advantages and provide alternative ways of describing the feasible sets of MILP formulations. To describe the inner representation of a polyhedron more formally we will need to introduce additional definitions and notation.

We call $x \in P$ an *extreme point* of P if there do not exist any $x^1,\ x^2 \in P$ with $x^1 \neq x^2$ such that $x = \frac{1}{2}x^1 + \frac{1}{2}x^2$. Note that all vertices of a convex polyhedron are extreme points.

Let $P^0 = \{r \in \mathbb{R}^n\ :\ Ar \leq 0\}$. If $P = \{x \in \mathbb{R}^n\ :\ Ax \leq b\} \neq \emptyset$, then $r \in P^0 \setminus \{0\}$ is called a *ray* of P. The P^0 in this definition is also called the *recession cone* (or characteristic cone) of P and can be described as the set of the directions of P that go to "infinity". A point $r \in \mathbb{R}^n$ is a ray of P if and only if for any point $x \in P$ the set $\{y \in \mathbb{R}^n\ :\ y = x + \lambda r, \lambda \in \mathbb{R}^1_+\} \subseteq P$. Note that a nonempty polytope contains no rays. We call a ray r of P an *extreme ray* if there do not exist rays $r^1, r^2 \in P^0$ where $r^1 \neq \lambda r^2$ for any $\lambda \in \mathbb{R}^1_+$ such that $r = \frac{1}{2}r^1 + \frac{1}{2}r^2$. Moreover, if $P \neq \emptyset$ a ray r is an extreme ray of P if and only if $\{\lambda r\ :\ \lambda \in \mathbb{R}^1_+\}$ is a one-dimensional face of P^0.

The *convex hull* of a set of points $V \subseteq \mathbb{R}^n$, denoted $\text{conv}(V)$, is defined as the intersection of all convex sets containing the set of points V. Similarly, the *conical hull* of a set of rays $R \subseteq \mathbb{R}^n$, denoted $\text{cone}(R)$, is the intersection of all convex cones containing the rays in R.

We will now state the following theorem on the inner representation of polyhedra without proof (see [16]).

Theorem 2.2.2. *(Minkowski's Theorem). If $P \neq \emptyset$ and $\text{rank}(A) = n$ then*

$$P = \left\{ x \in \mathbb{R}^n\ :\ x = \sum_{k \in K} \lambda_k x^k + \sum_{j \in J} \mu_j r^j; \sum_{k \in K} \lambda_k = 1; \lambda_k \geq 0, k \in K; \mu_j \geq 0, j \in J \right\},$$

where $\{x^k\}_{k \in K}$ is the set of extreme points of P and $\{r^j\}_{j \in J}$ is the set of extreme rays of P.

A polyhedron can in other words be defined by the sum of the convex combination of its extreme points and the conic combination of its extreme rays. An important converse to this theorem (see Weyl's theorem in [16]) is that an arbitrary sum of a convex combination of points and a conic combination of points in $\mathbb{R}^n$ is also a polyhedron.

2.2.3 Integral polyhedra

MILPs and IPs are most often much more difficult to solve than LPs. One reason for this is that an optimal solution to a LP, if it exists, is always found at an extreme point of its feasible region. For the LP relaxation of a MILP or IP, however, an extreme point may contain infeasible, fractional variable values. Consider a general IP:

$$\max\{cx\ :\ Ax \leq b, x \in \mathbb{Z}^n_+\}.$$

The feasible region S of this problem can be defined as $S = \{x \in \mathbb{Z}^n_+\ :\ Ax \leq b\} = P \cap \mathbb{Z}^n_+$ where $P = \{x \in \mathbb{R}^n_+\ :\ Ax \leq b\}$. As a consequence of the reverse of Minkowski's theorem it can be shown that $\text{conv}(S)$, i.e., the convex hull of integer points in P, is a rational polyhedron. Thus, there exists a rational linear system $\bar{A}x \leq \bar{b}$ that constitutes an

outer representation of conv(S), i.e., conv(S) = $\{x \in \mathbb{R}^n_+ \ : \ \bar{A}x \leq \bar{b}\}$. A similar result holds true for MILPs as well. This is important since the extreme points of conv(S) are integers (i.e., conv(S) is an *integral polyhedron*) and hence, since the optimal solutions to a LP are found at its extreme points, the original IP (or MILP) can theoretically be "reduced" to the LP:

$$\max\{cx \ : \ \bar{A}x \leq \bar{b}, x \in \mathbb{R}^n_+\}.$$

The LP of maximizing (or minimizing) the objective function over conv(S) that is equivalent to solving the IP or MILP is sometimes referred to as the *polyhedral approach* to solving the IP or MILP.

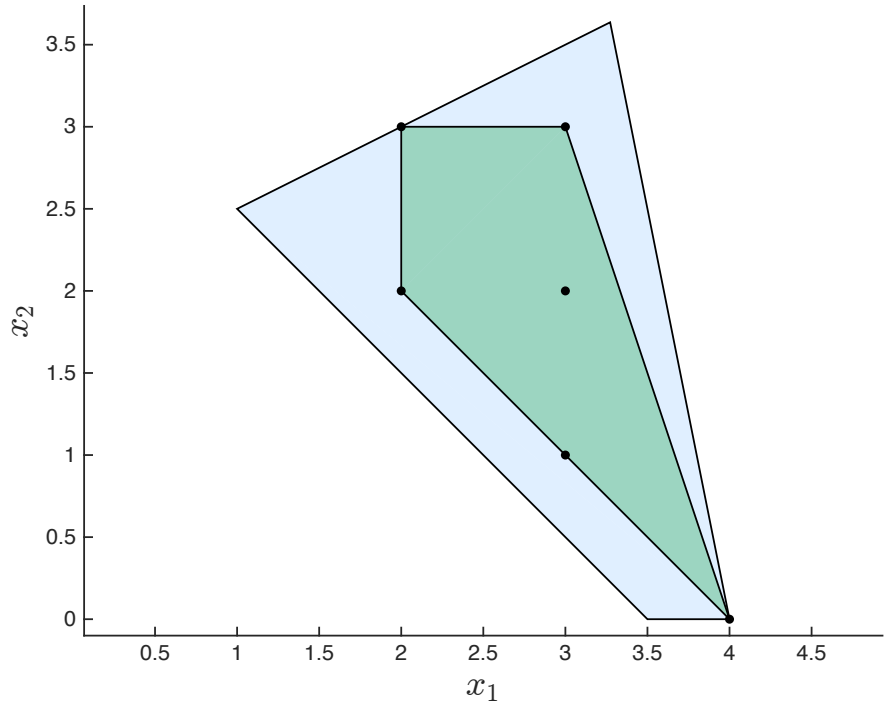


Figure 2.1: Two-dimensional example of an IP with polytope P defined by constraints $Ax \leq b$ and $x \in \mathbb{R}^n_+$ (light-blue outer-polytope), feasible integer points in $S = P \cap \mathbb{Z}^n_+$ (black dots), and polytope conv(S) defined by constraints $\bar{A}x \leq \bar{b}$ (green inner-polytope). Example taken from [16], p.206.

This means that we can, in principle, use LP techniques to solve IPs or MILPs. Unfortunately however, the linear system of inequalities $\bar{A}x \leq \bar{b}$ in the outer representation of conv(S) may be (and in fact often are) much larger than the original system of inequalities $Ax \leq b$ which can make the resulting LP "reduction" difficult or even impossible to solve. An even more basic problem when attempting to "reduce" an IP or MILP in this way is how to construct the inequalities $\bar{A}x \leq \bar{b}$, both in theory and in practice (see section (2.3) for more on this). Even if one cannot find a complete outer description of conv(S), a partial description may sometimes still produce better lower bounds on the

optimal value which can in turn be useful in obtaining good approximate solutions, or proving the optimality of a solution found. A partial description of conv(S) may even be sufficient to find a feasible optimal value. More on this in the following subsection.

2.2.4 Strong and weak formulations

A commonly adopted strategy in algorithms for solving MILPs and IPs is to first solve their LP relaxation and then use the information obtained from this solution to reduce the size of the feasible region to the unrelaxed problem, (more on this in section 2.4) or divide the original problem into simpler subproblems (e.g., branch-and-bound). The LP relaxation is most often solved using some variant of the well-known simplex algorithm. The simplex algorithm basically searches for an optimal solution at the extreme points of the LP's feasible region and, although requiring an exponential amount of time to terminate in the worst case, is for the most part quite efficient for the majority of LPs as well as highly optimized due to its popularity (for a walkthrough of the simplex algorithm see [16]). Via the solution to the LP relaxation one obtains (assuming that we are minimizing) a *lower bound* (LB) on the objective value of the optimal MILP or IP solution. This LB is also known as the *LP relaxation bound*. The constraints that form the feasible region of a MILP or IP model are called *strong* if the LB obtained by the corresponding LP relaxation is close to the optimal objective value. If instead the LP relaxation yields a LB that is far from the optimal objective value then the constraints describing the feasible region of the MILP or IP model are called *weak*.

To clarify, a 2D example of a weak and a strong formulation of the same feasible set of a MILP is presented in figure 2.2. In this figure the feasible points of the MILP model are represented by small, filled circles. The small, non-filled circles are points outside of the MILPs feasible region. We have two variables (along the vertical and horizontal axis) that are subject to non-negativity constraints, as well as two sets of constraints corresponding to a weak and a strong formulation. The solid lines, marked with (1) and (2), represent the constraints of the weak formulation. The constraints of the strong formulation are represented by blue dashed lines. The arrow at the origin indicates the direction of decreasing objective function values (again, assuming we are minimizing). The LP relaxation of the weak formulation yields a fractional optimal solution, marked by the non-filled circle at the intersection of the solid lines marked (1) and (2). The optimal solution to the MILP model equals to that of the LP relaxation of the strong formulation, which is marked by the filled circle at the intersection of the two dashed lines.

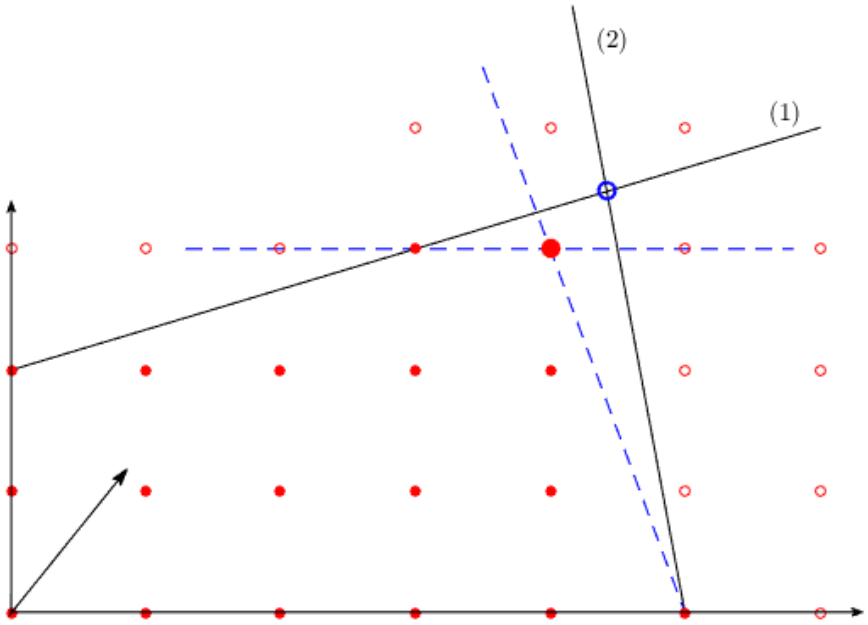


Figure 2.2: Weak and strong formulations of the same feasible set of a MILP model. The LP relaxation of the weaker formulation, whose constraints are indicated by solid lines, yields a solution with fractional variable values. The LP relaxation of the strong formulation however, whose constraints are indicated by dashed lines, yields a solution that is also optimal for the MILP.

By adding stronger constraints (represented by linear inequalities) it is thus possible to cut off fractional solutions from the feasible region. The gap between the optimal value of the MILP and the optimal value of its corresponding LP relaxation is then reduced, and if the constraints represent the feasible region strongly enough, the optimal solution to the MILP can even be found at an extreme point of its LP relaxation.

2.3 Theory of valid inequalities

In subsection 2.2.3 we established that solving an IP or MILP can theoretically be reduced to solving a LP over polyhedra. Furthermore we discussed facet inducing VIs that are both sufficient and necessary in any minimal outer representation of a polyhedron. In figure 2.2 of subsection 2.2.4 we saw an example of this, where adding stronger, facet-inducing VIs reduced the feasible region of a MILP to the point where it was solvable as a LP. This is an example of an ideal scenario however, since finding facet-inducing VIs that support the convex hull of feasible integer points to a MILP or IP has in practice often proven to be very difficult. If we let S again denote the feasible region of an arbitrary MILP or IP problem, and P denote the feasible region of its corresponding LP relaxation, then it is clear that any VIs that are valid for P are also valid for S, since $S \subseteq P$. Furthermore, unless $\text{conv}(S) = P$, there are VIs for S that are not valid for P. Even if such a VI of S, that is not valid for P, is not enough to define $\text{conv}(S)$, it can still

be useful in the sense that by reducing the set of feasible solutions to the LP relaxation of P one can potentially close some of the gap between the LP relaxation bound and the solution to the original MILP, if the VI is active in the optimal solution of the LP relaxation.

2.4 Cutting-plane algorithms

In the following section we will discuss *cutting-plane algorithms*, a general approach to solving, or approximately solving, IP and MILP problems. Within the realm of mathematical optimization, a cutting-plane algorithm refers to an algorithm that iteratively refines a feasible set or objective function using VIs. VIs that refine the problem are called *cuts* or *cutting-planes*.

An early version of a cutting-plane algorithm for solving MILPs was proposed by Gomory in the late 50s (see [16] for a walkthrough of this work as well as references to the original paper). However, due to the numerical instability as well as the poor convergence speed of Gomory's algorithm it was considered to be impractical for solving larger problems. It was not until decades later, after the discovery of the ellipsoid method in the 70s, that new interest in cutting-plane methods arose. With some adjustments to the algorithm, cutting-plane methods have become one of the most important contributors to the solving of IP and MILP problems over recent decades.

The theory of cutting-planes is quite extensive and much is beyond the scope of this thesis. Generally speaking however, todays cutting-plane algorithms consist of first finding a theoretically strong class of VIs to a target IP or MILP (before computations have started) and then testing to see whether some current solution to the corresponding LP relaxation violates any one or several of the inequalities in this class. Let Q denote the IP problem $\min\{cx \ : x \in P, x \in \mathbb{Z}^n_+\}$, where $P = \{x \in \mathbb{R}^n_+ \ : \ Ax \leq b\}$. Let $S = P \cap \mathbb{Z}^n_+$ denote the feasible region of Q. One way of visualizing the cutting-plane method is by considering the polyhedron $P^k \in \mathbb{R}^n_+$ that is generated in each iteration k. One can think of P^k as the current approximation of the target polyhedron $\text{conv}(S)$. These polyhedra are thus nested in the sense that

$$\text{conv}(S) \subseteq \cdots \subset P^{k+1} \subset P^k \subset \cdots \subset P^0 = P.$$

Let Π denote a finite class of theoretically strong VIs for $\text{conv}(S)$ that are known. In an ideal scenario this class gives a complete description of $\text{conv}(S)$ although more frequently it will only yield a partial description. Even if Π completely describes $\text{conv}(S)$, however, the total number of inequalities in Π will often be too large for a LP solver to handle. A general cutting-plane algorithm can be formulated as follows:

Cutting-plane algorithm

1. (initialization)	Take a set of VIs $(A^0,b^0) \subset \Pi$ small enough to be handled by a LP solver. Let $k := 0$.
2. (optimization)	Solve the corresponding LP: $\max\{cx \ : \ A^k x \leq b^k\}$ to obtain optimal LP solution x^k.
3. (optimality check)	If x^k is feasible to $\text{conv}(S)$ then x^k is optimal in which case we terminate the algorithm.
4. (separation)	If x^k is not optimal, check if it violates one or several of the remaining VIs in Π. If not then stop.
5. (enhance)	If one or more VIs in Π are violated, append violated VIs to (A^k,b^k) and denote this new set (A^{k+1},b^{k+1}). Let $k := k+1$ and return to step 2.

Note that if the class of VIs Π is very large then nothing will stop this algorithm from adding VIs to a degree that a LP solver cannot handle, if one of the termination criteria is not reached. Furthermore, if Π is only a partial description of $\text{conv}(S)$ then we are not guaranteed to find an optimal solution using this algorithm.

In each iteration k of the cutting-plane algorithm the objective value $z^k = cx^k$ of the optimal LP solution x^k gives us a lower bound on the optimal objective value of $\text{conv}(S)$. If we let z^* denote the optimal objective value of $\text{conv}(S)$ then a natural theoretical measure of the cutting-plane algorithm's success for a given problem instance is the so-called *optimality gap*: $z^k - z^*$. This is, in other words, the difference between the best lower bound found so far via the cutting-plane algorithm and the optimal objective value of the unrelaxed problem. If the optimal objective value z^* is not known (since this is often what one wishes to find), the objective value of the best solution found so far is used instead.

3

Mathematical model

In this chapter the MILP formulation of the FJSP that is used in our cutting-plane algorithm is introduced. We will start by briefly describing the background to the MILP model's development, to clarify the underlying motivation to some of the modeling decisions that have been made. This will later also relate to a discussion on our choice of objective functions. The model in its original form (as it is found in [22]) will be stated in section 3.2. To simplify implementation of the cutting-plane algorithm an adjusted but equivalent version of this model has been formulated. This adjusted model will be the topic of section 3.2.3. In section 3.3 we will describe the new VIs first in the form from which they originate, for the SMSP, followed by their extension to the FJSP and an intuitive description of how they work.

3.1 Model formulation

As mentioned in section 1.1, the mathematical model used in this thesis was originally formulated to model a real-life scenario: the scheduling of ten resources in a multitask production cell, five of which are multipurpose machines. In the licenciate thesis [22] the first model that was implemented and tested was a Manne family model referred to as the *engineer's model*. Preliminary computational testing found the engineer's model to be too slow for larger, more realistically sized instances which motivated the decomposition of the full scheduling problem (i.e., the scheduling of all ten resources of the multitask cell) into two subproblems.

The workload on the multipurpose machines is much higher then that of the other five resources. Therefore, the first subproblem (referred to as the *machining problem*) consists of constructing an optimal schedule for only the multipurpose machines; a FJSP. After the machining problem has been solved, the optimal scheduling of the multipurpose machines is then used to create a feasible schedule for the remaining five resources in what is referred to as the *feasibility problem*.

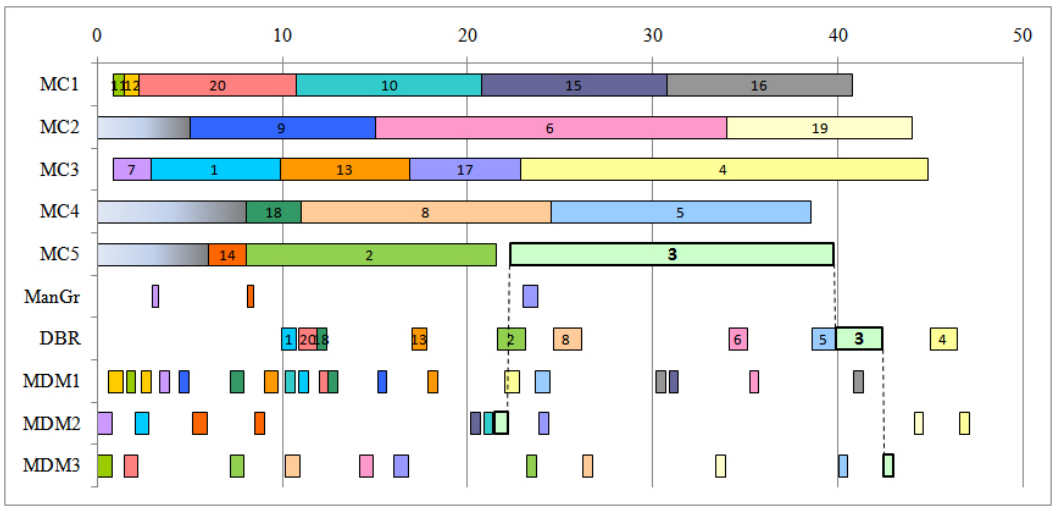


Figure 3.1: An example of a schedule based on real data. The work load on the five multipurpose machines (MC1–5) is much higher than that of the other resources. The machining problem consists of finding an optimal schedule for the five multipurpose machines. The route of job 3 is indicated by dotted lines.

Dividing the full scheduling problem in this way resulted in significantly reduced computation time, but was still not considered fast enough for real applications. Since the vast majority of computation time was being spent on the machining portion (i.e., the FJSP portion) of the two subproblems, it was decided that the Manne family model of the machining problem would be reformulated as a time-indexed model instead, to see if this would operate faster. Two versions of a time-indexed model were formulated, one making use of so-called *plateau* decision variables (see [22] for more on this model) and the other utilizing so-called *nail* decision variables. The time-indexed model using nail variables outperformed all others tested, which is why this model was chosen to be used in this thesis.

Choice of objective function

Within the framework of a MILP model, any linear objective that can be formulated in terms of the variables used in the model can be utilized. Choosing a suitable objective function for a model of a scheduling problem is no trivial task however, and depends greatly on the scheduling problem's context and consequent priorities. The choice of objective function can also have significant impact on the performance of a scheduling method and will affect which VIs are added in our cutting-plane algorithm and their subsequent effect on the solution space. As stated in section 2.1.4, the most common objective function used for research within scheduling theory is the minimization of *makespan*. To maintain comparability between the results from the model used in this thesis and other research done on solving the same test instances, we will also be including makespan as one of our objectives. Use of the makespan objective can, however, sometimes be ill-suited to problems where not all jobs are available for processing at

the same time, but are instead expected to arrive at given release dates (as in the case of the multitask cell at GKN Aerospace). For example, if the release date of one job is much later than all others, such that the corresponding job must be scheduled far later in the planning period, then use of the makespan objective will allow all other jobs with far earlier release dates to be scheduled arbitrarily without having an effect on the objective value. If, for example, the reliability of the due dates is a priority (such as in a production with a just-in-time objective) then use of the makespan objective will thus sometimes result in unsatisfactory scheduling. An objective that strives to minimize the tardiness of jobs can help maintain that due dates are met, and aid in keeping a reliable pace of production. Prioritizing tardiness on its own, however, can instead lead to arbitrary scheduling for instances with no tardy jobs.

According to the managers of the multitask cell, the main priorities when scheduling production are maximizing utilization of the multitask cell and minimizing tardiness of jobs. To reflect these priorities and with the goal of achieving realistic schedules in mind, an objective function that takes into account both the minimization of total completion times as well as total tardiness was also included as a topic of study in the Ph.D. thesis [23]. For the same reasons we will also be including this objective in this thesis. For instances in which there are no tardy jobs, the objective equals the minimization of the sum of completion times, and will hence strive to shorten production lead times. If the scheduling procedure consistently produces schedules with no tardy jobs the manager of the production has the opportunity to set more challenging due dates, which will in turn shorten planned production lead times. Since the corresponding costs of tardy jobs vary depending on the job and current situation, weights linked to the tardiness of individual jobs are also included in the objective function. Let C_j again denote the completion time of job j from the set $\mathcal{J}$ of all jobs to be scheduled, and assume that the starting time of the earliest scheduled job is 0. The makespan $C_{\max}$ is then defined as $C_{\max} := \max\{C_j : j = 1, \ldots, n\}$. If we have due date d_j of job j, the tardiness (T_j) of job j can be defined as: $T_j := \max\{0, C_j - d_j\}$. Furthermore let β_j denote the associated cost of job j being tardy. An objective function including the minimization of completion times and weighted total tardiness can thus be expressed as:

$$\text{minimize} \sum_{j \in \mathcal{J}} (C_j + \beta_j T_j). \tag{3.1}$$

Although it is not desirable to complete jobs too early either (as this can cause choking in the system and increased lead times), minimizing job earliness is not included in the objective function. The reason for this choice is partly because minimizing earliness was simply not as highly prioritized as maximizing utilization and minimizing tardiness. In addition to this, however, allowing for a degree of earliness of jobs can compensate for the uncertainty of everyday reality in the multitask cell (e.g., operators getting sick, machine break-downs etc.), and in other words yield more robust schedules that have a margin of flexibility in the case of such events.

3.2 Time-indexed model

In the following section the time-indexed model of the FJSP, using nail variables, is presented as is found in [24].

3.2.1 Indices, sets, and parameters

Sets

$\mathcal{J}$	set of jobs; $j \in \mathcal{J} := \{1,\ldots,n\}$
$\mathcal{N}_j$	set of operations of job j; $i \in \mathcal{N}_j := \{1,\ldots,n_j\}$
$\mathcal{K}$	set of resources; $k \in \mathcal{K} := \{1,\ldots,m\}$
$\mathcal{M}_{ij}$	set of resources that are allowed to process operation i of job j, $(\mathcal{M}_{ij} \subseteq \mathcal{K})$
$\mathcal{H}$	set of time periods; $u \in \mathcal{H} := \{0,1,\ldots,H_{\max}\}$

Parameters

r_{ij}	release date of operation i of job j
p_{ijk}	processing time of operation i of job j on resource k
δ_{ij}	shortest possible remaining processing time from the starting time of operation i of job j to the completion of job j

To remind the reader, the FJSP can be stated as the problem of finding an optimal processing sequence, with respect to the chosen objective function, of jobs $j \in \mathcal{J}$ (consisting of operations $i \in \mathcal{N}_j$) on resources $\mathcal{M}_{ij} \subseteq \mathcal{K}$ within a given planning horizon $[0,\ H_{\max}]$. In the time-indexed formulation of the FJSP the planning horizon is discretized into $H_{\max}+1$ intervals of equal length. In this thesis the index $u \in \mathcal{H} = \{0,1,\ldots,H_{\max}\}$ refers to the time interval $[u,\ u+1]$ that begins at time u and ends at $u+1$.

Let O_{ij} denote operation i of job j for $j \in \mathcal{J}$ and $i \in \mathcal{N}_j$. There are precedence relations between operations of the form $O_{1j} \to O_{2j} \to \cdots \to O_{n_j j}$, for $j \in \mathcal{J}$, meaning no operation within a job j may be scheduled before the previous operation has been completed. In all test instances that are used in this thesis, all jobs are available to be processed from the beginning of time period 0 (i.e., $r_{1j} = 0\ \forall\ j \in \mathcal{J}$). Due to the precedence relations between operations and since the processing times are resource dependent, the release dates of operations following the initial operation are defined as $r_{ij} := r_{i-1,j} + \min_{k\in\mathcal{M}_{i-1,j}}\{p_{i-1,j,k}\}$ for $i = 2,\ldots,n_j,\ j \in \mathcal{J}$. Similarly, $\delta_{ij} := \delta_{i+1,j} + \min_{k\in\mathcal{M}_{ij}}\{p_{ijk}\}$ for $i = n_j-1,\ldots,1,\ j \in \mathcal{J}$, and $\delta_{n_j j} := \min_{k\in\mathcal{M}_{n_j j}}\{p_{n_j jk}\}$. To simplify future writing we will use the release dates r_{ij} and δ_{ij} to define the following set of time intervals in which it is possible for operation i of job j to begin and be processed to completion:

$$\mathcal{H}_{ij} := \{r_{ij},\ldots,H_{\max}-\delta_{ij}\},\ \text{the time window of operation } i \text{ of job } j.$$

3.2.2 Original model

For each operation $i \in \mathcal{N}_j$ of job $j \in \mathcal{J}$, resource $k \in \mathcal{K}$ and time interval $u \in \mathcal{H}$ we define the binary decision variables:

$$x_{ijku} = \begin{cases} 1, & \text{if operation } i \text{ of job } j \text{ starts in the beginning of time period } u \text{ on resource } k \\ 0, & \text{otherwise.} \end{cases}$$

We will now present the model with the objective of minimizing the makespan, $C_{\max}$, of the schedule. This model will henceforth be referred to as *TI-make*. Throughout the remainder of this thesis define the operator $()_+$ such that $(z)_+ := \max\{z,0\}$ for any $z \in \mathbb{R}$.

TI-make

$$\text{minimize} \qquad C_{\max} \tag{3.2a}$$

$$\text{subject to} \qquad \sum_{k\in\mathcal{M}_{ij}} \sum_{u\in\mathcal{H}} x_{ijku} = 1, \qquad i \in \mathcal{N}_j,\ j \in \mathcal{J}, \tag{3.2b}$$

$$\sum_{k\in\mathcal{K}\setminus\mathcal{M}_{ij}} \sum_{u\in\mathcal{H}} x_{ijku} = 0, \qquad i \in \mathcal{N}_j,\ j \in \mathcal{J}, \tag{3.2c}$$

$$\sum_{j\in\mathcal{J}} \sum_{i\in\mathcal{N}_j} \sum_{\mu=(u-p_{ijk}+1)_+}^{u} x_{ijk\mu} \le 1, \qquad k \in \mathcal{K},\ u \in \mathcal{H}, \tag{3.2d}$$

$$\sum_{k\in\mathcal{M}_{ij}} \sum_{\mu=r_{ij}}^{u-p_{ijk}} x_{ijk\mu} - \sum_{l\in\mathcal{M}_{i+1,j}} \sum_{\nu=r_{i+1,j}}^{u} x_{i+1,jl\nu} \ge 0, \qquad \begin{array}{l} u\in\mathcal{H}_{i+1,j}, \\ i = 1,\dots,n_j-1,\ j\in\mathcal{J}, \end{array} \tag{3.2e}$$

$$\sum_{k\in M_{n_jj}} \sum_{u\in\mathcal{H}} (u + p_{n_jjk}) x_{n_jjku} \le C_{\max},\ j \in \mathcal{J}, \tag{3.2f}$$

$$x_{ijku} = 0, \qquad \begin{array}{l} u \in \mathcal{H} \setminus \mathcal{H}_{ij}, \\ k \in \mathcal{M}_{ij},\ i \in \mathcal{N}_j,\ j \in \mathcal{J}, \end{array} \tag{3.2g}$$

$$x_{ijku} \in \{0,1\},\ \begin{array}{l} i \in \mathcal{N}_j,\ j \in \mathcal{J},\ k \in \mathcal{K}, \\ u \in \mathcal{H}. \end{array} \tag{3.2h}$$

The constraints (3.2b) ensure that each operation i of job j is scheduled exactly once on the resources that are allowed to process it. Constraints (3.2c) set all variables corresponding to an operation to zero for the set of resources for which the operation is not allowed to be processed. Note that the constraints (3.2c) are redundant. However, in [23] it was discovered that their inclusion enabled the solver (AMPL-CPLEX12 (Fourer et al., 2002; IBM Corp., 2009) to parallelize computations leading to faster solution time

(w.r.t. clocktime). For this reason these constraints are included anyways. The capacity constraints (3.2d) make sure that at most one operation at a time can be processed on each resource. The precedence constraints (3.2e) ensure that no operation can be scheduled to start processing before the preceding operation of the same job has been completed. The makespan of the schedule is defined by constraints (3.2f). Constraints (3.2g) ensure that operation i of job j cannot be scheduled outside of its time window $\mathcal{H}_{ij}$. The integrality constraints (3.2h) ensure that the decision variables x_{ijku} can only attain values 0 or 1.

3.2.3 Operations and jobs → tasks

A slight change to the indices used in the TI-make model (3.2) has been made to simplify the extension of the new class of VIs to the FJSP, as well as the implementation of the cutting-plane algorithm. The adjustment that has been made is that the pairs of indices used to indicate an operation i of job j have been replaced with a single index t denoting what will henceforce be referred to as *tasks*. To show how this is done, first let (i,j) denote operation i of job j. All operations (i,j) for $i \in \mathcal{N}_j := \{1,\ldots,n_j\}$ and $j \in \mathcal{J} := \{1,\ldots,n\}$ are then enumerated according to the following procedure:

(i,j)	$\rightarrow$	t
(1,1)	$\rightarrow$	1
(2,1)	$\rightarrow$	2
	$\vdots$	
$(n_1,1)$	$\rightarrow$	n_1
(1,2)	$\rightarrow$	n_1+1
	$\vdots$	
(n_n,n)	$\rightarrow$	$\sum_{j\in\mathcal{J}} n_j$

Using this enumeration we can now define the following set:

$$\mathcal{T} \quad \text{set of tasks; } t \in \mathcal{T} := \{1,\ldots,\textstyle\sum_{j\in\mathcal{J}} n_j\}.$$

To use the set of tasks $\mathcal{T}$ as a replacement for the set of operations $\mathcal{N}_j$ and jobs $\mathcal{J}$, we will also need to keep track of which operations correspond to which job. To do this an additional set containing all of the final operations (henceforth referred to as *terminal tasks*) of each job is defined as follows:

$$\widetilde{\mathcal{T}} \quad \text{set of terminal tasks; } t \in \widetilde{\mathcal{T}} := \{n_1,\ (n_1+n_2),\ldots,\ (\textstyle\sum_{j\in\mathcal{J}} n_j)\}$$
(i.e. tasks corresponding to the final operation of each job).

Using the sets $\mathcal{T}$ and $\widetilde{\mathcal{T}}$ we can now replace the pairs of indices (i,j) in the model TI-make with the single index t. The adjusted version of TI-make using the task index t will henceforth be referred to as *TI-make-task*. The index k used for resources, as well as index u used for time steps will remain the same as before.

TI-make-task

$$\text{minimize} \qquad C_{\max} \tag{3.3a}$$

$$\text{subject to} \qquad \sum_{k\in\mathcal{M}_t}\sum_{u\in\mathcal{H}} x_{tku} = 1, \qquad t\in\mathcal{T}, \tag{3.3b}$$

$$\sum_{k\in\mathcal{K}\setminus\mathcal{M}_t}\sum_{u\in\mathcal{H}} x_{tku} = 0, \qquad t\in\mathcal{T}, \tag{3.3c}$$

$$\sum_{t\in\mathcal{T}}\sum_{\mu=(u-p_{tk}+1)_+}^{u} x_{tk\mu} \le 1, \qquad k\in\mathcal{K},\ u\in\mathcal{H}, \tag{3.3d}$$

$$\sum_{k\in\mathcal{M}_t}\sum_{\mu=r_t}^{u-p_{tk}} x_{tk\mu} - \sum_{l\in\mathcal{M}_{t+1}}\sum_{\nu=r_{t+1}}^{u} x_{t+1,l\nu} \ge 0, \qquad \begin{array}{l} u\in\mathcal{H}_{t+1}, \\ t\in\mathcal{T}\setminus\widetilde{\mathcal{T}}, \end{array} \tag{3.3e}$$

$$\sum_{k\in M_t}\sum_{u\in\mathcal{H}} (u+p_{tk})x_{tku} \le C_{\max}, \quad t\in\widetilde{\mathcal{T}}, \tag{3.3f}$$

$$x_{tku} = 0, \qquad \begin{array}{l} u\in\mathcal{H}\setminus\mathcal{H}_t, \\ k\in\mathcal{M}_t,\ t\in\mathcal{T}, \end{array} \tag{3.3g}$$

$$x_{tku}\in\{0,1\}, \quad \begin{array}{l} t\in\mathcal{T},\ k\in\mathcal{K}, \\ u\in\mathcal{H}. \end{array} \tag{3.3h}$$

The models TI-make and TI-make-task are equivalent but it has been noted that use of TI-make-task leads to a slightly longer computation time, which was unexpected. A possible explanation to this could be related to the use of the terminal tasks set that keeps track of which operations correspond to which job. The solver may find it harder to recognize useful patterns to parallelize computation when using this set instead of the sets of operations and jobs.

3.2.4 Alternative precedence constraints

To obtain the model of the LP relaxation of TI-make-task, the integrality constraints (3.3h) are simply replaced with $0 \le x_{tku} \le 1$. The number of precedence constraints (3.3e) is in the order of (tasks) × (time steps) and can thus become very large with increasing problem size. If a problem instance causes the model TI-make-task to become too large, to the degree that the memory requirements or time to solve the LP relaxation is too long for practical purposes, then the following alternative set of precedence constraints will be used:

$$\sum_{k\in\mathcal{M}_t}\sum_{\mu\in\mathcal{H}_t}(\mu+p_{tk})x_{tk\mu} \le \sum_{l\in\mathcal{M}_{t+1}}\sum_{\nu\in\mathcal{H}_{t+1}}\nu x_{t+1,l\nu}, \ t\in\mathcal{T}\setminus\widetilde{\mathcal{T}}. \tag{3.4}$$

Note that the number of constraints (3.4) are dependent only on the number of tasks. Provided that the integrality constraints (3.3h) are fulfilled, the constraints (3.4) are equivalent to (3.3e). The use of these alternative constraints, however, do not yield as tight of a LP relaxation bound. The model (3.3a)–(3.3d), (3.4), (3.3f)–(3.3h), that makes use of the alternative precedence constraints, will henceforth be referred to as *TI-make-task-prec*.

3.2.5 Tardiness objective

Modifying TI-make-task to make use of the objective (3.1) (i.e., minimizing the sum of completion times and total tardiness) is not difficult. Using the same sets, parameters and variables as in TI-make-task the completion times C_t of terminal tasks can be written as:

$$C_t = \sum_{k \in \mathcal{M}_t} \sum_{u \in \mathcal{H}} (u + p_{tk}) x_{tku}, \ t \in \widetilde{\mathcal{T}}.$$

Let d_t, for $t \in \widetilde{\mathcal{T}}$, denote the due dates (i.e., desired completion times) of each corresponding job. The objective (3.1) of minimizing the sum of completion times and total tardiness can thus be expressed as to

$$\text{minimize} \sum_{t \in \widetilde{\mathcal{T}}} \sum_{k \in \mathcal{M}_t} \sum_{u \in \mathcal{H}} \left((u + p_{tk}) + \beta_t (u + p_{tk} - d_t)_+ \right) x_{tku}. \tag{3.5}$$

Note that the $()_+$ operator in this objective applies only to parameters, hence the objective function remains linear. To alter the model TI-make-task to consider the objective (3.5) instead of makespan, the only change needed is to remove the constraints (3.3f) that define the makespan, and to replace the objective function (3.3a) with (3.5). The model (3.5), (3.3b)–(3.3e), (3.3g)–(3.3h), utilizing the tardiness objective, will henceforth be referred to as *TI-tard-task*. The model making use of the tardiness objective and alternative precedence constraints, i.e. (3.5), (3.3b)–(3.3d), (3.4), (3.3g)–(3.3h), will be referred to as *TI-tard-task-prec*.

Due dates

The Fattahi benchmark test instances that are used as input data in this thesis (more on this in section 4.1) do not include due dates. Thus, to utilize the tardiness objective in our study, we will be using the same due dates and tardiness weights as in [23] (where the model used in this thesis is solved using the aforementioned iterative solution procedure). These due dates were generated to reflect experience from a real flexible job shop and the extension of the Fattahi instances to include these due dates are soon to be made publicly available. The tardiness weights are defined as a non-increasing function of the due dates of the respective job (or terminal task), as is described below.[1]

[1]For the due dates and tardiness weights used in this thesis contact Karin Thörnblad at karin.thornblad@gknaerospace.com

Tardiness weights

In real flexible job shops, jobs will sometimes already be tardy when first available to be processed (i.e., jobs can sometimes have due dates before their respective release dates). To represent this in models like the one used in this thesis, jobs may sometimes possess negative due dates and will thus always have positive tardiness. To create a distinction in the objective function between jobs that are late and *very* late, and prioritize jobs that are most delayed, the tardiness weights (β_t) used in [23] are defined as follows:

$$\beta_t := B\left(1 - \frac{d_t}{\max_{t\in\widetilde{\mathcal{T}}}\{|d_t|\}}\right)_+, \ t \in \widetilde{\mathcal{T}}.$$

Using the definition for β_t above, the weight $B > 0$ is utilized for all jobs such that $0 \leq \beta_t \leq 2B$ hold for all $t \in \widetilde{\mathcal{T}}$. In this way the jobs with earlier due dates will recieve higher tardiness weights than those with later due dates. In this thesis, the due dates that are used are all positive (i.e., $d_t \geq 0 \ \forall \ t \in \widetilde{\mathcal{T}}$). Note that, using the same definition for β_t with only positive due dates, we will instead generate tardiness weights that satisfy $0 \leq \beta_t \leq B$.

3.3 New valid inequalities

The class of VIs used in our cutting-plane algorithm are an extension of a class of VIs that were originally derived by Sousa and Wolsey (1992) (see [21]) for a time-indexed model of the SMSP. This class of VIs are known to be facet-inducing for this model of the SMSP and have also previously been extended to a time-indexed model of a PMSP in [2], resulting in tighter LP relaxation bounds. As a reminder to the reader, the SMSP can be stated as the problem of scheduling n jobs $j \in \mathcal{J}$ on a single machine within a given time horizon $[0,H_{\max}]$. For each job $j \in \mathcal{J}$, and time interval $u \in \mathcal{H}$ the model of the SMSP makes use of the following binary decision variables:

$$x_{ju} = \begin{cases} 1, & \text{if job } j \text{ starts in time period } u \\ 0, & \text{otherwise.} \end{cases}$$

Using the same notation for sets and parameters as before, the time-indexed formulation of the SMSP in [21] can be stated as follows:

$$\text{minimize} \qquad \sum_{j\in\mathcal{J}}\sum_{u\in\mathcal{H}} c_{ju}x_{ju} \tag{3.6a}$$

$$\text{subject to} \qquad \sum_{u\in\mathcal{H}} x_{ju} = 1, \qquad j \in \mathcal{J}, \tag{3.6b}$$

$$\sum_{j\in\mathcal{J}}\sum_{s=(u-p_j+1)_+}^{u} x_{js} \leq 1, \qquad u \in \mathcal{H}, \tag{3.6c}$$

$$x_{ju} \in \{0,1\}, \ j \in \mathcal{J}, \ u \in \mathcal{H}. \tag{3.6d}$$

In this formulation the objective (3.6a) is the minimization of the costs c_{ju} associated with scheduling job j at time u. Just as in TI-make-task/TI-tard-task, the constraints (3.6b) ensure that each job is scheduled exactly once, and the capacity constraints (3.6c) make sure that only one job at a time can be processed on the single resource. The integrality constraints (3.6d), as before, enforce that the decision variables x_{ju} can only attain values of 0 and 1.

The family of VIs to the SMSP formulation above can be stated as follows. Consider a job $j \in \mathcal{J}$, time period $u \in \mathcal{H}$ and $\Delta \in \{1, \ldots, \overline{p}_j - 1\}$ where $\overline{p}_j = \max_{l \neq j}\{p_l\}$ denotes the largest processing time among jobs $l \neq j$. Then

$$\sum_{s=u-p_j}^{u+\Delta-1} x_{js} + \sum_{l \neq j} \sum_{\nu=u-p_l+\Delta}^{u-1} x_{l\nu} \leq 1 \tag{3.7}$$

is a VI for the feasible region of (3.6).

3.3.1 Extension of VIs to the FJSP

The FJSP is a multi-operation model, meaning that each job consists of a set of operations that are to be completed in order to complete the job. Furthermore, associated with each operation is a set of resources on which they are allowed to be processed. To extend the family of VIs (3.7) to the models TI-make-task and TI-tard-task, we will need to introduce the following notation:

$$\begin{aligned} \mathcal{T}_k \quad & \text{set of tasks allowed to be processed on resource } k, \ (\mathcal{T}_k \subseteq \mathcal{T}), \\ \overline{p}_{tk} = & \max_{s \in \mathcal{T}_k \setminus \{t\}} \{p_{sk}\}, \text{ largest processing time among tasks } T_k \setminus \{t\} \\ & \text{competing with task } t \text{ for resource } k. \end{aligned}$$

Using this notation, the extension of the family of VIs (3.7) to TI-make-task and TI-tard-task can be expressed as:

$$\sum_{s=u-p_{tk}}^{u+\Delta-1} x_{tks} + \sum_{l \in \mathcal{T}_k \setminus \{t\}} \sum_{\nu=u-p_{lk}+\Delta}^{u-1} x_{lk\nu} \leq 1, \qquad \begin{array}{l} t \in \mathcal{T}, \ u \in \mathcal{H}, \\ k \in \mathcal{M}_t, \ \Delta \in \{1, \ldots, \overline{p}_{tk} - 1\}. \end{array} \tag{3.8}$$

Note that the number of VIs in this family is in the order of (tasks) $\times$ (time steps) $\times$ (resources) $\times$ ($\overline{p}_{tk}$) and hence grows extremely large with problem size. It is for this reason a cutting-plane method was considered suitable for testing their strength.

3.3.2 Proof of validity

The family of inequalities (3.8) can be constructed from constraints in (3.3) using integer rounding: Consider one constraint (3.3b) for some chosen task $t \in \mathcal{T}$. Furthermore, consider two of the constraints (3.3d), one for period $u - 1$ and arbitrary machine $m \in \mathcal{M}_t$,

and one for period $u+\Delta-1$ (where $\Delta \in \{1,\dots,\overline{p}_{tk}-1\}$) and the same machine m. By summing these three constraints with coefficients $\frac{1}{2}$ we obtain a new constraint that is valid for the LP relaxation of the FJSP. By first rounding down the LHS (in effect "loosening" the new constraint) and then rounding down the RHS (by reimposing integrality constraints) we obtain our new VIs (3.8) that are valid for the original FJSP. To clarify, this procedure is demonstrated below.

One constraint (3.3b) for arbitrary task t:

$$\sum_{k\in\mathcal{M}_t}\sum_{u\in\mathcal{H}} x_{tku} = 1, \text{ for some } t \in \mathcal{T}.$$

Two constraints (3.3d):

$$\sum_{t\in\mathcal{T}_m}\sum_{\mu=u-p_{tm}}^{u-1} x_{tm\mu} \le 1, \text{ for some } m \in \mathcal{M}_t, \text{ and period } u-1 \in \mathcal{H},$$

$$\sum_{t\in\mathcal{T}_m}\sum_{\mu=u-p_{tm}+\Delta}^{u+\Delta-1} x_{tm\mu} \le 1, \text{ for the same machine } m, \text{ and period } u+\Delta-1 \in \mathcal{H}.$$

Adding these together with coefficients $\frac{1}{2}$ yields:
(Note that $u-p_{tk} < u-p_{tk}+\Delta$ and $u-1 < u+\Delta-1, \quad \forall\, \Delta \in \{1,\dots,\overline{p}_{tk}-1\}$.)

$$\underbrace{\sum_{w\in\mathcal{T}_m}\left(\frac{1}{2}\sum_{\mu=u-p_{wm}}^{u-1} x_{wm\mu} + \frac{1}{2}\sum_{\mu=u-p_{wm}+\Delta}^{u+\Delta-1} x_{wm\mu}\right)}_{(3.3d)} + \underbrace{\frac{1}{2}\sum_{k\in\mathcal{M}_t}\sum_{s\in\mathcal{H}} x_{tks}}_{(3.3b)}$$

$$= \sum_{w\in\mathcal{T}_m}\left(\sum_{\mu=u-p_{wm}+\Delta}^{u-1} x_{wm\mu} + \frac{1}{2}\sum_{\mu=u-p_{wm}}^{u-p_{wm}+\Delta-1} x_{wm\mu} + \frac{1}{2}\sum_{\mu=u}^{u+\Delta-1} x_{wm\mu}\right) + \frac{1}{2}\sum_{k\in\mathcal{M}_t}\sum_{s\in\mathcal{H}} x_{tks}$$

$$= \sum_{s=u-p_{tm}}^{u+\Delta-1} x_{tms} + \frac{1}{2}\left(\sum_{s=0}^{u-p_{tm}-1} x_{tms} + \sum_{s=u+\Delta}^{H_{\max}} x_{tms}\right) + \frac{1}{2}\sum_{\mu=u-p_{tm}+\Delta}^{u-1} x_{tm\mu}$$

$$+ \sum_{w\in\mathcal{T}_m\setminus\{t\}}\left(\sum_{\mu=u-p_{wm}+\Delta}^{u-1} x_{wm\mu} + \frac{1}{2}\sum_{\mu=u-p_{wm}}^{u-p_{wm}+\Delta-1} x_{wm\mu} + \frac{1}{2}\sum_{\mu=u}^{u+\Delta-1} x_{wm\mu}\right)$$

$$+ \frac{1}{2}\sum_{k\in\mathcal{M}_t\setminus\{m\}}\sum_{s\in\mathcal{H}} x_{tks} \le 1 + \frac{1}{2}.$$

Rounding down the LHS yields

$$\sum_{s=u-p_{tm}}^{u+\Delta-1} x_{tms} + \sum_{w\in\mathcal{T}_m\setminus\{t\}}\sum_{\mu=u-p_{wm}+\Delta}^{u-1} x_{wm\mu} \le 1 + \frac{1}{2}.$$

Reimposing integrality constraints implies that the LHS is integral; hence the RHS can be rounded down to the nearest integer. We obtain the valid inequality

$$\sum_{s=u-p_{tm}}^{u+\Delta-1} x_{tms} + \sum_{w\in\mathcal{T}_m\setminus\{t\}} \sum_{\mu=u-p_{wm}+\Delta}^{u-1} x_{wm\mu} \leq 1.$$

which is, in fact, the inequality (3.8) for arbitrary $t \in \mathcal{T}$, $m \in \mathcal{M}_t$, $u \in \mathcal{H}$, and $\Delta \in \{1, \ldots, \overline{p}_{tk} - 1\}$. □

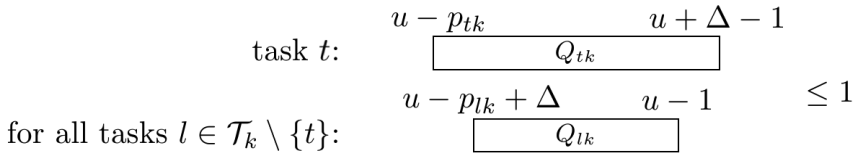
3.3.3 Intuitive description of new VIs

The validity of inequalities (3.8) can also be understood more intuitively by observing the time intervals in each inequality that are being used. Denote these time intervals Q_{tk} and Q_{lk}, where

$$Q_{tk} = [u - p_{tk}, u + \Delta - 1], \qquad \text{for task } t \in \mathcal{T} \text{ and } k \in \mathcal{M}_t,$$

$$Q_{lk} = \begin{cases} [u - p_{lk} + \Delta, u - 1], & \text{if } p_{lk} > \Delta \\ \emptyset, & \text{otherwise} \end{cases} \qquad \text{for all tasks } l \in \mathcal{T}_k \setminus \{t\} \text{ and same } k.$$

To aid in the description of the inequalities we will use the following diagram as a visual aid, similar to those used in [26] and [2]. For each individual machine k we have a diagram that contains one line for each task. The blocks on each line associated with a task t indicate the time periods u for which x_{tku} occurs in the inequality. The inequalities (3.8) are thus represented by the following diagram, for some machine k:



Let us first take note of the earliest possible completion times of tasks scheduled to begin within these intervals. A task t that is scheduled to begin at some point within time interval Q_{tk} will, at the earliest, be completed at time u (since the earliest starting time within this interval is $u - p_{tk}$). Likewise, any task $l \in \mathcal{T}_k \setminus \{t\}$ that is scheduled to begin at some point within time interval Q_{lk} will, at the earliest, be completed at time $u + \Delta$ (since the earliest starting time within this interval is $u - p_{lk} + \Delta$).

Let some task $l' \in \mathcal{T}_k \setminus \{t\}$ be scheduled to begin within time interval $Q_{l'k}$. Task l' is, by construction of $Q_{l'k}$, guaranteed to occupy machine k at time u and time $u - 1$ (since task l' completes at the earliest at time $u + \Delta$ and is scheduled to begin at $u - 1$ at the latest). This implies that no other task $l \in \mathcal{T}_k \setminus \{t\}$ can be scheduled within its respective time interval Q_{lk} (since these are also guaranteed to occupy machine k at time u).

Furthermore, this also implies that task t cannot begin at any point within Q_{tk} *before* or *after* the starting time of l'. Task t cannot begin *before* since (if it is scheduled to begin within Q_{tk}) it will be completed, at the earliest, at time u (and again l' is guaranteed to occupy machine k at time $u-1$). Task t cannot be scheduled *after* l' has completed either, since within Q_{tk} it can at the latest begin at $u+\Delta-1$, which is one time step before the earliest completion time of l'.

Now instead let task t be scheduled to begin at some point within time interval Q_{tk}. This implies that no task $l \in \mathcal{T}_k \setminus \{t\}$ can be scheduled to begin within Q_{lk} *before* the starting time of t, since t will begin at time $u+\Delta-1$ at the latest, and the earliest completion time for $l \in \mathcal{T}_k \setminus \{t\}$ scheduled to begin in Q_{lk} is $u+\Delta$. Also, no $l \in \mathcal{T}_k \setminus \{t\}$ can be scheduled to begin, at some point within Q_{lk}, *after* the starting time of t either, since it can begin at the latest at time $u-1 < u$ (the earliest completion time of t). This implies the validity of (3.8).

4

Implementation and results

In this chapter we will describe our cutting-plane implementation, followed by computational results. We will start by describing the test instances that have been used as input data in section 4.1. In section 4.2 we will present our implementation of a cutting-plane algorithm. Finally, tables and graphs containing computational results will be presented in section 4.3.

4.1 Test instances

Among data instances used in various approaches to the FJSP, the ones introduced by Fattahi et al. (2007) (see [8]) are one of the most frequently used for computational experimentation. The comparatively small size of these instances relative to other sets of standard benchmark instances also make them a preferable choice to test mathematical programming approaches to solving the FJSP (see [1] for a survey of common benchmark instances for the FJSP). The Fattahi instances are divided into ten small-sized test instances labeled "*sfjs1–sfjs10*", and ten medium-sized instances labeled "*mfjs1–mfjs10*". In this thesis the twelve largest Fattahi instances (i.e., *sfjs9–mfjs10*) have been chosen as input data for our model. The eight smallest instances (*sfjs1–sfjs8*) are solved in a matter of a few seconds, and are therefore not considered interesting for comparison.

Included in the Fattahi instances are: the number of resources m, jobs n and operations n_j for each job j, along with corresponding processing times. All processing times are integral. The precedence relations between operations along with their degree of flexibility (i.e., which resources can process them) are also included. As previously mentioned, when utilizing the tardiness objective we will be using due dates and tardiness weights from [23]. Release dates, and tardiness weights are calculated as described in section 3.2.1 and 3.2.5 respectively. The size of the chosen instances range from $n = 3, m = 3$, and $n_j \leq 3$ for the smallest instance *sfjs9* to $n = 12, m = 8$, and $n_j \leq 4$ for the largest

instance *mfjs10*.

In order for our MILP formulation of the FJSP to be feasible, the size of the planning horizon ($H_{\max}$) must of course be large enough to contain at least one optimal schedule. However, as we remember from section 3.2, one of the main weaknesses of using a time-indexed formulation is that they tend to grow quickly with the size of the problem. It is therefore also preferable to try and keep $H_{\max}$ as small as possible, to reduce memory requirements and computation time. Initially, to ensure that the planning horizon was long enough to contain an optimal schedule, the size of $H_{\max}$ was calculated simply by summing together all of the longest possible processing times of each operation i of job j, i.e., $H_{\max} = \sum_{j \in \mathcal{J}} \sum_{i \in \mathcal{N}_j} \max_{k \in \mathcal{M}_{ij}}(p_{ijk})$. This is essentially the makespan of the longest possible schedule. The use of this $H_{\max}$, however, induces a very large model, causing the time to solve the LP relaxation of even the smallest test instance (*sfjs9*) to be too long for practical purposes. In addition to this, the size of the family of new VIs also increases with the size of the planning horizon. To reduce memory requirements, we have instead used the final planning horizon resulting from the last iteration of the aforementioned iterative solution procedure (described in [23, 24]), when applied to the model utilizing the tardiness objective. This has been done to enable us to add as many of the new VIs as possible to the base model, in order to best test their collective potential effect. The sizes of the planning horizon that were used, along with the sizes of each test instance, are displayed in table 4.1 below.

Problem instance (maxops, jobs, resources)	$H_{\max}$
sfjs9 (3,3,3)	992
sfjs10 (3,4,5)	843
mfjs1 (3,5,6)	763
mfjs2 (3,5,7)	677
mfjs3 (3,6,7)	721
mfjs4 (3,7,7)	877
mfjs5 (3,7,7)	902
mfjs6 (3,8,7)	1026
mfjs7 (4,8,7)	1332
mfjs8 (4,9,8)	1243
mfjs9 (4,11,8)	1503
mfjs10 (4,12,8)	2001

Table 4.1: Sizes of test instances and planning horizon

4.2 Implementation

The basic outline of the cutting-plane implementation used in this thesis is illustrated in figure 4.1 below.

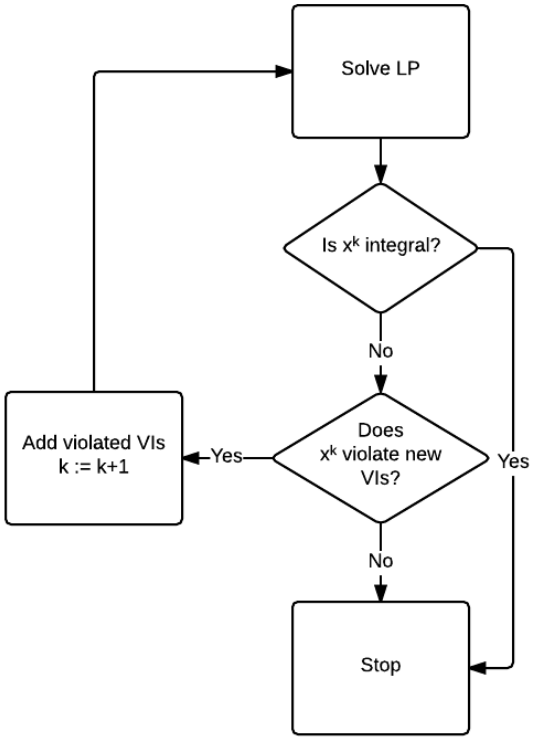


Figure 4.1: The cutting-plane algorithm

To obtain the LP solution in each iteration of the cutting-plane algorithm we have used IBM's mathematical programming solver CPLEX. During the separation stage of the cutting-plane algorithm, the LP solution of the current iteration k, denoted x^k, is checked to see whether or not it violates any of the new family of VIs (3.8). This is basically done by enumerating all of the VIs and then checking them one by one in sequential order. Any violated VIs are then added to the model in the next iteration. Ideally we would like to add as many violated VIs to the model as we can in each iteration, to best assess the new family of VIs collective potential to close the initial optimality gap. Unfortunately, the number of VIs in (3.8) exceeds 500000 even for the smallest of our test instances, and attempting to add too many at once only results in memory failure when solving. To deal with this problem we have chosen an upper limit of 8000 total new VIs to be added to the model. This limit was chosen simply through trial and error, by running the cutting-plane algorithm for different choices of upper limit and different problem sizes. Increasing the upper limit further would lead to memory failure for larger test instances (*mfjs7–10*).

The cutting-plane algorithm is also terminated if a pre-specified time limit is exceeded. As mentioned in section 1.1, part of what inspired the subject of this thesis is the potential application of the new VIs to improve the performance of the so-called iterative solution procedure developed in [23]. For each iteration of the iterative solution procedure a time limit of 7200 CPU seconds was set for each call to the CPLEX solver, and for the model using the tardiness objective the iterative solution procedure would take 4–7 iterations before terminating. With this in mind, and the potential application of running the cutting-plane algorithm parallell to the iterative solution procedure, a time limit for each call to the CPLEX solver is set to 40000 CPU seconds, by taking $\frac{4 \cdot 7200 + 7 \cdot 7200}{2} \approx 40000$. Note that since the time limit is for each call to the CPLEX solver the total computation time may still exceed 40000 CPU seconds. Furthermore, since computation times tend to be quite long, we have decided that the cutting-plane algorithm will run for a maximum of 5 completed iterations.

To summarize, the termination criteria for our implementation of the cutting-plane algorithm are:

- LP solution is feasible to unrelaxed problem, hence optimal
- time limit of 40000 CPU seconds reached (for each call to CPLEX)
- limit of 5 completed iterations reached
- limit of 8000 VIs added reached
- no violated VIs found

4.3 Computational Results

In this section we will present the results from running our cutting-plane algorithm on the models TI-make-task(-prec) and TI-tard-task(-prec) described in chapter 3, using the twelve largest Fattahi test instances as input data as mentioned in section 4.1. Our main priority is to evaluate the new VIs ability to bridge the gap between the initial LP relaxation bound z^0, and the best objective value found z^*. One measurement that will be used is the *percentage of initial optimality gap closed* (piogap), which is defined in this thesis as:

$$\text{piogap} = \frac{z(x^k) - z^0}{z^* - z^0} \cdot 100\%. \tag{4.1}$$

Here $z(x^k)$ denotes the objective value corresponding to LP solution x^k, after k iterations of the cutting-plane algorithm (e.g., $z^0 = z(x^0)$). To remind the reader, the *mipgap* is defined as the relative difference between the best LB found and the best objective value found. The mipgap is defined in this thesis as:

$$\text{mipgap} = \frac{z^* - z(x^k)}{z^*} \cdot 100\%. \tag{4.2}$$

The best objective values found, i.e., z^*, that are used in this thesis have been calculated via the iterative solution procedure described in [23, 24]. The values of z^* that were used will be displayed along with the results for each model. As has been mentioned earlier, the main strength of the TI-formulation is that it yields very strong LBs. Although it may be beneficial to tighten the bounds further, when the initial optimality gap is small to begin with the positive effects of tightening it further may not be worth the computational cost of adding the new cuts. With this in mind we present both the new LBs obtained for each iteration of the cutting-plane algorithm as well as the total time for solving and resolving the LP relaxation.

Computations were carried out using AMPL-CPLEX 12.1.0 (Fourer et al., 2002; IBM Corp., 2009) on a computer with four 2.90 GHz Intel Core i5-3470S processors, each with four cores, with a total memory of 16.75 Gbyte of RAM. All results presented in this section are those obtained from this computer. Since computation times were often very long, and since we would sometimes encounter memory issues during preliminary testing, the results were also verified using a second computer with additional memory. This second computer has two 2.66 GHz Intel Xeon X5650 processors, each with six cores, with a total of 48 Gbyte of RAM.

4.3.1 Diversifying cuts

Let $VI(t,k,u,\Delta)$ denote a single VI from the family (3.8) for some task $t \in \mathcal{T}$, resource $k \in \mathcal{M}_t$, time interval $u \in \mathcal{H}$, and $\Delta \in \{1, \ldots, \overline{p}_{tk} - 1\}$. During preliminary testing of the cutting-plane algorithm we discovered that many cuts of a similiar "type" were being added. By "type" we mean that clusters of VIs were being added for the same task, resource and time interval but for varying values of Δ (e.g., $VI(1,1,1,1 \rightarrow 100)$ or $VI(2,2,2,1 \rightarrow 80)$). This led us to believe the LP relaxation bound could perhaps be raised higher if we were to add a more diverse range of cuts, to avoid adding similar VIs in the sense that they cut off much of the same portions of the feasible region. To do this we experimented with checking only, for example, every third or tenth Δ of the VIs in the separation stage of the cutting-plane algorithm. We also experimented with skipping different time intervals u but saw greater effects for skipping Δ. Figure 4.2 displays the effect of skipping different sized chunks of Δ on the percentage of the initial optimality gap that is closed after the cutting-plane algorithm has terminated. The results from the smallest five instances *sfjs9–10* and *mfjs1–3* are omitted for TI-tard-task (Tardiness), since no difference in the percentage of the initial optimality gap closed was observed for these instances. For the same reasons results for *sfjs9*, *mfjs4*, *mfjs6–8* and *mfjs10* are omitted for TI-make-task (Makespan).

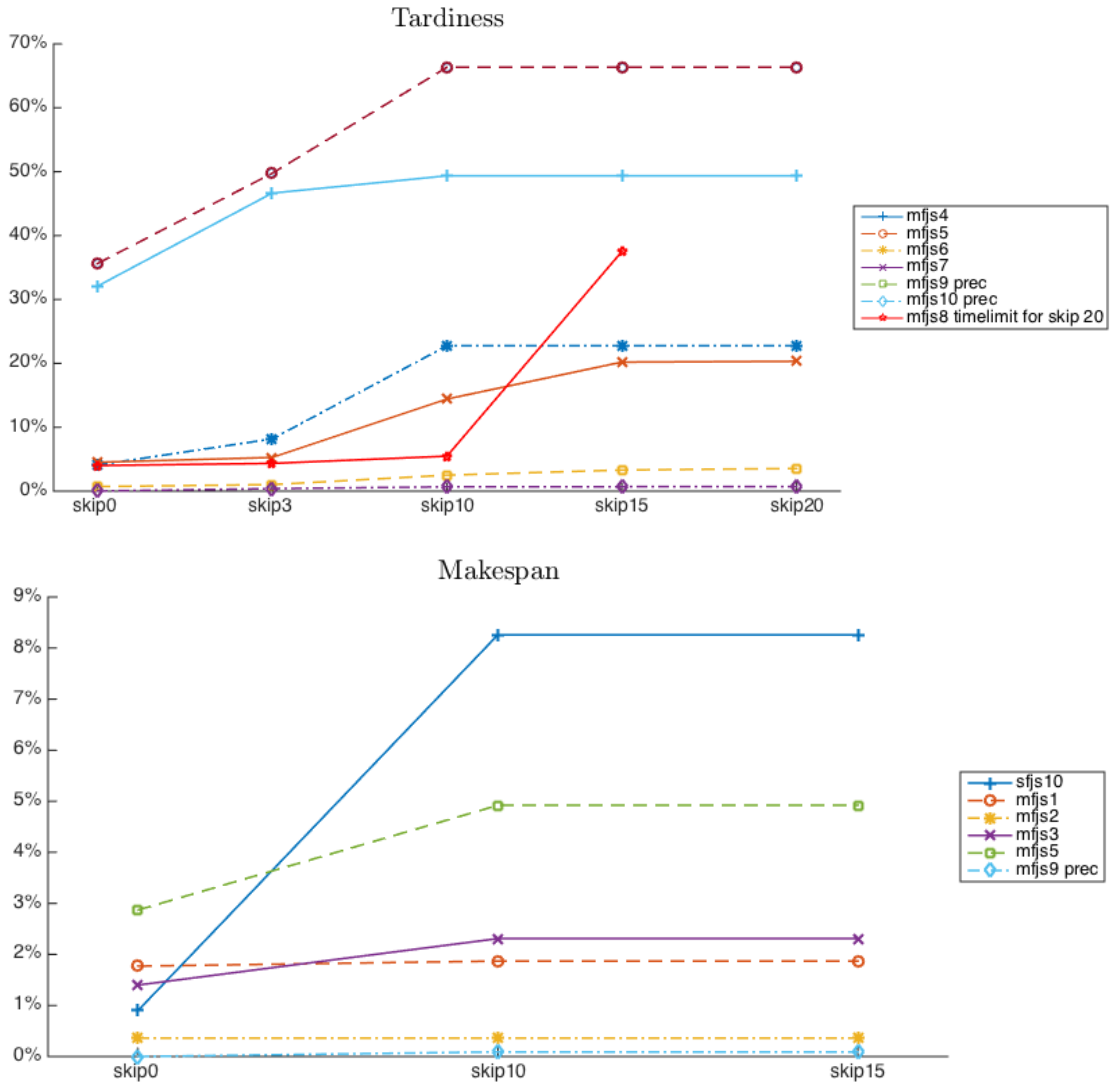


Figure 4.2: The piogap after termination of the cutting-plane algorithm while attempting to diversify the cuts added in each iteration. "skip 1–20" denotes the number of Δ that are skipped when checking for violated VIs in the separation stage of the cutting-plane algorithm.

As we can see, by skipping to check certain Δ, and consequently adding a more diverse range of VIs before hitting our upper limit on cuts added, the initial optimality gap could be closed even further than when checking every Δ (skip 0). Furthermore, a smaller number of cuts were added for higher values of skipped Δ, but still managed to achieve the same LB. For TI-make-task the best results where achieved for skip 10 and 15. For TI-tard-task we achieved the best results for skip 15 and skip 20 but ended up hitting our time limit for *mfjs8* with skip 20. Since using skip 15 yielded the best results

within our time limit for both objectives we will only be presenting the results for these below. Also worth noting from the two graphs in figure 4.2 is that the new VIs seem to have a significantly greater effect on bridging the initial optimality gap for TI-tard-task model as opposed to the TI-make-task model.

4.3.2 Results for the minimization of tardiness

A summary of the results from running our cutting-plane algorithm using skip15 (see figure 4.2) on the model TI-tard-task (TI-tard-task-prec for the two largest instances) is displayed in table 4.2. In this table, the best objective value (z^*) along with the initial LP relaxation bound (LB before) and the LB after the cutting-plane algorithm has terminated (LB after) are presented. The piogap and the mipgap are calculated according to the aforementioned formulas (4.1) and (4.2) respectively. For instances *sfjs9* and *mfjs2–3* the LP relaxation solution that was found was also optimal to the unrelaxed problem (marked with "int" for integer feasible). For the two largest instances *mfjs9–10* solving the initial LP relaxation of the model TI-tard-task went beyond our time limit. For these instances we have thus instead used the smaller model TI-tard-task-prec utilizing the alternative precedence constraints (3.4) (marked in the table with "prec" to signify this).

Tardiness results

Problem	z^*	LB before	LB after	mipgap before	mipgap after	piogap
sfjs9	554.329	554.329	-	int 0.00%	-	-
sfjs10	5736.04	5521.378	5547.495	3.74%	3.29%	12.17%
mfjs1	10 732.4	10 696.906	10 728.881	0.33%	0.03%	90.09%
mfjs2	13 915.752	13 915.752	-	int 0.00%	-	-
mfjs3	9671.92	9671.920	-	int 0.00%	-	-
mfjs4	19 300.8	19 145.468	19 222.144	0.80%	0.41%	49.36%
mfjs5	12 416.3	12 371.236	12 401.132	0.36%	0.12%	66.34%
mfjs6	19 585.9	19 179.748	19 272.132	2.07%	1.60%	22.75%
mfjs7	35 111.1	33 702.649	33 987.216	4.01%	3.20%	20.20%
mfjs8	25 225.3	24 794.569	24 956.039	1.71%	1.07%	37.49%
mfjs9 prec	60 159.5	54 326.835	54 520.069	9.70%	9.37%	3.31%
mfjs10 prec	45 568.4	42 152.024	42 176.057	7.50%	7.44%	0.70%

Table 4.2: Cutting-plane algorithm results for TI-tard-task(-prec).

We can see that adding the new VIs yield tighter LBs for all test instances in which violated VIs were found. The effect of these seem to diminish however for the two largest instances using the alternative precedence constraints. Note that, as previously mentioned, using the alternative precedence constraints results in a looser initial LP relaxation bound. The resulting initial mipgap is therefore relatively larger for these

instances. In retrospect we could have, in addition to the new VIs, checked if the current LP solution x^k also violated the original precedence constraints (3.3e) for the two largest instances. By doing this it may have been possible to raise the LB even further.

Time elapsed (tardiness)

The total time spent (in CPU seconds) on solving the initial LP relaxation (LP relax), resolving the LP problem each time violated VIs are appended to the model (resolve), checking for violated constraints (loop), and the total time spent from initialization to termination of the cutting-plane algorithm (total) are presented in table 4.3. In addition to the computation times, the number of completed iterations (#it) and reason for the cutting-plane algorithm's termination (termination) are also reported.

Tardiness times

Problem	LP relax	resolve	loop	total	#it	termination
sfjs9	1	0	0	1	0	optimal
sfjs10	1	1	16	17	1	no cuts found
mfjs1	4	1	32	37	1	no cuts found
mfjs2	1	0	18	19	0	optimal
mfjs3	3	0	36	39	0	optimal
mfjs4	12	183	176	371	2	no cuts found
mfjs5	20	773	260	1053	3	no cuts found
mfjs6	62	2071	348	2481	3	no cuts found
mfjs7	1076	18154	364	19594	2	max VIs
mfjs8	4431	24698	174	29303	1	max VIs
mfjs9 prec	89	1083	199	1371	1	max VIs
mfjs10 prec	425	3356	1369	5150	2	max VIs

Table 4.3: Computation times (in CPU seconds), number of completed iterations, and reason for termination for the cutting-plane algorithm for TI-tard-task(-prec). The termination critera are: current LP solution is optimal (optimal), time limit reached (time limit), maximum completed iterations reached (max #it), maximum number of added VIs reached (max VIs), and no violated VIs were found (no cuts found).

The majority of time spent was on solving and resolving the LP problem in each iteration. The time limit that was set and the maximum number of iterations were not reached for any of the test instances. Although not observable in the table, the addition of the VIs between iterations to the model would most often cause an increase in resolve time, but would sometimes also cause it to decrease. Furthermore, the increase in resolve time would most often occur already after the first iteration. This was also, in the majority of cases, the iteration in which the largest number of cuts were added to the model (more on this in the following subsection). The loop time for each iteration increases together with problem size, since the size of the family of VIs that are checked also grows with

problem size. Note that the loop for checking the VIs for violation was executed in AMPL which is not particularily efficient for this purpose. Implementing this step in a language that is better at handling this type of operation could greatly reduce the time requirement of this step.

Cuts added (tardiness)

In table 4.4 the number of VIs added in each iteration is presented. The sum of all VIs that have been added once the cutting-plane algorithm has terminated is also displayed. Cells in the table containing a "0" denote iterations in which no violated VIs were found. Cells that have been left empty mean that the cutting-plane algorithm terminated before checking for additional cuts in the corresponding iteration.

Tardiness cuts added

Problem	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Total
sfjs9	0				0
sfjs10	24	0			24
mfjs1	251	0			251
mfjs2	0				0
mfjs3	0				0
mfjs4	1524	487	0		2011
mfjs5	370	1946	142	0	2458
mfjs6	3789	248	219	0	4256
mfjs7	7175	825			8000
mfjs8	8000				8000
mfjs9 prec	8000				8000
mfjs10 prec	6243	1757			8000

Table 4.4: Number of VIs added in each iteration of the cutting-plane algorithm for the model TI-tard-task(-prec)

As can be observed in both table 4.3 and 4.4, the cutting-plane algorithm would most often either terminate due to the maximum number of added VIs being reached or because no violated VIs were found. The majority of cuts found for almost all instances were also found within the first iteration of the cutting-plane alorithm. For larger problem instances (*mfjs7–10*) the upper limit on number of cuts added was reached. It may be possible to bridge the initial optimality gap even further for these instances by adding more cuts. As previously mentioned, however, this will often lead to memory problems or induce a computation time that is beyond our time limit. Since the size of the family of new VIs increases with problem size, it may also be possible to obtain higher LBs, with the same upper limit for cuts added, by skipping even larger chunks of Δ. By doing this we could potentially reach VIs "later on" in our enumeration before hitting the upper limit of violated VIs added, and thus potentially find a more diverse group of

violated VIs to add to the model. With this in mind, it may have been a good idea to implement a "skip Δ" that is dependent on the problem size instead of keeping it static for all instances.

4.3.3 Results for the minimization of makespan

In this section we present the results of running our cutting-plane algorithm using skip15 on the model TI-make-task(-prec). These are displayed in table 4.5 in the same form as in the previous subsection for tardiness. For the smallest instance *sfjs9* the solution found was optimal for the unrelaxed model and is therefore marked with "int" (for integer feasible) to represent this. Computation time for solving the LP relaxation of TI-make-task for the two largest instances, *mfjs9–10*, went beyond our time limit. For these instances we will thus instead be presenting the results of the model TI-make-task-prec utilizing alternative precedence constraints. Once more, these instances are marked with "prec" to signify this.

Makespan results

Problem	z^*	LB before	LB after	mipgap before	mipgap after	piogap
sfjs9	210	210	-	int 0.00%	-	-
sfjs10	516	456.955	461.832	11.44%	10.50%	8.26%
mfjs1	468	410.852	411.922	12.21%	11.98%	1.87%
mfjs2	446	402.928	403.082	9.66%	9.62%	0.36%
mfjs3	466	416.616	417.757	10.60%	10.35%	2.31%
mfjs4	554	496	496	10.47%	10.47%	0.00%
mfjs5	514	453.241	456.228	11.82%	11.24%	4.92%
mfjs6	634	614	614	3.15%	3.15%	0.00%
mfjs7	879	764	764	13.08%	13.08%	0.00%
mfjs8	884	764	764	13.57%	13.57%	0.00%
mfjs9 prec	1081	801.747	801.994	25.83%	25.81%	0.09%
mfjs10 prec	1208	944	944	21.85%	21.85%	0.00%

Table 4.5: Cutting-plane algorithm results for TI-make-task(-prec).

From table 4.5 it seems that the new VIs generally have less of an effect on closing the initial optimality gap for the model utilizing the makespan objective instead of tardiness. Furthermore, the cuts do not seem to consistently improve the LBs; bridging a small portion of the mipgap for some instances and for others having no effect on the LB whatsoever. For instances *mfjs4*, *mfjs6–8* and *mfjs10* the additional VIs have no effect on the LB. Note that for all of these instances the initial LP relaxation bound was integer valued.

Time elapsed (makespan)

In table 4.6 the computation times for the model TI-make-task(-prec) are presented in the same form as before for TI-tard-task(-prec). Just as before, the number of completed iterations and reason for termination are also presented.

Makespan times

Problem	LP relax	solve	loop	total	#it	termination
sfjs9	10	0	1	11	0	optimal
sfjs10	2	325	59	386	4	no cuts found
mfjs1	60	861	80	1001	4	no cuts found
mfjs2	27	190	42	260	1	no cuts found
mfjs3	93	1606	122	1821	2	no cuts found
mfjs4	548	5569	307	6424	5	max #it
mfjs5	342	5886	158	6386	2	no cuts found
mfjs6	1456	33635	591	35682	5	max #it
mfjs7	3896	59560	632	64088	2	time limit
mfjs8	12482	26802	506	39790	1	time limit
mfjs9 prec	265	4779	242	5287	1	max VIs
mfjs10 prec	430	9484	1047	10962	2	max VIs

Table 4.6: Computation times (in CPU seconds), number of completed iterations, and reason for termination for the cutting-plane algorithm for TI-tard-task(-prec). The termination critera are: current LP solution is optimal (optimal), time limit reached (time limit), maximum completed iterations reached (max #it), maximum number of added VIs reached (max VIs), and no violated VIs were found (no cuts found).

The model TI-make-task utilizing the makespan objective is generally more difficult to solve than the model TI-tard-task that makes use of the tardiness objective. By comparing the tables 4.6 and 4.3 (computation times for TI-tard-task(-prec)), we can also see that the computation times are larger for TI-make-task(-prec) than TI-tard-task(-prec) for all instances. Computation time for instances *mfjs7–8* went beyond our time limit. However, by observing the previous table 4.5 we can see that the added VIs from previous completed iterations for instances *mfjs7–8* had no effect on the LB. Furthermore, the initial LP relaxation bounds for these instances are also integer valued. If the instances *mfsj7–8* were to follow the trend of other instances with integer valued initial LB (i.e., *mfjs4*, *mfjs6* and *mfjs10*), then allowing the cutting-plane algorithm to go beyond the time limit would still have no effect on raising the LB. Just as before for the tardiness objective, the addition of the VIs to the model would most often induce an increase in time to resolve the LP relaxation. Note again that, although not observable in the table, this increase in resolve time would most often occur already after the first iteration when the largest number of cuts were added.

Cuts added (makespan)

The number of added VIs in each iteration of the cutting-plane algorithm are presented in table 4.7 for the model TI-make-task(-prec), in the same form as before for TI-tard-task(-prec).

Makespan cuts added

Problem	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5	Total
sfjs9	0					0
sfjs10	1631	894	375	6	0	2906
mfjs1	1266	306	17	3	0	1592
mfjs2	328	0				328
mfjs3	1971	205	0			2176
mfjs4	973	95	2	148	86	1304
mfjs5	2143	493	0			2636
mfjs6	2409	84	471	19	16	2999
mfjs7	1306	634	167			2107
mfjs8	1727	384				2111
mfjs9 prec	8000					8000
mfjs10 prec	6849	1151				8000

Table 4.7: Number of VIs added in each iteration of the cutting-plane algorithm for the model TI-make-task(-prec)

As seen in table 4.7 above, the cutting-plane algorithm would most often terminate because no additional violated VIs were found. Similar to before, the majority of cuts found were within the first iteration of the cutting-plane algorithm. The upper limit on new VIs added was reached only for the two largest instances for the model using alternative precedence constraints.

5

Conclusions

In this final chapter we will review and summarize the results of this thesis and draw general conclusions from these. Furthermore, we will discuss our study in retrospect along with ideas for potential future research.

5.1 Conclusions

In this thesis we have shown that a family of VIs originally formulated for a time-indexed MILP model of the SMSP can be extended to a time-indexed MILP model of the FJSP. These VIs have previously been shown to be facet-inducing for the SMSP and strong for the PMSP. To evaluate the strength of the VIs for the FJSP we have implemented a cutting-plane algorithm and observed the VIs ability to bridge the initial optimality gap for the ten largest Fattahi test instances. This has been done for two different models: the model TI-tard-task utilizing a tardiness objective, and the model TI-make-task utilizing a makespan objective.

The importance of the choice of objective function is reflected in the varying effect of the VIs on the LB, as well as computation time, of the two models tested. The model TI-make-task is generally more difficult to solve than the model TI-tard-task, which is indicated by larger computation times for all test instances. The overall effect of the VIs on bridging the initial optimality gap for the Fattahi instances is also in general much greater and more reliable for the objective of minimizing tardiness. Note, however, that the initial optimality gap for tardiness is relatively smaller than for makespan to begin with. For the model TI-tard-task the VIs improve the LB for all instances in which violated VIs were found. Unfortunately, the relative effects seem to diminish with increased problem size and when utilizing the alternative precedence constraints. The added VIs will, in the majority of cases, also induce an increase in the time required to solve the LP relaxation. For the model TI-make-task the VIs do not reliably improve the LB for the

instances tested, sometimes improving the LB but in other cases having no effect on the LB at all. Adding the VIs to TI-make-task would also significantly increase computation time for all instances, causing the time to solve larger instances to go beyond our time limit. For larger instances (i.e., *mfjs7–10*) of the model TI-tard-task(-prec), and the two largest instances for TI-make-task-prec, the upper limit on VIs added was reached. A large portion of the family of VIs would therefore go unchecked, since adding additional VIs would cause memory problems. In section 4.3.1 we experimented with diversifying the VIs added by changing the order in which the VIs were checked for violation in the separation stage of the cutting-plane algorithm. The result of this experimentation was that the LB could be raised higher than otherwise, while adding fewer VIs to the model. It is possible that by diversifying them further, or according to a different procedure, the LB for these larger instances could be raised even higher.

In its current state, the cutting-plane algorithm does not appear to be useful for improving the performance of the previously mentioned iterative solution procedure developed in [23] (also described in [24]). The addition of the VIs to the model using the objective of minimizing tardiness do seem to have potential to raise the LB to a degree that would be helpful. However, to improve the iterative solution procedure the improved LB must be made available at an earlier iteration of this procedure. Currently the cutting-plane algorithm does not operate quickly enough to achieve this. Furthermore, the VIs ability to improve the LB for larger problem instances (for which we are also most interested in improving the iterative solution procedure) is still too small to justify the increase in computation time. This is true in particular when utilizing the objective of minimizing makespan where the effect on the LB is either small or non-existant, yet still induces a heavy increase in computation time.

5.2 Discussion and future research

During the course of this study we have had several ideas for improving the performance of the cutting-plane algorithm that we did not have the time to implement, many of which have already been mentioned. It has been noted that the separation step, in which the current LP solution is checked to see if it violates any of the family of VIs, requires a longer computation time than expected. Given more time we would have liked to implement this step in another programming language (for instance C) that can more easily recognize opportunities to parallelize computation and thus potentially perform this step much faster. In addition to this, we feel that the LB could be raised even higher (in particular for larger instances where the maximum number of added VIs was reached) by diversifying the VIs added further than what has been done. A condition that is dependent on problem size, skipping larger chunks of Δ for larger problems, may have been useful in achieving this. It would have also been interesting to experiment more with changing the order in which the tasks, resources and time periods are checked as well. Another idea to diversify the VIs added is to apply a condition that checks whether or not cuts to be added are sufficiently orthogonal to eachother in the sense

that their scalar product is close to zero.

For the two largest instances *mfjs9–10* we were unable to solve the LP relaxation of TI-tard-task and TI-make-task within our time limit and were therefore forced to resort to the two smaller models TI-tard-task-prec and TI-make-task-prec using alternative precedence constraints for these instances. In [23], it has already been established that the models using these alternative constraints do not have as strong lower bounds as those using the original precedence constraints. In retrospect we could have, in addition to the cutting-plane algorithm using the new VIs, used the original precedence constraints in its own cutting-plane algorithm. The results could then be compared to further evaluate the strength of the new VIs. Another possibility would be to experiment with adding the VIs in combination with other types of relaxations, to better assess the VIs ability to achieve higher LBs for these larger, more computationally demanding instances.

As previously mentioned in section 1.1, the family of VIs that were extended to the FJSP in this thesis have also previously been extended to a time-indexed MILP model of the PMSP in the Ph.D. thesis [2]. In [2], two additional families of VIs were developed (one of which was based on the family of VIs that we have extended to the FJSP) and tested for the same model of the PMSP with positive results. The extension of these other two families of VIs to the FJSP may also be another interesting topic for future research.

Summary of notation

Sets

$\mathcal{J}$	set of jobs; $j \in \mathcal{J} := \{1, \ldots, n\}$
$\mathcal{N}_j$	set of operations of job j; $i \in \mathcal{N}_j := \{1, \ldots, n_j\}$
$\mathcal{K}$	set of resources; $k \in \mathcal{K} := \{1, \ldots, m\}$
$\mathcal{T}$	set of tasks; $t \in \mathcal{T} := \{1, \ldots, \sum_{j \in \mathcal{J}} n_j\}$
$\widetilde{\mathcal{T}}$	set of terminal tasks; $t \in \widetilde{\mathcal{T}} := \{n_1, (n_1 + n_2), \ldots, (\sum_{j \in \mathcal{J}} n_j)\}$ (i.e., tasks corresponding to the final operation of each job)
$\mathcal{M}_t$	set of resources that are allowed to process task t, $(\mathcal{M}_t \subseteq \mathcal{K})$
$\mathcal{T}_k$	set of tasks allowed to be processed on resource k, $(\mathcal{T}_k \subseteq \mathcal{T})$
$\mathcal{H}$	set of time periods; $u \in \mathcal{H} := \{1, \ldots, H_{\max}\}$
$\mathcal{H}_t$	time window of task t; $\mathcal{H}_t := \{r_t, \ldots, H_{\max} - \delta_t\}$ (i.e., time steps in which task t can begin and be processed to completion)

Parameters

n	total number of jobs
n_j	total number of operations corresponding to job j
m	total number of resources
$H_{\max}$	size of planning horizon
r_t	release date of task t
d_t	due date of terminal task $t \in \widetilde{\mathcal{T}}$
p_{tk}	processing time of task t on resource k
$\overline{p}_{tk}$	largest processing time among tasks $T_k \setminus \{t\}$ competing with task t for resource k, $\overline{p}_{tk} = \max_{s \in \mathcal{T}_k \setminus \{t\}} \{p_{sk}\}$
δ_t	shortest possible remaining processing time from the starting time of task t to completion of corresponding terminal task
β_t	weights used for tardiness objective function
$C_{\max}$	makespan (i.e., time difference between the starting time of the earliest scheduled job and the time in which all jobs have been completed)

Variables

$$x_{tku} = \begin{cases} 1, & \text{if task } t \text{ starts in the beginning of time step } u \text{ on resource } k \\ 0, & \text{otherwise} \end{cases}$$

Glossary

FJSP	flexible job shop scheduling problem
JSP	job shop scheduling problem
SMSP	single-machine scheduling problem
PMSP	parallell-machine scheduling problem
MILP	mixed integer linear programming
IP	integer linear programming
LP	linear programming
VI	valid inequality
TI	time-indexed
LB	lower bound

Bibliography

[1] D. Behnke and M. J. Geiger, *Test instances for the flexible job shop scheduling problem with work centers*, (2012).

[2] L. Berghman, *Machine Scheduling Models for Warehousing Docking Operations*, PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, 2012.

[3] E. G. Birgin, P. Feofiloff, C. G. Fernandes, E. L. de Melo, M. T. Oshiro, and D. P. Ronconi, *A MILP model for an extended version of the flexible job shop problem*, Optimization Letters, (2013), pp. 1–15.

[4] E. H. Bowman, *The schedule-sequencing problem*, Operations Research, 7 (1959), pp. 621–624.

[5] P. Brandimarte, *Routing and scheduling in a flexible job shop by tabu search*, Annals of Operations research, 41 (1993), pp. 157–183.

[6] P. Brucker, *Scheduling algorithms*, Springer, Berlin Heidelberg, Germany, 5^{th} ed., 2007.

[7] Y. Demir and S. Kürşat İşleyen, *Evaluation of mathematical models for flexible job-shop scheduling problems*, Applied Mathematical Modelling, 37 (2013), pp. 977–988.

[8] P. Fattahi, M. S. Mehrabad, and F. Jolai, *Mathematical modeling and heuristic approaches to flexible job shop scheduling problems*, Journal of Intelligent Manufacturing, 18 (2007), pp. 331–342.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman & Co., New York, NY, USA, 1979.

[10] M. R. Garey, D. S. Johnson, and R. Sethi, *The complexity of flowshop and jobshop scheduling*, Mathematics of Operations Research, 1 (1976), pp. 117–129.

[11] IBM Corp., *IBM ILOG 12.2 User's Manual for CPLEX*, 2010.

[12] A. S. Jain and S. Meeran, *A state-of-the-art review of job-shop scheduling techniques*, Working paper, Department of Applied Physics, Electronic and Mechanical engineering. University of Dundee, Dundee, Scotland, (1998).

[13] T. Jansson, *Resource utilization in a multitask cell*, Master's thesis, Department of Mathematical Sciences, Chalmers University of Technology, Göteborg, Sweden, 2006.

[14] B. Jurisch, *Scheduling jobs in shops with multi-purpose machines*, PhD thesis, Fachbereich Mathematik/Informatik, Universität Osnabrück, Osnabrück, Germany, 1992.

[15] A. S. Manne, *On the job-shop scheduling problem*, Operations Research, 8 (1960), pp. 219–223.

[16] G. L. Nemhauser and L. A. Wolsey, *Integer and combinatorial optimization*, John Wiley & Sons, Inc., New York, NY, USA, 1988.

[17] C. Özgüven, L. Özbakir, and Y. Yavuz, *Mathematical models for job-shop scheduling problems with routing and process plan flexibility*, Applied Mathematical Modelling, 34 (2010), pp. 1539–1548.

[18] F. Pezzella, G. Morganti, and G. Ciaschetti, *A genetic algorithm for the flexible job-shop scheduling problem*, Computers & Operations Research, 35 (2008), pp. 3202–3212.

[19] W. R. Pulleyblank, *Polyhedral combinatorics*, Springer, Berlin Heidelberg, Germany, 1983.

[20] M. Queyranne and A. S. Schulz, *Polyhedral approaches to machine scheduling*, Preprint 408/1994, Department of Mathematics, Technical University of Berlin, (1994).

[21] J. P. Sousa and L. A. Wolsey, *A time indexed formulation of non-preemptive single machine scheduling problems*, Mathematical Programming, 54 (1992), pp. 353–367.

[22] K. Thörnblad, *On the Optimization of Schedules of a Multitask Production Cell*, licenciate thesis, Chalmers University of Technology and University of Gothenburg, Göteborg, Sweden, 2011.

[23] ——, *Mathematical Optimization in Flexible Job Shop Scheduling: Modelling, Analysis, and Case Studies*, PhD thesis, Chalmers University of Technology and University of Gothenburg, Göteborg, Sweden, 2013.

[24] K. Thörnblad, A.-B. Strömberg, M. Patriksson, and T. Almgren, *A competitive iterative procedure using a time-indexed model for solving flexible job shop scheduling problems*, Preprint nr 2013:10, Department of Mathematical Sciences, Chalmers University of Technology, Göteborg, Sweden, (2013).

[25] K. Thörnblad, A.-B. Strömberg, M. Patriksson, and T. Almgren, *An efficient algorithm for solving the flexible job shop scheduling problem*, In the precedings of the 25th NOFOMA conference, (2013), pp. 1–15.

[26] J. Van den Akker, C. A. Hurkens, and M. W. Savelsbergh, *Time-indexed formulations for machine scheduling problems: Column generation*, INFORMS Journal on Computing, 12 (2000), pp. 111–124.

[27] W. Xia and Z. Wu, *An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems*, Computers & Industrial Engineering, 48 (2005), pp. 409–425.