MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

"Flexible Job Shop Scheduling Problem using Iterative Combinatorial Auctions"

verfasst von / submitted by

Gabriel Reiter, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2016 / Vienna, 2016

Studienkennzahl lt Studienblatt: degree programme code as it appears on the student record sheet:	A 066 821
Studienrichtung lt Studienblatt: degree programme as it appears on the student record sheet:	Masterstudium Mathematik
Betreuer von / Supervisor:	o. Univ.-Prof. Dr. Arnold Neumaier

Abstract

This thesis puts forth an algorithm for solving the flexible job shop scheduling problem. The algorithm consist of a subgradient search and of a list scheduling heuristic, which is designed to generate feasible solutions deriving from the infeasible solutions we obtain from the subgradient search. Several methods for so-called price discrimination are introduced and their effects on the subgradient search are examined. The list scheduling heuristic uses a priority function to rank operations and creates a schedule that does not violate any constraints. Implementation in Java was written to test the method. The results of testing the algorithm on realistic industrial data are shown, compared to results from other programs and interpreted.

Zusammenfassung

Die vorliegende Masterarbeit beschäftigt sich mit dem Thema "job shop scheduling" (**jss**), also der Erstellung von Ablaufplänen. Beim **jss** sollen mehrere Aufträge so auf Maschinen aufgeteilt werden, dass ein möglichst dichter Ablaufplan entsteht. Jeder dieser Aufträge besteht aus Teilen, die in einer fixen Reihenfolge bearbeitet werden müssen und jeweils nur auf bestimmten Maschinen erledigt werden können.

Der Wert eines Ablaufplans wird als die Summe gewichteter Verspätungen berechnet. Jeder Auftrag bekommt ein Gewicht zugewiesen, das dessen Dringlichkeit beschreibt, sowie einem Fälligkeitstermin. Mit jeder Zeiteinheit Verspätung erhöht sich der Zielwert eines Auftrags um dessen Gewicht. Die Summe dieser Zielwerte ergibt den Wert des gesamten Ablaufplans. Ein guter Plan ist nun durch einen geringen Zielwert gekennzeichnet.

Als Strategie zur Lösung dieses Problems wird auf die Lagrange Relaxierung zurückgegriffen. Ein Subgradientenverfahren liefert eine untere Schranke für den Zielwert des Problems als auch einen unfertigen, unzulässigen Ablaufplan. Dieser Plan kann noch Bedingungen, die an solche gestellt werden, verletzen. So ist hier noch eine Doppelbelegung von Maschinen möglich. Mittels einer Heuristik, dem sogenannten List Scheduling, wird dieser zu einem fertigen, zulässigen Ablaufplan verändert und man erhält eine obere Schranke für das Problem.

Der Ansatz war, das Subgradientenverfahren so zu formulieren, dass es einer Auktion ähnelt. Dazu wird das Problem auf einzelne auftragsspezifische Probleme aufgeteilt. Jeder Auftrag berechnet einen persönlichen Ablaufplan und bietet nun für Zeitfenster auf den Maschinen. Hierbei fungieren die Lagrange-Multiplikatoren als Preise. Der Algorithmus geht iterativ vor, wobei in jeder Iteration die Preise für Zeitfenster auf Maschinen gemäß der Nachfrage verändert werden. Diese neuen Lagrange-Multiplikatoren dienen als Grundlage für die auftragsspezifischen Probleme der nächsten Iteration.

Im Zuge der Arbeit werden verschiedene Vorgehensweisen zur Adaptierung der Lagrange-Multiplikatoren vorgestellt. So werden auftragsspezifische Preise berechnet, die die Koordination zwischen den Aufträgen erleichtern soll.

Das List Scheduling nimmt das Ergebnis einer Iteration und erstellt daraus einen zulässi-

gen Ablaufplan, indem es jedem Teilauftrag eine Gewicht zuweist und bei Doppelbelegung den Auftrag mit dem geringeren Gewicht nach hinten verschiebt.

Der Algorithmus wurde in Java programmiert und anschließend an einer Auswahl von Musterbeispielen getestet. Hierbei wird der Einfluss der verschiedene Prozesse zur Adaptierung der Lagrange-Multiplikatoren verglichen. Zu Beginn der Arbeit wird ein Überblick über die geschichtliche Entwicklung der Ablaufplanung als auch der kombinatorischen Auktion gegeben.

Contents

Chapter 1

Introduction

Lawler, Lenstra, Rinnoy Kan, and Shmoys (1993, p. 445) state that "sequencing and scheduling is concerned with the optimal allocation of scarce resources to activities over time". Scheduling is ubiquitous, even now you may think about what to do next and when to do it. Informal scheduling was always a part of factories from the middle of the nineteenth century onwards, when foremen were tasked to organise their shops. More formal approaches began to appear in the late nineteenth century for example with the paper of Binsse (1887). Frederick Taylor in the late nineteenth century was the first to separate planning from execution and his pupil Henry Gantt (1919) provided us with the scheduling and planning tool called the Gantt chart, which is still in use today. Naturally, there have been many research innovations in scheduling since the work of Gantt and Taylor.

The purpose of this work is to present the notion of job shop scheduling and to give some insight into the usage of combinatorial auctions in mathematical optimization. It will put these two pieces together and to deal with the problem of job shop scheduling with the help of combinatorial auctions. To this end a computer program tackling the problem was written and tested on problem instances. The results of these test offer an insight into the workings of the algorithm.

The focus of this thesis is on job shop scheduling, therefore not the history of scheduling but a succinct history of job shop scheduling will be given in the following. A more detailed history of scheduling is presented in Herrmann (2006), where the short overview above was also taken from.

A classical job shop scheduling problem ($I \times M$) consists of I jobs which have to be executed on M machines. A job can be the completion of some kind of product,. Each job requires a specific choice of machines in a predetermined order. A part of a job which is scheduled on a machine is called an operation of a job. Therefore, a job consists of a sequence of operations and the question "which operation of which job is best scheduled

on which machine at what time?" arises.

Job shop scheduling emerged in the 1950 with papers of Johnson (1954), J. Jackson (1956) and Akers and Friedman (1955). The former offered an optimal algorithm for the two machine flow shop problem, which means that every job has the same sequence of machines. In his paper Jackson deals with the $2 \times M$ problem where each job has at most two operations. Akers and Friedman on the other hand work with the $I \times 2$ problem, with both Jackson and Akers operating in the job shop environment. The book "Industrial Scheduling" edited by Muth and Thompson (1963) collected all research at the time and was a basis for following investigations.

The Gantt chart, a way of depicting a schedule, was introduced by Gantt (1919). A different representation, the disjunctive graph, was proposed by Roy and Sussman (1964) and extended by White and Rogers (1990). It facilitates a very visual way to deal with the scheduling problem, for example the makespan, the total time needed to complete all jobs, corresponds to the length of the longest path in this graph.

In the next decade the main focus of research was finding exact solutions using enumerative algorithms like branch and bound. One example of the enumerative approach is by Balas (1965) who utilizes the disjunctive graph representation. Due to the complexity of the problem, these exact algorithms are of limited practical value. Computational approaches to job shop problems based on these algorithms show in the worst case exponential runtime as expected from a NP-hard problem.

In the 70s and 80s the research centred around the complexity of the job shop problem, as in for example Cook (1971) and Lawler, Lenstra, and Rinnooy Kan (1982). Since then many approximation algorithms have been proposed, the first by Panwalkar and Iskander (1977) using priority dispatch rules. Other approaches make use of fuzzy logic Grabot and Geneste (1994), genetic local search Moscato (1989), Grefenstette (1987) and Dorndorf and Pesch (1995), tabu search Glover (1989) and Glover (1990), Taillard (1994), Nowicki and Smutnicki (1996), simulated annealing Van Laarhoven, Aarts, and Lenstra (1992), Sadeh and Nakakuki (1996) and genetic algorithms Falkenauer and Bouffoix (1991), Nakano and Yamada (1991). Furthermore shifting bottleneck procedure Adams, Balas, and Zawack (1988) or large step optimisation Lourenço (1993) and Lourenço (1995), Lagrangian relaxation Hoitomt, Luh, and Pattipati (1993) and Wang, Luh, Zhao, and Wang (1997) have been applied. All of the above is a focused summary of previous research, largely based on the paper by Jain and Meeran (1999). An extensive discussion of different approaches to the job shop scheduling problem is given in Jones, Rabelo, and Sharawi (1999).

A literature review by the author revealed that newer works apply variable neighborhood search Sevkli and Aydin (2006), Adibi, Zandieh, and Amiri (2010), Amiri, Zandieh, Yazdani, and Bagheri (2010), ant colony algorithms Zhou, Lee, and Nee (2008) and a combination of a genetic algorithm and variable neighbourhood search Gao, Sun, and Gen (2008). combinatorial auctions together with Lagrangian relaxation were used to solve

the problem in Kutanoglu and Wu (1999), Dewan and Joshi (2002), Liu, Abdelrahman, and Ramaswamy (2007). Or as Lawler, Lenstra, Rinnooy Kan, and Shmoys (1993, p. 446) have put it "all techniques of combinatorial optimization have at some point been applied to scheduling problems".

The second major facet considered in this thesis, besides scheduling, is combinatorial auctions. A combinatorial auction differs from a common auction in the fact, that in a combinatorial auction a bidder can place bids on a combination of items. A bidder could, for example, say: "I want either all of these three items, or both of these two, but not both the three and the two items." This way a bidder can put a value on groups of items more easily and can act in a safer environment. The following overview of research concerning combinatorial auctions and usage of these auctions is mainly taken from de Vries and Vohra (2007), to which readers who are inclined to learn more about combinatorial auctions in general, are referred to.

The research in combinatorial auctions was sparked by Rassenti, Smith, and Bulfin (1982), when a combinatorial auction was used to allocate airport time slots. Even before that, in C. Jackson (1976) combinatorial auctions were suggested for radio spectrum rights. Strevell and P. (1985) proposed combinatorial auctions for vacation time slots, Banks, Ledyard, and Porter (1989) for selecting projects on space shuttles and Caplice (1996) for shipper-carrier relationships. On the other hand Srinivasan, Stallert, and Whinston (1998) applied combinatorial auctions to trading financial securities, Kutanoglu and Wu (1999) and (Wellman, Walsh, Wurman, & MacKie-Mason, 2001) to scheduling and Davenport and Kalagnanam (2002) used them for large food manufacturers. Bikhchandani and Huang (1993) and Ausubel and Cramton (2002) describe the auction of treasury securities used by the U.S. Department of Treasury, Ledyard, Olson, Porter, Swanson, and Torma (2002) an auction used by Sears to select carriers. Furthermore Dai, Chen, and Yang (2014) devised an auction scheme for carrier collaboration, Liu et al. (2007) one for dynamic job shop scheduling, Kumar, Kumar, Tiwari, and Chan (2006) for scheduling in the steel making process.

Another research topic in connection with combinatorial auctions is winner determination, it deals with the question "who gets what?" and is, among many others, tackled in Sandholm (1999), Fujishima, Leyton-Brown, and Shoham (1999) and Rothkopf, Pekec, and Harstad (1998). Nisan (2000) and again Rothkopf et al. (1998) examine how bids can be expressed and how restriction to bidding on specific bundles influences the auction.

This thesis is structured as follows. The first chapter provides the basic definitions and concepts regarding job shop scheduling and combinatorial auctions. It revisits the topics of linear programming and Lagrangian relaxation and states definitions in job shop scheduling. The second chapter introduces combinatorial auctions and connects them to both Lagrangian relaxation and job shop scheduling. The definitions and result from chapters 2 and 3 are put to use in chapter 4 where the program designed is presented. The result of its usage and problems occurred are demonstrated in chapter 5.

While working on this thesis, I was employed at Profactor in Steyr. The basis for my thesis was already laid and I could build on the works an knowledge of my co-workers. The implementation was done using Eclipse and I was able to use its powerful debugging tools. The problem instances I tested my program on, were supplied by Prof. Dr. Lars Mönch of the FernUniversität Hagen, who provided not only the necessary data but also the best results previously achieved.

The main Results of this thesis are on the one hand a way to keep the lower bound property of the subgradient searches when using personalized prices. And on the other hand, the insight that personalized prices downgrade the lower bounds of the subgradient search, is gained. The effect of these prices on the feasible schedules was indecisive and maybe more experiments and improvement on the methods from this thesis can bring a way to augment subgradient searches through more elaborate pricing to light.

Chapter 2

Mathematical Basics

This chapter will provide definitions of the mathematical concepts, which are used throughout the paper, and lay the foundations for the chapters to come. The first topic will be **job shop scheduling**, where different aspects of the problem are looked at and the notation used in this thesis is stated. For example we will see various criteria for determining the performance of a particular schedule. Then **linear programming** and **Lagrangian relaxation** are introduced, which will later be used to formulate the job shop scheduling problem as a linear program. Furthermore, the concepts of the simple subgradient search and the surrogate subgradient search are introduced.

2.1 Job Shop Scheduling

The following part on notation, basic concepts and definitions relies heavily on the book Pinedo (2008), where a more elaborate taxonomy can be found.

A **job shop scheduling problem** (**jss**) consist of a set of I jobs $\mathcal{I} = \{0, 1, \ldots, I-1\}$ which need to be scheduled on a set of M machines $\mathcal{M} = \{0, 1, \ldots M-1\}$. Each job $i \in \mathcal{I}$ consists of J_i operations $\mathcal{J}_i = \{0, 1, \ldots J_i - 1\}$. An operation is represented by a pair (i, j) where $i \in \mathcal{I}$ and $j \in \mathcal{J}_i$, which means that the jth operation of the ith job is considered.

In this environment there is exactly one suitable machine $m_{ij} \in \mathcal{M}$ for the operation (i, j), i.e., the operation (i, j) can only be completed, and therefore scheduled, on the machine m_{ij}. P_{ij} is the time needed for operation j of job i to be completed. The sequence of operations of a job $i = \big((i, 0), (i, 1), \ldots (i, J_i - 1)\big)$ together with the machines m_{ij} is called the routing of a job. If the routing requires a job to visit a machine more than once it is said to **recirculate**. A solution of a job shop scheduling problem assigns a beginning time b_{ij} to each operation (i, j).

An extension of the job shop problem is the **flexible job shop scheduling problem** (**fjss**), where an operation of a job has not only one suitable machine, but a set $\mathcal{H}_{ij}$ of suitable machines. For each machine $m \in \mathcal{H}_{ij}$ there is a certain process time P_{ijm}, which is now also dependent on the machine m. To solve such a problem we need not only assign a beginning time to an operation but also a machine m from the set of suitable machines $\mathcal{H}_{ij}$. This means that the problem from above is extended and additionally to a table of beginning times, we also have to find a routing for each job.

Until now we have assumed that the transportation from one machine to another does not take any time. If we additionally want to take into consideration, that this is not the case, we are presented with the **flexible job shop problem with travel times** (**fjsstt**). This means we have additional information R_{mn} being the travel time from machine m to machine n.

2.1.1 Problem Description

Parameters

The following parameters will play roles in our problem. Their formulation is taken from Hämmerle, Weichhart, and Vorderwinkler (2015).

- $i \in \mathcal{I} = \{0, 1, \dots I-1\}$ the set of jobs
- $m \in \mathcal{M} = \{0, 1, \dots M-1\}$ the set of machines
- $\mathcal{J}_i = \{1, 2, \dots J_i - 1\}$ the set of operations of job i with index $(i, j) : i \in \mathcal{I}$ and $j \in \mathcal{J}_i$
- $\mathcal{H}_{ij}$ the set of suitable machines for operation (i, j)
- P_{ijm} the processing time of operation (i, j) on machine m
- $k \in \mathcal{K} = \{0, 1, \dots K-1\}$ the set of available time slots, we consider time to be discrete
- R_{mn} the transport time from machine m to machine n
- D_i the due date for job i
- W_i the (tardiness) weight of job i, expresses the importance of a job

Objective Functions

The following list of objective functions as well as the lost of constraints are just a selection of choices offered by Pinedo (2008) and more possibilities can be found in his book.

The objective of scheduling is to optimize some aspect of a process. If the completion time of operation (i, j) is c_{ij} we write $(C_i) = c_{iJ_i-1}$ as the completion time of job i. Some examples of objective functions, all of which depend on the completion time and are non-decreasing, to be minimized are:

Makespan

In this case we try to minimize the time it takes to complete all jobs i.e. $\max\{C_o, C_1, \dots C_{I-1}\}$ which is the time, the last job is finished.

Maximum Lateness

With the lateness of job i defined as $L_i = C_i - D_i$ one can seek to minimize $\max\{L_0, L_1, \dots, L_{I-1}\}$. This means that one wants to minimize the violation of due dates.

Total Weighted Completion Time

One can also try to minimize $\sum_i W_i C_i$. This sum is called the total weighted completion time. For example one could see a large job weight as a large inventory cost for this job and one therefore wants to prioritize this specific job.

Total Weighted Tardiness

If we define the tardiness of job i as $T_i = \max\{L_i, 0\}$, we can also minimize $\sum_i W_i T_i$. L_i is again the lateness and computed by $L_i = C_i - D_i$. We call this weighted tardiness and a job with a large weight could for example be a job for a important costumer.

Other Objectives

Other aspects to be optimized are for example the **maximal machine workload**, which is the maximum over all machines of working time on a single machine. Optimizing this means that we seek to balance the workload on all machines. If machines have different speeds, i.e. if processing times P_{ij} also depend on the machine used, we can minimize the **total machine workload** as well. This then is the total working time on all machines.

Constraints

Various kinds of constraints can be incorporated into the concept of scheduling.

Release Dates

If we are presented with a release date r_i for each job, we have to find a schedule where $b_{i0} \geq r_i$ for all $i \in I$. This means that we cannot start processing a job before its release date.

Preemptions

A schedule is called non-preemptive if once an operation is started on a machine, it cannot be interrupted by an operation of another job. If preemptions are allowed, this is possible and the time already processed for the first operation before the interruption is not lost. In other words when this operation is continued on a machine, only the remaining processing time is needed.

Precedence Constraints

Precedence constraints describe the fact that often some operation (i, j_r) has to be completed before operation (i, j_s) where without loss of generality $r < s$. One could even be presented with inter-job specific precedence constraints, meaning that (i_{l_1}, r) has to be processed before (i_{l_2}, s). We talk of chain like precedence constraints if each operation has at most one predecessor and successor. The terms intree and outtree are used to describe the cases where each operation has at most one successor respectively one predecessor. In this paper we will only consider chain like precedence constraints, which means that (i, r) has to be processed before $(i, r+1)$ for all $i \in I$ and for all $r \in \{1, 2, \ldots J_i - 2\}$.

2.2 Mathematical Programming and Lagrangian Relaxation

Mathematical programming encompasses a wide range of different problems and methods. The book Dantzig and Thapa (1997) is the basis for this chapter. In this thesis we will look at the notions of linear programming, integer programming as well as nonlinear programming. Dantzig and Thapa (1997, p 1) write that:

> *Mathematical programming (or optimization theory) is that branch of mathematics dealing with techniques for maximizing or minimizing an objective function subject to linear, nonlinear, an integer constraints on the variables.*

Linear programming is a special case thereof, in which constraints on variables are always linear and can be either equality or inequality constraints. A linear programming problem may be stated in the following form:

$$\begin{aligned} \mathcal{Z} = \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & Dx \leq e \\ & x \geq 0. \end{aligned} \tag{2.1}$$

Where A is a $m \times n$ matrix and the vectors c, x are $n \times 1$ vectors and b is a $m \times 1$ vector. The phrase 's. t.' means 'such that' and indicates the constraints to be fulfilled. The so called **dual** of this linear program is another linear program of the form:

$$\begin{aligned} \mathcal{Y} = \max \quad & b^T y \\ \text{s.t.} \quad & Ay \leq c. \end{aligned} \tag{2.2}$$

A again is a $m \times n$ matrix an the vectors b, y and c have suitable dimensions. A well know fact is that $\mathcal{Y} \leq \mathcal{Z}$, which means that the dual problem bounds the initial one. The difference $\mathcal{Z} - \mathcal{Y}$ is called the duality gap. If the duality gap equals 0, we say that strong duality holds. Otherwise the property is called weak duality.

The field of mathematical programming started to develop in the 1940s and has been of great interest for mathematicians since then. Numerous different methods of tackling problems of this nature have been devised and we leave a detailed discussion of this topics aside. In 2.1.1 it is stated that we work with time slots and therefore assume time to be discrete. Hence allocating an operation to a machine is a discrete problem and our variables are integers. We will now briefly look at a variation of linear programs, so called integer programs. The formulation of an integer program looks exactly like the formulation of a linear program except that we add the requirement for x to be integer.

$$\begin{aligned} \mathcal{Z}_P = \min \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b \\ & Dx \leq e \\ & 0 \leq x \in \mathbb{Z}^n. \end{aligned} \tag{2.3}$$

It is easy to find linear programs that have no solution satisfying all requirements. Since such programs are of no interest to us, we will assume that from this point onwards all linear programs are feasible.

2.2.1 Lagrangian Relaxation

Lagrangian relaxation is a technique to help find a solution to a linear program conforming to its constraints. The following treatment of Lagrangian relaxation is largely based on the paper by Fisher (2004). The idea behind it is that this method can make a program easier to solve if we include some constraints into the objective function. We assume that we have split the constraints into two parts A and D, with A containing only equality constraints. We now reformulate the problem (2.3)

$$\begin{aligned} \mathcal{Z}_{LR}(\lambda) = \min \quad & c^T x + \lambda(Ax - b) \\ \text{s.t.} \quad & Dx \leq e \\ & 0 \leq x \in \mathbb{Z}^n. \end{aligned} \tag{2.4}$$

The vector λ has suitable dimensions for matrix multiplication and its entries are called Lagrange multipliers. A convenient property of the objective value $\mathcal{Z}_{LR}(\lambda)$ of the Lagrangian relaxation is the fact $\mathcal{Z}_{LR}(\lambda) \leq \mathcal{Z}_p$. If we consider x' to be an optimal solution to the program (2.3), we obtain the inequality

$$\mathcal{Z}_{LR}(\lambda) \leq c^T x' + \lambda(Ax' - b) = \mathcal{Z}_P, \tag{2.5}$$

since $\lambda(Ax' - b) = 0$ and $c^T x' = \mathcal{Z}_P$. A significant decision regarding the Lagrangian relaxation is the choice of vector λ. Since $\mathcal{Z}_{LR}(\lambda)$ offers a lower bound to $\mathcal{Z}_P$ the best choice would be one such that

$$\mathcal{Z}_{LR} = \max_{\lambda} \mathcal{Z}_{LR}(\lambda). \tag{2.6}$$

Since this program is dual to (2.4) it is sometimes referred to as the Lagrangian dual program.

2.2.2 Subgradient Search

We have become sidetracked on the quest of the search for an optimal solution to an integer program and are searching for a vector λ to help us find an optimal lower bound for the objective function value. Here, a method called subgradient search comes into play. Since our function $\mathcal{Z}_{LR}$ is not necessarily differentiable, we may not be able to use a gradient method to find our desired λ. Therefore, we will make use of so called subgradients, which are a generalization of gradients for non-differentiable functions. For

a convex function $f(\lambda): \quad U \to \mathbb{R}$, where U is a convex subset of $\mathbb{R}^m$, a subgradient at point λ' in U is a vector v in $\mathbb{R}^n$ satisfying

$$f(\lambda) - f(\lambda') \geq v(\lambda - \lambda'). \tag{2.7}$$

The subgradient method performs an iteration with steps $\lambda_{k+1} = \lambda_k + s_k v_k$ where s_k is called the step size and v_k is a subgradient. One can prove under some assumptions that the points $f(\lambda_k)$ generated by the subgradient method converge to the minimum of the function $f(\lambda)$. Since according to Wang et al. (1997) our function $\mathcal{Z}_{LR}$ is not convex but rather concave we will reformulate this property as

$$\mathcal{Z}_{LR}(\lambda) - \mathcal{Z}_{LR}(\lambda') \leq v(\lambda - \lambda') \quad \forall \lambda. \tag{2.8}$$

The vector $Ax_k - b$ is a subgradient to if x_k is an optimal solution to the problem $\mathcal{Z}_{LR}(\lambda)$. Thus the subgradient method applied to our problem generates a sequence of vectors λ_k of Lagrange multipliers according to the rule

$$\lambda_{k+1} = \lambda_k + s_k(Ax_k - b). \tag{2.9}$$

In this setting x^k is again an optimal solution to the problem (2.4) with λ_k substituted for λ. The step size s_k is often computed by

$$s_k = \frac{\alpha_k(\mathcal{Z}_\lambda - \mathcal{Z}_{LR}(\lambda_k))}{\| Ax_k - b \|^2}, \tag{2.10}$$

where $0 < \alpha \leq 2$ and $\mathcal{Z}_\lambda$ is an upper bound for the value of $\mathcal{Z}_{LR}$. An approach which has proven to be reasonable is to set $\alpha_0 = 2$ and halving it if the value $\mathcal{Z}_{LR}(\lambda)$ did not increase for a number of iterations. Usually $x_0 = 0$ is chosen as a starting point and since we cannot prove the optimality of the subgradient method, unless $\mathcal{Z}_{LR}$ equals the value of a known feasible solution, we performed this procedure for a fixed number of iterations. For this method to work we have to be able to solve the problem $\mathcal{Z}_{LR}(\lambda)$ optimally which can be very time consuming for big problems.

2.2.3 Surrogate Subgradient Search

In certain instances the relaxed problem (2.4) can be split up into a number of subproblems. Each of these subproblems can then be solved on its own and added up to yield the value $\mathcal{Z}_{LR}(\lambda)$. If this is the case and one is confronted with the difficulty of solving numerous subproblems, the surrogate subgradient method provides a technique

that allows one optimize, though not all, but a number of subproblems in one iteration. This method presented in this section is due to Zhao, Luh, and Wang (1999). Thus, it reduces computational requirement since we do not need to solve $\mathcal{Z}_{LR}$ optimally in each iteration and still has some of the desired properties of the subgradient method. Accordingly, suppose the problem can be separated, then we write

$$\min c^T x + \lambda(Ax - b) = \sum_{l=1}^{L} \left(c_l^T x_l + \lambda A_l x_l\right) - \lambda b \tag{2.11}$$

and postulate the surrogate dual problem as an extension to the Lagrange relaxed problem (2.4).

$$\begin{aligned} Z_{SG}(x,\lambda) = \min \quad & c^T x + \lambda(Ax - b) \\ \text{s.t.} \quad & Dx \leq e \\ & 0 \leq x \in \mathbb{Z}^n. \end{aligned} \tag{2.12}$$

One now again uses an iterative process to generate a sequence of points. In one such iteration step, given x_k and λ_k, first the vector

$$\lambda_{k+1} = \lambda_k + s_k(Ax_k - b) \tag{2.13}$$

is computed, where s_k is the stepsize and satisfies

$$s_k = \alpha_k \frac{s_{k-1}\|Ax_{k-1} - b\|}{\|Ax_k - b\|}. \tag{2.14}$$

Then approximate optimization for x_{k+1} is performed such that

$$Z_{SG}(x_{k+1}, \lambda_{k+1}) < Z_{SG}(x_k, \lambda_{k+1}). \tag{2.15}$$

This means that one must not necessarily optimize all subproblems fully, but enough to fulfill the above inequality, which can be done with approximate optimization. If one adheres to these restrictions one can show that

$$0 \leq Z_{LR} * - Z_{SG}(x_k, \lambda_k) \leq (\lambda' - \lambda_k)^T (Ax_k - b). \tag{2.16}$$

Z_{LR}^* and λ^* are the optimum solution to Z_{LR} and the corresponding Lagrangian multipliers, respectively. It can as well be shown that $\|\lambda' - \lambda_k\| < \|\lambda' - \lambda_{k-1}\|$.

2.3 Problem Formulation

The following passage is dedicated the concrete problem formulation and the definitions of both parameters and decision variables, which will be used in the remainder of this thesis, are given. We recall the definitions of parameters form the section 2.1.1 and introduce the decision variables. Just as the section 2.1.1 this section is based on Hämmerle et al. (2015).

Parameters

- $i \in \mathcal{I} = \{0, 1, \dots I-1\}$ the set of jobs
- $m \in \mathcal{M} = \{0, 1, \dots M-1\}$ the set of machines
- $\mathcal{J}_i = \{1, 2, \dots J_i - 1\}$ the set of operations of job i with index $(i, j) : i \in \mathcal{I}$ and $j \in \mathcal{J}_i$
- $\mathcal{H}_{ij}$ the set of suitable machines for operation (i, j)
- P_{ijm} the processing time of operation (i, j) on machine m
- $k \in \mathcal{K} = \{0, 1, \dots K-1\}$ the set of available time slots, we consider time to be discrete
- R_{mn} the transport time from machine m to machine n
- D_i the due date for job i
- W_i the (tardiness) weight of job i, expresses the importance of a job

Decision Variables

- $b_{ij}, i \in \mathcal{I}, j \in \mathcal{J}_i$ the beginning time of operation (i, j)
- $c_{ij}, i \in \mathcal{I}, j \in \mathcal{J}_i$ the completion time of operation (i, j)
- $m_{ij} \in \mathcal{H}_{ij} i \in \mathcal{I}, j \in \mathcal{J}_i$ the machine assigned to operation (i, j)
- $\delta_{ijmk}, i \in \mathcal{I}, j \in \mathcal{J}_i, m \in \mathcal{M}, k \in \mathcal{K}$ a binary variable which is 1 if operation (i, j) is scheduled on machine m in time slot k
- $\lambda_{mk}, m \in \mathcal{M}, k \in \mathcal{K}$ the Lagrange multiplier for machine m in time slot k

Constraints

It is easy to see that the decision variables δ_{ijmk}, b_{ij} and c_{ij} are not independent since we have the identity:

$$\delta_{ijmk} = \begin{cases} 1 & \text{if } b_{ij} \leq k \leq c_{ij} \text{ and } m = m_{ij} \\ 0 & \text{otherwise.} \end{cases} \tag{2.17}$$

Due to the non-preemptive constraints we also have

$$c_{ij} = b_{ij} + P_{ijm_{ij}} - 1, \forall i \in \mathcal{I}, \forall j \in \mathcal{J}_i, \forall m \in \mathcal{H}_{ij}. \tag{2.18}$$

Since operation (i, j) can only be started when $(i - j - 1)$ has been completed, we also have the precedence constraints

$$b_{ij} \geq c_{ij-1} + 1 + R_{m_{ij-1}m_{ij}}, \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{J}_i, \forall m \in \mathcal{H}_{ij}. \tag{2.19}$$

The fact that $R_{m_{ij-1}m_{ij}}$ appears in the previous constraints makes them non-linear. To linearize these constraints, we introduce auxiliary variables x_{ijm} and $z_{ijm_1m_2}$ defined as

$$x_{ijm} = \begin{cases} 1 & \text{if } m_{ij} = m \\ 2 & \text{otherwise.} \end{cases} \tag{2.20}$$

and

$$z_{ijm_1m_2} = \begin{cases} 1 & \text{if } m_1 = m_{ij-1} \text{ and } m_2 = m_{ij} \\ 0 & \text{otherwise.} \end{cases} \tag{2.21}$$

Additional constraints

$$z_{ijm_1m_2} \leq 0.5(x_{ij-1m_1} + x_{ijm_2} - 2) + 1 \tag{2.22}$$

force $z_{ijm_1m_2}$ to equal 0 exactly when either x_{ijm_1} or x_{ijm_2} or both do. The precedence constraints from (2.19) can now be reformulated into linear constraints as

$$b_{ij} \geq c_{ij-1} + 1 + \sum_{m_1, m_2 \in \mathcal{M}} z_{ijm_1m_2} R_{m_1m_2}, \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{J}_i. \tag{2.23}$$

The capacity constraints, which state that a machine can only process one operation at time, read

$$\sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}_i} \delta_{ijmk} \leq 1, \quad \forall m \in \mathcal{M}, \forall k \in \mathcal{K}. \tag{2.24}$$

Objective Function

The objective we are trying to minimize is the total weighted tardiness, which is computed as

$$\mathcal{Z} = min_{b_{ij} m_{ij}} \sum_{i \in \mathcal{I}} W_i T_i. \tag{2.25}$$

The tardiness T_i is defined as $\max(0, C_i - D_i)$, with the completion time C_i of job i being $c_{i\mathcal{J}_i - 1}$.

Chapter 3

Combinatorial Auctions for Job Shop Scheduling

We have stated above that the job shop scheduling problem is intractable so we will use a heuristic to find a good solution to the problem. In this thesis **combinatorial auction** is used to solve the Lagrangian dual problem. Thus the following part deals with combinatorial auctions and their application to job shop scheduling. We will discuss combinatorial auctions as a possible approach to solve the problem formulated above and how these two concepts are connected. A way to modify the Lagrangian function from the previous chapter to resemble an auction is shown. Also we will see why and how the **Lagrangian relaxation** is of use for us and can lead to the use of auctions.

3.1 Auctions

For many years, many different types of auctions have been used to sell and buy goods. Even so, choosing the right kind of auction for a given situation is difficult Klemperer (2002). Thus, auction design is a busy research area and many different types of auctions have been introduced. In the following passage a short overview of different approaches is provided.

Some aspects to be considered in designing an auction according to Kalagnanam and Parkes (2004) are:

Market structure

How many buyers and sellers are there? The most commonly known case is the so called **forward auction**, where one seller tries to sell goods to a number of interested buyers. The opposite case of one buyer and multiple sellers is called the **reverse auction** and

is common in procurement. The third instance is that of multiple sellers and multiple buyers and is called **double auction** or **exchange**.

Goods

Another facet of auctions is the quantity and kind of goods that are sold. It ranges from a single unit of an item to multiple units of a single item, single units of different items and multiple units of multiple items. Except for the first instance, it is possible that goods are not sold or demanded individually but in groups, called bundles.

Bid Structure

The manner in which bids are structured limits the possibility of the buyers to communicate their demand and interests. If there are multiple units of a single item available, buyers might want to include some kind of quantity discount into their bids. In an auction where multiple different items are sold, buyers can prefer to formulate logical conjunctions in their bid. Such bids may, for instance, take the form "I will pay 42 € for good A and B, but only if I get both" or "I want either A or B, but not both, and will pay 42 €" An auction designed to give the buyers the chance to voice such complex demand is called a **combinatorial auction**.

Information Feedback

The questions of if and how bidders should receive feedback about their bids is also central in auction design. If an auction uses a **direct** mechanism, the bidders receive no feedback from the auctioneer, wich is the case for a single-round sealed bid auction. In an **indirect** mechanism, for example an ascending price auction, bidders receive feedback by a **price signal** or a **provisional allocation**, with whose help bids can be readjusted. It can distinguished between a **price setting** approach and a **quantity setting** approach. Within the price setting approach in every round information feedback is given through a price update by the auctioneer and the bidders announce their demand with respect to the new prices. In the latter, a provisional allocation is provided by the auctioneer and the bidders readjust the prices in their bids to obtain the desired goods or bundles and change the allocation. In both cases the feedback procedure is performed every iteration.

Behaviour of Bidders

Another important question of auction design is how to get agents to bid in a reasonable manner. When modeling the behaviour of agents, especially that of bidders, one can distinguish between two approaches: The first approach stems from **game theoretic** considerations where agents play best response strategies to each other and an equilibrium is defined as a state, where one-sided deviation from a strategy does not produce better outcomes. The second path is called **price-taking** or **myopic best-response** in which

case an agent answers with a best response to a current price and not to other bidders; an equilibrium is styled as an **competitive equilibrium**. These explanations assume that bidders reveal their valuation of goods and bundles as well as their desires truthfully. Untruthful bidding could potentially be used to manipulate an auction to ones favour and much effort is made to design auctions to encourage truthful bidding. Throughout the rest of this thesis, it is assumed that the bidders act truthfully within the auction.

3.1.1 Bidding Language

A major part of auctions is communication. Therefore, it is essential to specify the manner in which information is transmitted between participants. References that deal with the possibilities and consequences of bidding language are, for example, Nisan (2000), Boutilier and Hoos (2001). In auctions where multiple units are disposable it is favourable to allow bidders to state their preferences. However, if bid formulation is not subject to restrictions, a bidder could specify up to 2^n bids for a set consisting of n items.

Thus, it is necessary to limit and specify the types of bids a bidder can post, allowing the bidder to formulate his desires, on the one hand, and making the bids processable, on the other hand. This is done by means of a bidding language. Two types of bidding languages have been introduced: $\mathbb{L}_G$, where goods are logically combined and $\mathbb{L}_B$, where bundles of goods are components of bids.

In $\mathbb{L}_G$, bids for single items are linked through logical conjunctions and a price the bidder is willing to pay for this package is added. The language $\mathbb{L}_B$, however, consists of bids for bundles which are linked through logical conjunctions. This language comes in two versions: There is the **additive-or** language $\mathbb{L}_B^{OR}$, where one or more of the mentioned bids can be served and the price is the sum of these individual bids. The other possibility is **exclusive-or** $\mathbb{L}_B^{XOR}$ where at most one bid can be served.

3.2 Design of a Combinatorial Auction

In this section, the concrete auction protocol, which will be implemented in the next chapter, is brought forward. We try to shed light on every possible aspect of the protocol. The first part will deal with the general design principles as mentioned above. The second part deals with the method of price updating, which is of crucial interest for our implementation.

3.2.1 General structure

Market Structure

The combinatorial auction is designed as a forward auction where an auctioning agent, who represents all machines in our environment, tries to sell his goods to a number of bidders, each serving as an agent for a single job. The set of bidders is, therefore, identical to the set of jobs and is indexed equally as $i \in \mathcal{I} = \{0, 1, \ldots I-1\}$.

Goods

The goods to be sold by the auctioning agent are the available time slots of the machines at hand. Since every single time slot on a machine is sold as a single good, there are $|M||K|$ goods allocatable.

Information Feedback

An indirect protocol is chosen and a price setting approach is used to resolve conflicts between bidders. The auction proceeds in rounds, in which the auctioneer releases a price vector containing every price for every single time slot in each round. The bidders then submit their bids in response to these current prices. Additionally, a provisional allocation is posted, but the main tool to coordinate bidders is the price update. A more detailed insight to how this price update is realized will be delivered below.

Bid Structure and Bidding Language

The bids consist of a job id and a list of desired time slots. These time slots define the whole schedule of a job and time slots on a machine not to be split are grouped together to satisfy the non-preemption constraints of job operations. The value of the objective function of a job, if it obtains the time slots bid for, is also included into the bid. Since no alternative schedules are bid for, the bidding language is neither $\mathbb{L}_B^{OR}$ nor $\mathbb{L}_B^{XOR}$ and since the time slots grouped together constitute a bundle the language is also not $\mathbb{L}_G$.

Behaviour of Bidders

Furthermore, we expect bidders to follow a price-takings strategy and to bid truthfully as well as reveal their true objective function value.

3.2.2 Price Update

The fact that price update is the tool used to resolve conflicts between bidders and to coordinate these bidders, these mechanisms merit our special attention. Different methods to update prices, often called tâtonnement, are implemented and their results are examined. From now on, if we speak about a mechanism, we mean the way prices are updated during our auction.

The main idea is to adjust prices as a function of demand. If there is excess demand the prices are raised and in the case of a lack of demand, prices will be reduced. Since a good is a single time slot on a specific machine and a machine can only process one job at a time, excess demand is given when more than one job agent bids for a certain time slot. Thus, the formula for excess demand γ_{mk} for a time slot and, consequentially, for the price update λ_{mk} for a time slot is:

$$\gamma_{mk} = \sum_{i \in \mathcal{I}} \sum_{j \in | \in \mathcal{J}_\rangle} \delta_{ijmk} - 1 \tag{3.1a}$$

$$\lambda_{mk}^{l+1} = \max \left\{ 0, \lambda_{mk}^{l} + f\left(\gamma_{mk}^{l}\right) \right\}. \tag{3.1b}$$

The superscript l indicates the auction round, the subscripts m and k are the afore mentioned machine and time slot variables. The choice of the function f generates multiple different possibilities. The first and simplest is to choose f as a linear function with a constant multiplier. This results in

$$f\left(\gamma_{mk}^{l}\right) = s * \gamma_{mk}^{l}. \tag{3.2}$$

This approach is often called **Walrasian** or **non-adaptive tâtonnement** and s is called the step size. The nomenclature for the forms of price update as well as their formulations are taken from Kutanoglu and Wu (1999) and this form of price update stems from Walras (1954).

A more sophisticated choice would be to raise the prices quickly in the first rounds of the auction and to fine-tune the auction in later rounds. This can be done by multiplying the demand with larger factors in the beginning and scaling them down afterwards. Then the price update follows

$$f\left(\gamma_{mk}^{l}\right) = s_l * \gamma_{mk}^{l}, \tag{3.3}$$

where s_l is a decreasing function and the name given to this type of price updating mechanism is **adaptive tâtonnement**, thereby contrasting the previous choice.

Until now, the proposed mechanisms produced the same prices for every bidder. But since Jennergren (1973) and Jose, Harker, and Ungar (1997) showed that uniform prices generally do not lead to an equilibrium when bidders are characterized by linear programs we would be wise to choose a mechanism with price discrimination. If we pigeonhole all the styles of price updating above as **regular tâtonnement**, we can contrast this with **augmented tâtonnement**, where each bidder possibly pays a different price for a time slot:

$$\lambda_{imk}^{l+1} = \max \left\{ 0, \lambda_{mk}^{l+1} + f\left(\gamma_{mk}^{l}, t(i)\right) \right\}. \tag{3.4}$$

In this case, λ_{mk}^{l+1} is the original price in round $l+1$ as computed by an adaptive price update or similar methods and $t(i)$ is some aspect or function of job i or its schedule. For example, this can be a function depending on the resource usage, i.e. time slot usage, of a job, or a predetermined job profile.

3.3 Solving the Lagrangian Dual with Combinatorial Auctions

As the deliberations in the previous chapter look quite akin to the explanations in the section 2.2, it is easy to guess that we want to link the concept of Combinatorial Auctions to integer programming and Lagrangian Relaxation through the similarities between subgradient search and price updating mechanisms.

3.3.1 Decomposition into subproblems

To connect these two approaches, we first look at the objective function (2.25) and the capacity constraits (2.24) in Section 2.3 and use the Lagrangian relaxation technique form Section 2.2 to include these constraints into our objective function. We get the following formula and will see that we can split the problem into subproblems. This part is due to Hämmerle et al. (2015).

$$\mathcal{Z}_D(\lambda) = \min_{b_{ij},m_{ij}} \sum_{i\in\mathcal{I}} W_i T_i + \sum_{m\in\mathcal{M}} \sum_{k\in\mathcal{K}} \lambda_{mk} \Big[\sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}_i} \delta_{ijmk} - 1 \Big], \tag{3.5}$$

Rearranging the order of summations, we get:

$$\min_{b_{ij},m_{ij}} \sum_{i\in\mathcal{I}} W_i T_i + \sum_{i,j} \sum_{m\in\mathcal{M}} \sum_{k\in\mathcal{K}} \lambda_{mk}\delta_{ijmk} - \sum_{m,k} \lambda_{mk}. \tag{3.6}$$

By using the fact

$$\sum_{m\in\mathcal{M}} \sum_{k\in\mathcal{K}} \lambda_{mk}\delta_{ijmk} = \sum_{k=b_{ij}}^{c_{ij}} \lambda_{m_{ij}k}, \tag{3.7}$$

we can rewrite $\mathcal{Z}_D(\lambda)$ as

$$\mathcal{Z}_D(\lambda) = \min_{b_{ij},m_{ij}} \sum_{i\in\mathcal{I}} W_i T_i + \sum_{i,j} \sum_{k=b_{ij}}^{c_{ij}} \lambda_{m_{ij}k} - \sum_{m,k} \lambda_{mk}. \tag{3.8}$$

The last formulation of $\mathcal{Z}_D(\lambda)$ makes clear that we can divide it into subproblems

$$\mathcal{S}_i = \min_{b_{ij}, m_{ij}} W_i T_i + \sum_{j \in \mathcal{J}_i} \sum_{k=b_{ij}}^{c_{ij}} \lambda_{m_{ij} k}. \tag{3.9}$$

Each subproblem $\mathcal{S}_1$ is a one-job scheduling problem with linear precedence constraints as in (2.23) and processing time constraints (2.18). As stated in Wang et al. (1997), the one job scheduling problem with linear precedence constraints is not $\mathcal{NP}$-hard. The problem can be pictured as a minimization problem, where each operation of a job has to be scheduled on one out of a set of alternative machines. The objective thereby is to minimize the sum of the weighted tardiness and the cost of machine time usage where λ_{mk} is the cost for machine k at time slot k. Using the subproblems, we can write

$$\mathcal{Z}_D(\lambda) = \sum_{i \in \mathcal{I}} \mathcal{S}_i - \sum_{m,k} \lambda_{mk}. \tag{3.10}$$

As in Section 2.2, we can formulate the Lagrangian dual problem of our original problem, which amounts to

$$\mathcal{Z}_D = \max_{\lambda} \mathcal{Z}_D(\lambda). \tag{3.11}$$

3.3.2 A Combinatorial Auction for the Job Shop Scheduling

The following sections up to and including 3.3.4 are based on the paper by Kutanoglu and Wu (1999). If we design an auction as stated in 3.2.1, a good is a pair of a machine and a time slot. The set of goods, therefore, is $G = \{(m,k) : 0 \leq m \leq M-1, 0 \leq k \leq K-1\}$. Due to the preemption constraints, a bid for one operation B_{ij} has to include P_{ijm} consecutive time slots and is a set of time slots

$$\begin{aligned} B_{ij} \in \{&(m_{ij}, k) : m_{ij} \in \mathcal{H}_{ij}, \\ &1 \leq b_{ij} \leq k \leq c_{ij} \leq K-1, c_{ij} = b_{ij} + P_{ijm} - 1\}. \end{aligned} \tag{3.12}$$

A feasible job bid B_i, is a combination of operation bids $\bigcup_{j \in \mathcal{J}_i} B_{ij}$ but subject to additional precedence constraints. Since the premise is a weighted tardiness problem, we have a due date and, thus, a tardiness T_i for each job. If we additionally assume that a job has to pay a price for each time slot it uses and that tardiness causes some cost, we can formulate the utility function of a bidder in dependency of a bid B_i as

$$U_i(B_i) = -W_i T_i - C(B_i). \tag{3.13}$$

Here W_i is the cost incurred by a unit of tardiness and $C(B_i)$ is the cost of time slots contained in bid B_i, called the **payment function**. Hence a job agent searches for a bid that maximizes its utility function, trying to find the middle ground between keeping the due date and keeping cost for time slots low. If the payment function is defined as the sum of prices of time slots given as a vector of Lagrangian multipliers λ, then $C(B_i) = \sum_{k=b_{ij}}^{c_{ij}} \lambda_{m_{ij}k}$. The utility function can be rewritten as

$$U_i(B_i) = -W_i T_i - \sum_{j \in \mathcal{J}_i} \sum_{k=b_{ij}}^{c_{ij}} \lambda_{m_{ij}k}. \tag{3.14}$$

The constraints above are the same as the ones in Section 2.3 and the utility function is the negative of the job specific subproblem $\mathcal{S}_i$. Minimizing the subproblem is, therefore, equivalent to maximizing the utility function and a bidder bidding optimally and truthfully will solve the job-specific subproblem.

The auctioneer, on the other hand, tries to maximize his revenue. As he does not know the valuations of the bidders, he can only use his current knowledge to achieve this. When he has collected the bids, he can compute the overall demand. Given this demand, he tries to maximize his earnings by adjusting prices. This means for solutions to subproblems $\mathcal{S}_i$, the auctioneer tries to find the resource prices to reach his goal. This formulation describes the Lagrangian dual

$$\mathcal{Z}_D = \max_{\lambda} \mathcal{Z}_D(\lambda). \tag{3.15}$$

3.3.3 The Subgradient Search in Combinatorial Auctions

From Section 2.2 we know that if we can solve (3.5) optimally for a given vector of Lagrange multipliers λ, we can use the subgradient search to solve the Lagrangian dual (3.11). It is not hard to see that the rule to generate a sequence of Lagrange multiplier in the subgradient search with similar notation to Section 2.2

$$\lambda^{l+1} = \lambda^l + s_l \left(Ax^l - b \right) \tag{3.16}$$

is a variant of

$$\lambda^{l+1} = \max \left\{ 0, \lambda^l + f\left(\gamma^l\right) \right\}. \tag{3.17}$$

In this equation γ^l is the vector of excess demand as defined in (3.1a) and we now designate the iteration in the superscript. The advantage of this change of notation is

due to the fact that the subscript now denotes the time slot and the machine. Since

$$\Big(Ax^{\star l} - b\Big) = \Big(\sum_{(i,j)} \delta^{\star l}_{ijmk} - 1\Big)_{mk} = \gamma^l \tag{3.18}$$

provided that both $x^{\star l}$ and $\delta^{\star l}_{ijmk}$ are optimal solutions of the Lagrangian Relaxation. And $f\left(\gamma^l\right)$ is chosen as in the subgradient search as $s_l\gamma$ with the step size set to

$$s^l = \alpha^l \frac{\mathcal{Z}^* - \mathcal{Z}_D(\lambda^l)}{\|\gamma^l\|^2}. \tag{3.19}$$

Once again α^l is a scalar satisfying $0 < \alpha^l < 2$, and $\mathcal{Z}^*$ is an upper bound of $\mathcal{Z}_D$. Fisher (2004)

All together, this means that with the information about Integer Programming and Combinatorial auction, we can use the latter to simulate a subgradient search if we set a job specific objective function as (3.9). We expect the job agent to solve this problem optimally and to bid accordingly, meaning to communicate the result of its one job scheduling problem. As, since the subproblems are independent, minimizing them separately yields the same result as minimizing (3.10) and the adaptive price updating mechanism simulates a gradient search using the settings derived above.

3.3.4 Augmented Price Update

As shown in the papers by Jennergren (1973) and Jose et al. (1997), regular tâtonnement is often not powerful enough to coordinate multiple agent represented by linear programs and a quadratic term in the payment function can resolve this problem. Since our job agents are characterized by linear programs, we will put this insight to use. Thus, we will take the same course of action as Kutanoglu and Wu (1999) and formulate a subproblem as

$$\begin{aligned}\mathcal{S}_i &= \min_{b_{ij},m_{ij}} W_iT_i + \Big(\sum_{m\in\mathcal{M}}\sum_{k\in\mathcal{K}} \lambda_{mk} + \sum_{m\in\mathcal{M}}\sum_{k\in\mathcal{K}} q\delta_{ijmk}\Big)\delta_{ijmk} \\ &= \min_{b_{ij},m_{ij}} W_iT_i + \sum_{m\in\mathcal{M}}\sum_{k\in\mathcal{K}} \lambda_{mk}\delta_{ijmk} + \sum_{m\in\mathcal{M}}\sum_{k\in\mathcal{K}} q\delta^2_{ijmk}.\end{aligned} \tag{3.20}$$

However since $\delta^2_{ijmk} = \delta_{ijmk}$, the quadratic term joins the linear term and this reformu-

lation does not yield a new subproblem.

$$\begin{aligned}\mathcal{S}_i &= \min_{b_{ij},m_{ij}} W_i T_i + \sum_{m\in\mathcal{M}}\sum_{k\in\mathcal{K}} \lambda_{mk}\delta_{ijmk} + \sum_{m\in\mathcal{M}}\sum_{k\in\mathcal{K}} q\delta_{ijmk} \\ &= \min_{b_{ij},m_{ij}} W_i T_i + \sum_{m\in\mathcal{M}}\sum_{k\in\mathcal{K}} (\lambda_{mk}+q)\delta_{ijmk}.\end{aligned} \tag{3.21}$$

We can approach this issue, as it was done in Kutanoglu and Wu (1999), and join multiple time slots together to create time zones. We can keep the linear term in the price function unchanged and add a quadratic term to the payment function. Using the formulation from (3.9), we define the subproblem

$$\mathcal{S}_i = \min_{b_{ij},m_{ij}} W_i T_i + \sum_{j\in\mathcal{J}_i}\sum_{k=b_{ij}}^{c_{ij}} \lambda_{m_{ij}k} + \mu \sum_{j\in\mathcal{J}_i}\sum_{t=0}^{T-1}\Big(\sum_{t=k_t}^{k_t+\tau-1}\delta_{ijmk}\Big)^2. \tag{3.22}$$

Here we assumed $\mathcal{K}$ partitioned into $\mathcal{T} = \{0, 1, \dots T-1\}$ time zones, each of length τ. The first time slot in time zone t is denoted by k_t. When not grouping time slots into time zones but grouping them by operation, we come up with

$$\mathcal{S}_i = \min_{b_{ij},m_{ij}} W_i T_i + \sum_{j\in\mathcal{J}_i}\sum_{k=b_{ij}}^{c_{ij}} \lambda_{m_{ij}k} + \mu \sum_{j\in\mathcal{J}_i}\Big(\sum_{k=b_{ij}}^{c_{ij}}\delta_{ijmk}\Big)^2. \tag{3.23}$$

The sum Z_M of theses job specific problems $\mathcal{S}_i$ subtracted by the prices for time slots $\sum_{m,k}\lambda_{mk}$, as in (3.10), does not necessarily satisfy the inequality $Z_M \leq Z$ as in (2.5) since we have added a quadratic term.

However, if we denote as $m_{ij}*$, the machine which maximizes the process time of operation (i,j) and additionally subtract

$$\mu \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}_i}\big(\lfloor\frac{P_{ijm_{ij}*}}{\tau}\rfloor\big)^2 + (P_{ijm_{ij}*} \mod \tau)^2 =: K(\mu,\tau) \tag{3.24}$$

and since

$$K(\mu,\tau) \geq \mu \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}_i}\sum_{t=0}^{T-1}\Big(\sum_{t=k_t}^{k_t+\tau-1}\delta_{ijmk}\Big)^2 \tag{3.25}$$

we again get the lower bound identity $Z_K = \sum_{i\in\mathcal{I}}\mathcal{S}_i - \sum_{m,k}\lambda_{mk} - K(\mu,\tau) \leq Z$, when the subproblems $\mathcal{S}_i$ are as defined in (3.22).

3.3.5 Third Order Pricing

Above, we introduced the concept of price discrimination. In this section we extend this notion to what was called **third order pricing** in Bikhchandani and Ostroy (2002). For example, if one produces bidder specific prices, the subproblems that we constructed in the last paragraphs appear the following:

$$\mathcal{S}_i = \min_{b_{ij}, m_{ij}} W_i T_i + \sum_{j \in \mathcal{J}_i} \sum_{k=b_{ij}}^{c_{ij}} \lambda_{i m_{ij} k}. \tag{3.26}$$

In the right summand, the payment function, an i is added because we now have personalized prices. When formulating the entire minimization problem like in (3.10), one is confronted with the problem of finding appropriate Lagrange multipliers that still lead to a lower bound for the original, unrelaxed problem. This is central to the usefulness of the Lagrangian relaxation technique. If we sum over all subproblems, we get

$$\sum_{i \in \mathcal{I}} \mathcal{S}_i = \sum_{i \in \mathcal{I}} W_i T_i + \sum_{i,j} \sum_{k=b_{ij}}^{c_{ij}} \lambda_{i m_{ij} k} \tag{3.27}$$

And can formulate a integer program similar to the program (P) in the style of statement (3.7) in Section 3.3.1.

$$\mathcal{Z}_M = \min_{b_{ij}, m_{ij}} \sum_{i \in \mathcal{I}} W_i T_i + \sum_{i,j} \sum_{k=b_{ij}}^{c_{ij}} \lambda_{i m_{ij} k} - M(\lambda) \tag{3.28}$$

subject to the same constraints as the program (LR). If we now choose an appropriate summation to subtract in place of $M(\lambda)$, analogous to $\sum_{m,k} \lambda_{mk}$ in (3.6), we can deduce the lower bound property of our new program.

Proposition 1. *Let (AP) be the integer problem as stated in* (3.28)*, then the inequality*

$$Z_M \leq Z \tag{3.29}$$

holds for each of the following choices for $M(\lambda)$:

$$M(\lambda)_1 = \sum_{k\in\mathcal{K}} \sum_{m\in\mathcal{M}} \max_i \lambda_{imk}, \tag{3.30}$$

$$M(\lambda)_2 = \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} \max_m \lambda_{imk}, \tag{3.31}$$

$$M(\lambda)_3 = \sum_{k\in\mathcal{K}} |\mathcal{M}| \max_{i,m} \lambda_{imk}, \tag{3.32}$$

$$M(\lambda)_4 = \sum_{k\in\mathcal{K}} |\mathcal{I}| \max_{i,m} \lambda_{imk}. \tag{3.33}$$

Proof. Let δ^*_{ijmk} be variables corresponding to an optimal solution to the original problem (2.25) from Section 2.3 satisfying all constraints. First we can reformulate

$$Z_M = \min_{b_{ij},m_{ij}} \sum_{i\in\mathcal{I}} W_i T_i + \sum_{i,j} \sum_{k=b_{ij}}^{c_{ij}} \lambda_{im_{ij}k} - M(\lambda) \tag{3.34}$$

to

$$Z_M = \min_{b_{ij},m_{ij}} \sum_{i\in\mathcal{I}} W_i T_i + \sum_{i,j} \sum_{m\in\mathcal{M}} \sum_{k\in\mathcal{K}} \lambda_{imk}\delta_{ijmk} - M(\lambda) \tag{3.35}$$

due to the fact

$$\sum_{m\in\mathcal{M}} \sum_{k\in\mathcal{K}} \lambda_{imk}\delta_{ijmk} = \sum_{k=b_{ij}}^{c_{ij}} \lambda_{im_{ij}k} \tag{3.36}$$

and analogous to (3.7) form Section 3.3.1. Since $\delta^*_{ijmk} \in \{0,1\}$ and, for fixed m,k, takes the value 1 for at most one job i, we use the property

$$\sum_{k\in\mathcal{K}} \sum_{m\in\mathcal{M}} \max_i \lambda_{imk} \geq \sum_{i,j} \sum_{m\in\mathcal{M}} \sum_{k\in\mathcal{K}} \lambda_{imk}\delta^*_{ijmk} \tag{3.37}$$

to obtain the desired inequality

$$\begin{aligned} \mathcal{Z}_{M_1} \leq & \min_{b_{ij},m_{ij}} \sum_{i\in\mathcal{I}} W_i T_i \\ & + \sum_{i,j} \sum_{m\in\mathcal{M}} \sum_{k\in\mathcal{K}} \lambda_{imk}\delta^*_{ijmk} - \sum_{k\in\mathcal{K}} \sum_{m\in\mathcal{M}} \max_i \lambda_{imk} \leq \mathcal{Z}. \end{aligned} \tag{3.38}$$

The choice $M(\lambda)_2$ satisfies the identity

$$\sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} \max_m \lambda_{imk} \geq \sum_{i,j} \sum_{m\in\mathcal{M}} \sum_{k\in\mathcal{K}} \lambda_{imk}\delta^*_{ijmk}, \tag{3.39}$$

as, analogously to above, for a job i $\delta^*_{ijmk} = 1$ for at most one m and we attain that $Z_{M_2} \leq Z$. The third and fourth possibility for $M(\lambda)$ satisfy the inequalities

$$\sum_{k\in\mathcal{K}} |\mathcal{M}| \max_{i,m} \lambda_{imk} \geq \sum_{i,j} \sum_{m\in\mathcal{M}} \sum_{k\in\mathcal{K}} \lambda_{imk}\delta^*_{ijmk}, \tag{3.40}$$

$$\sum_{k\in\mathcal{K}} |\mathcal{I}| \max_{i,m} \lambda_{imk} \geq \sum_{i,j} \sum_{m\in\mathcal{M}} \sum_{k\in\mathcal{K}} \lambda_{imk}\delta^*_{ijmk}, \tag{3.41}$$

since, in a feasible solution, at most $\mathcal{I}$ as well as at most $\mathcal{M}$ jobs can be scheduled at a given time k. These two inequalities subsequently lead to analogous inequations as (3.38) for each choice, i.e. $Z_{M_3} \leq Z$ and $_{Z_{M_4}\leq Z}$. Furthermore, we can see that

$$\sum_{k\in\mathcal{K}} \sum_{m\in\mathcal{M}} \max_i \lambda_{imk} \leq \sum_{k\in\mathcal{K}} |\mathcal{M}| \max_{i,m} \lambda_{imk}. \tag{3.42}$$

This is based on the fact that, if we fix k, then $\sum_m \max_i \lambda_{imk} \leq |\mathcal{M}| \max_{i,m} \lambda_{imk}$, because there are $|\mathcal{M}|$ summands on the left side of the inequality and for every m' on the left $\max_i \lambda_{im'k} \leq \max_{i,m} \lambda_{imk}$. This gives us $Z_{M_3} \leq Z_{M_1}$. The same reasoning with jobs and machines interchanged provides us with the fact $Z_{M_4} \leq Z_{M_2}$. □

Without price discrimination, the choice $M(\lambda)_1$ leads to

$$\sum_{k\in\mathcal{K}} \sum_{m\in\mathcal{M}} \max_i \lambda_{imk} = \sum_{k\in\mathcal{K}} \sum_{m\in\mathcal{M}} \lambda_{mk}, \tag{3.43}$$

as $\lambda_{mk} = \lambda_{lmk} = \lambda_{imk} \forall i, j \in \mathcal{I}$ and accordingly

$$\max_i \lambda_{imk} = \lambda_{mk} \forall i \in \mathcal{I}. \tag{3.44}$$

The same reason leads to

$$\sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} \max_m \lambda_{imk} = \sum_{k\in\mathcal{K}} |\mathcal{I}| \max_{i,m} \lambda_{imk} = \sum_{k\in\mathcal{K}} |\mathcal{I}| \max_m \lambda_{mk} \tag{3.45}$$

with choices $M(\lambda)_2$ and $M(\lambda)_4$. As well as for the third choice

$$\sum_{k\in\mathcal{K}} |\mathcal{M}| \max_{i,m} \lambda_{imk} = \sum_{k\in\mathcal{K}} |\mathcal{M}| \max_m \lambda_{mk}. \tag{3.46}$$

Chapter 4

Implementation

In this chapter we will transform the mathematical problem formulation into an algorithmic problem. We will state the algorithm in pseudo-code to give a better understanding of the procedure. Also, information regarding the implemented programs will be given and all input and output variables explained.

The following is the design of a combinatorial auction to help solve the flexible job shop scheduling with transport time problem. This design is, among others, based on the paper by Kutanoglu and Wu (1999). It is used to obtain an infeasible schedule which, by means of list scheduling, will be transformed into a feasible schedule. This infeasible schedule determines the lagrangian multipliers for each time slot on each machine.

4.0.1 Initialization

Since we have proved that we can split our problem into job specific subproblems in the previous chapters 3.3.1, the first step in the Initialization is to divide the fjsstt problem into its job subproblems. At this point, the method of price adjustment is chosen and also which form of price discrimination is used.

4.1 Two Different Subgradient Methods

4.1.1 Simple Subgradient Search

The method of price adjustment implemented is the adaptive update whose step size s_l varies between rounds and equals a subgradient search. This formulation is taken from Fisher (2004) and is, amongst others, treated in Held, Wolfe, and Crowder (1974).

With similar notation as in previous chapters with l denoting the current iteration and according to (3.16) in Section 3.3.3, the price update follows

$$f\left(\gamma_{mk}^{l}\right) = s^{l}\gamma_{mk}^{l}, \tag{4.1}$$

with s^l defined as,

$$s^{l} = \alpha^{l}\frac{\mathcal{Z}^{*} - \mathcal{Z}_{D}(\lambda^{l})}{\|\gamma^{l}\|^{2}}. \tag{4.2}$$

$\mathcal{Z}^*$ is and upper bound for the value of $\mathcal{Z}$ and the Lagrangian multipliers are updated by

$$\lambda_{mk}^{l} = \lambda_{mk}^{l-1} + f\left(\gamma mk^{l}\right). \tag{4.3}$$

Algorithm 1 shows how the implemented program is structured.

4.1.2 Surrogate Subgradient Search

The following implementation is based on Bragin, Luh, Yan, Yu, and Stern (2014). In order for the multipliers λ^l to converge to λ^*, the parameter α^l as in equation (2.10) in Section 2.2.2 is computed by

$$\alpha^{l} = 1 - \frac{1}{Ml^{p}},\ p = 1 - \frac{1}{l^{r}} \text{ with } M \leq 1, 0 < r < 1, k = 2, 3, \ldots \tag{4.4}$$

The stepsize s^l is updated according to the formula

$$s^{l} = \alpha^{l}\frac{s^{l-1} \parallel Ax^{l-1} - b \parallel}{\parallel Ax^{l} - b \parallel^{2}}. \tag{4.5}$$

The initial step size can either be chosen beforehand and input at the start of the algorithm, or it is calculated by

$$s^{0} = \frac{\mathcal{Z}_{est} - \mathcal{Z}_{SG}(x^{0}, \lambda^{0})}{\parallel Ax^{0} - b \parallel^{2}}, \tag{4.6}$$

where $\mathcal{Z}_{est}$ is an estimation of the optimal dual cost $\mathcal{Z}_{LR}^{*}$. Updating of the Lagrange multipliers follows the equations (4.1) and (4.3).

Data: fjsstt-problem, α, σ, ϕ, number of iterations it

begin
  Initialization Split problem into subproblems as in (3.9).
  The Lagrangian multipliers λ_0 are initialized equalling 0.
  for $i = 0$ **to** it **do**
    Bid generation
    Merge solutions
    Calculate subgradients
    Calculate lower bound
    if *New lower bound is better* **then**
      Update lower bound
    end
    if *Lower bound hasn't increasied in* σ *iterations* **then**
      Update α
    end
    Update Lagrangian multipliers
    if ϕ *iterations passed since the last feasibility repair* **then**
      Call feasibility repair
      if *New upper bound is better* **then**
        Update upper bound
      end
    end
  end
end

Algorithm 1: Algorithm for the simple subgradient search

Data: fjsstt-problem, r, M, initial step size s, number of subproblems n optimized each iteration, ϕ, number of iterations it

```
begin
    Initialization Split problem into subproblem as in (3.9).
    The Lagrangian multipliers λ^l are initialized equalling 0.
    If not set as an input, the initial step size is computed as in (4.6)
    for i = 0 to it do
        Bid generation
        Merge solutions
        Calculate subgradients
        Calculate lower bound
        if New lower bound is better then
            Update lower bound
        end
        if Lower bound hasn't increased in σ iterations then
            Update α
        end
        Update step size
        Update Lagrangian multipliers
        if φ iterations passed since the last feasibility repair then
            Call feasibility repair
            if New upper bound is better then
                Update upper bound
            end
        end
    end
end
```

Algorithm 2: Algorithm for the surrogate subgradient search

This choice of variables differs from the simple subgradient search since we want to ensure that the multipliers λ^l converge to the value λ^*, even if we only optimize a share of the job specific subproblems. This possibility will be useful when solving big problems because it helps minimizing the computational requirements. The algorithm for the surrogate subgradient search is shown in Algorithm 2.

4.2 Three Kinds of Price Discrimination and The Auction Algorithm

4.2.1 Augmented Price Update

One kind of price discrimination that was implemented is based on the paper by Kutanoglu and Wu (1999). It is called augmented price update or augmented tatonnement to that effect and was already treated in 3.3.4. In this case, the job specific subproblem is formulated as

$$\mathcal{S}_i = \min_{b_{ij}, m_{ij}} W_i T_i + \sum_{j \in \mathcal{J}_i} \sum_{k=b_{ij}}^{c_{ij}} \lambda_{m_{ij}k} + \mu \sum_{j \in \mathcal{J}_i} \sum_{t=0}^{T-1} \Big(\sum_{t=k_t}^{k_t+\tau-1} \delta_{ijmk} \Big)^2. \tag{4.7}$$

4.2.2 Based on Preceding Iterations

From an economic point of view, the auctioneer will always try to maximize his or her revenue. Assuming that a job agent, who requires a specific time slot on a machine more urgently, is willing to pay a higher price for this slot, it is important for the auctioneer to identify these agents and slots. One way to achieve this is by looking back at preceding iterations. Since a job agent in dire need of a specific slot will bid on it in several consecutive auction iterations, the auctioneer could post discriminatory prices on slots by raising prices for job agents who have bidden on a slot for a number of previous rounds. Another positive consequence of this process could be that jobs will post different solutions to their problems since requiring the same time slot each iteration will raise their prices. Since feasibility restoration uses the infeasible solution generated by merging the individual job solutions, greater diversity in infeasible schedules could mean that more different feasible schedules are found.

The price for a time slot for a job agent, with a factor $\rho < 1$, would then be

$$\begin{aligned} \lambda_{imk}^{l+1'} =& \lambda_{mk}^{l+1} + \rho \delta_{ijmk}^{l} + \rho^2 \delta_{ijmk}^{l-1} + \rho^3 \delta_{ijmk}^{l-2} \dots \\ =& \lambda_{mk}^{l+1} + \sum_{t=0}^{\iota-1} \rho^{l-t} \delta_{ijmk}^{l-t} \end{aligned} \tag{4.8}$$

The basic price update λ_{imk}^{l+1} is done in the way of (4.3). The magnitude of the effect of the price discrimination is dependent on the value of ρ and the number ι of preceding

iteration taken into account. An individual job problem would then take the form

$$\mathcal{S}_i = \min_{b_{ij}, m_{ij}} W_i T_i + \sum_{m \in \mathcal{M}} \sum_{k \in \mathcal{K}} \lambda_{imk} \delta_{ijmk}. \tag{4.9}$$

4.2.3 Based on Tardiness and Weight

Following the same train of thought as delineated above one can argue that an agent with a pressing due date and a large job weight is willing to pay a higher price for his time slots. It can, therefore, be wise to discriminate agents based on due date and weight. This leads to following prices

$$\lambda_{imk}^{l+1'} = \lambda_{mk}^{l+1} + f(D_i, W_i). \tag{4.10}$$

λ_{mk}^{l+1} is computed as in (4.3). The function f needs to depend on $P_i = \frac{W_i}{D_i}$ since P_i is highest for urgent due dates and great weight and smallest for late due dates and small weights. If we want to normalize this rating in the interval $(0, 1)$, then

$$f_i = \frac{\frac{D_i}{W_i}}{\max_{i \in \mathcal{I}} \frac{D_i}{W_i}} \tag{4.11}$$

will do the trick. As the price function for a slot, we will assume

$$\lambda_{imk}^{l+1'} = \lambda_{mk}^{l+1} + f_i \kappa \lambda_{mk}^{l+1}, \tag{4.12}$$

and the job specific subproblem is again

$$\mathcal{S}_i = \min_{b_{ij}, m_{ij}} W_i T_i + \sum_{m \in \mathcal{M}} \sum_{k \in \mathcal{K}} \lambda_{imk} \delta_{ijmk}. \tag{4.13}$$

4.2.4 The Auction Algorithm

Since the auction algorithm relies on either the simple subgradient search or the surrogate subgradient search, it is very similar to the algorithms above. The only difference lies in the sections 'Calculate lower bound' and 'Update Lagrange multipliers'. When using the price discrimination based on weighted tardiness or on preceding iterations, calculating the lower bound follows the same formula as in Proposition 1 and updating the Lagrange multipliers is done as in either (4.12) or (4.8). If the augmented price update is chosen, the

initialization of the subproblems uses the formula (4.7) and the lower bound is computed by using the result of Section 4.2.1.

4.3 Feasibility Repair

In general, we cannot expect to obtain a feasible solution by applying a combinatorial auction. This means that we need a tool to translate the infeasible schedule generated by a combinatorial auction into a feasible one. The method of list scheduling offers itself for such an endeavor. The algorithm described below is based on the paper by Hoitomt et al. (1993).

4.3.1 Input

The infeasible schedule is represented by two matrices, in each of these matrices a line represents a job and each column an operation. The first matrix entries are the beginning times b_{ij} of the operations (i, j), the seconds are machine assignments m_{ij} for operation (i, j).

4.3.2 Initialization

Prioritizing Function

In order to modify the infeasible schedule and to resolve scheduling conflicts we need to know which operations to keep at the time slots specified in the infeasible schedule and which operations to push back in the schedule. Therefore, we need a notion of importance of an operation. To determine this importance, we make use of prioritizing rules. Possible choices are

1. $f(i,j) = W((T_i + 1)^2 - T_i^2)$ or
2. $f(i,j,t) = \frac{W_i}{P_i} e^{\max\{0, D_i - P_i - t\}/KP}$

if delaying the beginning time of operation (i, j) by one time unit leads to an increased weighted tardiness. The first choice favours jobs which already incur tardiness and stems from Hoitomt et al. (1993). The second option is the apparent tardiness cost (ATC) as presented in Vepsalainen and Morton (1987) and which can also be found in Buchmeister (2013).

In the statement regarding the apparent tardiness cost, P_i is the remaining processing time of job i, P is the average remaining process time of all jobs both at time t and

K is a scaling factor. Since the ATC function as stated above is a dynamic scheduling rule and the value of $f(i,j,t)$ is computed anew each time a machine becomes unused, it is, in this form, not suitable for a static feasibility repair. For our purpose, we change the ATC function so that the value of $f(i,j,t)$ for operation (i,j) is computed with t equal to the beginning time obtained from the infeasible schedule. P_i is, therefore, the remaining process time of job i from and including operation j. P is computed as the average remaining process time of all jobs and of operations starting at time t or later.

The List U of all Operations

We use one of the prioritizing functions from above to produce a sequence U of all operations according to the following rules: The operation (i,j) is placed before the operation (u,v) in U, if

- $b_{ij} < b_{uv}$, or if
- $b_{ij} = b_{uv}$ and $f(i,j) > f(u,v)$, or if
- $b_{ij} = b_{uv}$ and $f(i,j) = f(u,v)$ through random selection.

Algorithm 3 shows the pseudo code for the feasibility repair. An extended version of the list scheduling feasibility repair was also implemented, its pseudo code can be seen in Algorithm 4. The difference between these two variants is that the basic version schedules the operation on the first free machine while the extended version searches through all machines an selects the machine, where the completion time is minimal. This means that the basic list scheduling deviates less from the input infeasible schedule than the extended version.

4.4 The Implemented Programs

For my thesis and during my time at Profactor, I implemented a program to solve an fjsstt-problem with the techniques described above. Not every implementation of algorithms mentioned here is my own work, for example, the simple subgradient search as well as the surrogate subgradient search where implemented by my colleagues during my time at Profactor. My task, on the other hand, was implementing the auction algorithm using the two subgradient searches as foundations. My other programming project was the feasibility restoration for which the above-mentioned list scheduling method was chosen. While employing list scheduling without the auction algorithm is possible, and it was used to compute an initial upper bound, this kind of usage is not its intend and the solutions obtained are comparatively bad. Hence, from a users point of view, the main

```
Data: fjsstt-problem, infeasible solution, priority rule f, scaling factor K
begin
    Initialization Generate list U according to f and K
    while U ≠ ∅ do
        Get first operation from U
        while UnsearchedMachines ≠ ∅ do
            while !NeedAnotherMachine do
                Compute earliest possible starting time
                Compute latest possible starting time
                Check if machine is available within these bounds
                if Machine is available then
                    The operation is scheduled on this machine
                    Set NeedAnotherMachine = true
                    Set UnsearchedMachines = ∅
                else
                    Set NeedAnotherMachine = true
            end
            if UnsearchedMachines ≠ ∅ then
                Delete previous machine from UnsearchedMachines
                if UnsearchedMachines = ∅ then
                    Push back current operation back in U
                else
                    Select another suitable machine
            end
        end
    end
end
```

Algorithm 3: List scheduling

```
Data: fjsstt-problem, infeasible solution, priority rule f, scaling factor K
begin
    Initialization Generate list U according to f and K
    while U ≠ ∅ do
        Get first operation from U
        for m ∈ SuitableMachines do
            Compute earliest available starting time
            Compute earliest available completion time t_m
            Add t_m to the list of completion times T_m
        end
        Take the minimum of T_m and schedule the operation accordingly
        Delete the operation from U
    end
end
```

Algorithm 4: Extended list scheduling

matter in utilizing my program is the auction algorithm. Since the two subgradient algorithms are essential for the auction to work and are the basis for the auction algorithm using the auction means using and configuring the subgradient searches.

4.4.1 Using the Program

This section will familiarize the reader with the use of my program. It is intended to give only an overview and not a in depth treatment of specific parts of the algorithm. An example of initializing a problem and solving it with the program at hand is given in the Appendix in the section A.1.

Input

Since fjsstt-problems require a lot of input information, this information is provided through three plain text files. Each of these files has the same name, which is that of the problem with different file extensions. The *.properties* file gives a summary of configurations as well as information concerning the two other files. The second file, with a *.fjs* extension, contains all information for all jobs for example the number of operations for each job as well as the suitable machines and corresponding processing times for all operations. The file with the extension *.tt* stores the data of the transport times between machines.

To access the information from these three files, a parsing package was implemented. This program was already running when I started working on my thesis and its inner workings are not of interest to this documentation. For the user, it is enough to know that the method `parseFile(fileName)`, which takes a string as input, initializes a problem given in the above format.

Constructing an entirely new problem is possible. This is either done by writing the three files above or by initializing the problem in Java directly. While this is reasonable if one wants to check the proper operation of different parts of the algorithm, it is quite tedious for big problems. If one wants to initialize a problem manually the constructor `FJSSTT_problem()` has to be used.

```
FJSSTT_problem(noJobs, noOperations, noMaxOperations,
noMachines, noTimeSlots, suitableMachines, processTimes, travelTimes,
dueDates, objective, jobWeights)
```

In order to have enough information for the algorithm to work before this constructor is called, the following data has to be provided:

- The *objective*: The aspect to be minimized. This can be either the total weighted

completion time, or, as was the main focus of this thesis, the total weighted tardiness.

- The number of *jobs*: A positive integer.
- The number of *operations* per job: An array of positive integers, one for each job.
- The *maximum* number of operations per job: A positive integer.
- The number of *machines*: A positive integer.
- The number of *time slots*: A positive integer.
- The set of *suitable machines* for each operation: A `HashMap`, where the key is the integer tuple (job, operation) and the corresponding value is the set of machines.
- The *process times* for all operations: A three dimensional array of positive integers, where the first index is the job, the second the operation and the third the machine.
- The *travel times* between machines: A two dimensional array of positive integers, each index denoting a machine.
- The *due dates* of jobs: An array of positive integers, one for each job.
- The *weights* of jobs: An array of positive integers, one for each job.

The `setmTimeSlots()` method can be used to alter the number of time slots after the initialization of the problem, which is useful when the initial value was too low and produced an error.

List Scheduling

To use the list scheduling, it has to be initialized with the `ListScheduling()` constructor, which comes in three variants.

- `ListScheduling(fjsstt-problem)`
- `ListScheduling(fjsstt-problem, solution)`
- `ListScheduling(fjsstt-problem, solution, outputFile)`

The first just needs an fjsstt-problem as input, the second also an infeasible solution, which serves as a basis for the feasibility repair, the third has a file name as additional input. In this file the results will be printed and if it is not initialized, as in the to other constructors, a default file is used as output.

Once the list scheduling is initialized, the feasibility restoration is started with either `listScheduling(prioritizingFunction,valueOfK)` or `listSchedulingExtended(prioritizingFunction,valueOfK)`. Both these methods have two inputs or none at all. These are the kinds of prioritizing function to be used and, in the case of the ATC function, also the value of the variable K in (4.3.2), which can be any real number. If no input is give,n the default configuration, which is the ATC function with K equalling 5, will be used.

Simple Subgradient Search

The simple subgradient search must first be initialized through a constructor. This constructor

`SimpleSubgradientSearch(fjsstt-problem,` α`,`
`itsUntilAlphaIsHalved, itsBetweenFeasibilityRepair)`

has as inputs, first the fjsstt-problem to be solved, second an initial upper bound for this problem, third the value of α and fourth the number of iterations without an increase of the lower bound until α is halved, as well as the number of iterations between feasibility repair. The roles of the upper bound and α are shown in the equation (4.2).

While α can be any real number, the three other values have to be integers. If no initial upper bound is given or its value is negative, the list scheduling method is used to generate a suitable one. In this case, a random infeasible schedule is generated and used as a basis for list scheduling. If the number of iterations between list scheduling runs is initialized with 0, no feasibility restoration will be applied except after the subgradient search has run its course. If it is initialized with -1 no feasibility restoration will be done whatsoever.

Once the simple subgradient search is initialized, it is started with the `solve(noIterations)` method which takes the number of iterations that are performed as an input. This has to be a positive integer.

Surrogate Subgradient Search

Similar to the simple subgradient search, the surrogate search is initialized with the `SurrogateSubgradientSearch()` constructor.

`SurrogateSubgradientSearch(fjsstt-problem, fixedStepSize,`
`estimatedDualCost, initialStepSize, numberOfSubproblems,`
`r, M, itsBetweenFeasibilityRepair)`

The inputs are the following: First a fjsstt-problem, second a boolean dictating whether a fixed step size should be used or, if initialized as false, should be computed by the

equation (4.2). Furthermore an estimate for the dual cost, the value of the initial step size, which should be a postive real number and the number of subproblems solved in each iteration indicated by a positive integer. Next come values for the variables r and M from the equation (4.4), where r is some positive real number and M a positive integer. The last input is the number of iteration between feasibility repair with the same restrictions as above.

Like the simple subgradient search, the surrogate variety is started with the `solve(noIterations)` method, which again needs an integer indicating the number of iterations to be executed.

Auction

The three alternative auctions, each named for its pricing mechanism, are initialized with an individual constructor. These three constructors are:

- `AugmentedPriceUpdate(fjsstt-problem, subgradientSearch, scalingFactorMu, timeZoneLengthTau)`
- `DemandDependentDiscrimination(fjsstt-problem, subgradientSearch, scalingFactorRho, precedingIterationsIota)`
- `WeightedTardinessDsicrimination(fjsstt-problem, subgradientSearch, scalingFactor)`

The first two inputs are the same for each constructor. The first is a fjsstt-problem while the second is a kind of subgradient search. This can either be a simple or a surrogate subgradient search. This means that before the constructor is usable, a subgradient search has to be initialized. The input `scalingFactorMu` in the first constructor corresponds to the μ in (4.7) and should be a positive real number smaller than 1 and `timeZoneLength` to the variable τ, a positive integer.

In the second constructor, `scalingFactorRho` initializes the value of ρ, a with ρ being a positive real number, as in (4.8). `precedingIterationsIota` sets the value for ι in the same equation, which denotes the number of iterations the auctioneer looks back to compute prices for time slots and which has to be a positive integer. In this spirit, the value 1 will not increase this number since the auctioneer already calculates the prices based on the previous iteration.

The last constructor only has `scalingFactorKappa` as additional input, this corresponds to the value of κ in the equation (4.12) and has to be a positive real number.

Once any of these auctions is initialized, the algorithm is again started with `solve(noIterations)` with the number of iterations to be executed as the sole input.

Chapter 5

Numerical Results

To provide the means to evaluate the quality of the above implemented algorithms, I have applied these to a data-set of problem instances and analysed the performance of the algorithm in the light of different indicators. A grid search was performed for each algorithm on each problem instance. The first section describes the problem instances used. In the second section the problems encountered when implementing the algorithms are stated. The third part of this chapter is a summarized assessment of the program and algorithms. Finally a conclusion is drawn from these findings.

5.1 Problem Types and Problem Instances

The problem instances, which were used to configure and test the algorithms I programmed, were all taken from the website Moench (2016). From these problem instances six were used: WT1, WT2, Mk01 loose, Mk01 tight, Mk02 loose, and Mk02 tight. The difference between the problem types loose and tight of the same Mk-instance lies in the due dates. The due dates of tight instances are at a sooner time point, thus increasing the total weighted tardiness.

In order to put the outcome of my algorithm into perspective, the best results up to now were provided together with the problem data. What was not part of the data was a matrix of travel times between machines. Therefore, I generated these transport times myself by first determining the maximum of all minimal makespans of all jobs. This means that I calculated the minimal makespan of each job and took the maximum of all values. This is a lower bound for the completion time of the problem instance. I further split the machines into two groups with a difference in size of at most 1. I assumed a travel time between machines in two different groups of one time the maximum of all minimal makespans. This problem instance is indicated by the suffix "A", or twice that value, with the suffix "B". The thought process behind this assumption was that

the machines are situated in two different plants. The travel time between machines in the same plant can be neglected, while there is substantial travel time between plants. With these settings I had 18 different problems, the six original ones and two derivations thereof with different travel times.

Let us now take a closer look at the basic problem instances without travel times I used. The following table gives an overview. I indicates the number of jobs, M the number of different machines, J_i the number of operations per job and $\max|\mathcal{H}_{ij}|$ the maximum number of different machines suitable for an operation. The column with header D shows the minimum and maximum of due dates of jobs and the column TWT the best know total weighted tardiness. The first two instances originally stem from Brandimarte (1993).

Table 5.1: `FJSS` problem instances

Instance	I	M	J_i	$\max\|\mathcal{H}_{ij}\|$	D	TWT
WT1	10	5	5	3	45 ... 76	57
WT2	20	5	5	3	58 ... 156	252
Mk01 loose	10	6	5 ... 7	3	16 ... 36	55
Mk01 tight	10	6	5 ... 7	3	5 ... 12	393
Mk02 loose	10	6	5 ... 7	6	18 ... 28	18
Mk02 tight	10	6	5 ... 7	6	6 ... 9	361

This next table shows the travel times for problem instances of types A and B, as well as the grouping of machines in the last column. "$3+2$" means that the first 3 machines form a group and so do the remaining 2.

Table 5.2: Travel Times for `FJSSTT` problem instances

Instance	Type A	Type B	Grouping
WT1	36	72	3 + 2
WT2	38	76	3 + 2
Mk01	22	44	3 + 3
Mk02	18	36	3 + 3

5.2 Problems Encountered

The next section deals with my experience while working on my programs. Even though a course in programming is mandatory in the second semester of studies of mathematics

at the university of Vienna, I had never worked on a programming problem of this size. When I started to write my program many algorithm were already implemented. For example the simple subgradient search, the basis for the auction mechanism, was already running.

The first algorithm I implemented was the list scheduling method for feasibility repair. Although I already had a good understanding of how the algorithm works and which parts perform specific tasks and I had an idea of how to implement the single steps of the algorithm, structuring and bringing it in a concise form was challenging. After writing the program, a long time of debugging began. Although I was introduced to the debugging tools at an early stage, I used simple output messages to the console to solve most of my problems. Only problems I could not tackle this way were further examined using the debugging tool. All in all, I encountered many problems throughout implementation, but none stood out in a way to be considered major in comparison to others.

The only problem worth mentioning, although not strictly one within the code, was the case when too few time slots were considered to obtain a feasible schedule through list scheduling. Most of the time the list scheduling routine is called during execution of a subgradient search and the number of time slots is initialized along with this search. At first, I thought it would be easy to increase the number of time slots during list scheduling, as to not interrupt the algorithm and avoid the need to restart the whole program. This course of action proved unreasonable, since changing the time slots for list scheduling would also change the setting of the subgradient search between iterations. This seemed both mathematically dubious as well as unnecessary complex to implement.

5.3 Results and Configuration

The presentation of results will be split into different parts. One classification is in terms of subgradient searches applied. Therefore, dividing the findings into simple subgradient search and surrogate subgradient search is executed. Another approach to split the evaluation of the algorithms is by lower and upper bound which is sensible, because the lower bound property stems from the subgradient search and the upper bound is produced by the list scheduling method. A third distinction is made by looking at the impact of price discrimination on both the lower and the upper bound.

An entirely different matter is the configuration of the list scheduling routine and comparison of two variants, which will be dealt with at the end of this section. During these test the initial upper bound for the objective value needed by the subgradient searches was provided by applying the list scheduling method to a randomly generated infeasible schedule. For all tests mentioned in this section the number of iterations was fixed at 800. The following section will give a rough overview of how the results where established but not the whole chain of thought and reasoning behind reaching these conclusions. We

will first start with examining the lower bounds achieved by our subgradient searches and then look at the effects of price discrimination. Afterwards we will take a look at the upper bounds.

5.3.1 Lower Bounds without Price Discrimination

Simple Subgradient Search

All the subsequent figures portray, unless stated otherwise, the results of test runs for one problem only. The results were very similar for all problems. Since putting all problem instances into one figure would have made the graphics too crowded and including 18 similar figures would have been heavily redundant, the shown visualizations should exemplify the findings. In the case that one problem instance offered unique results it will be mentioned and treated in written form. The first part of our investigation into achieving the best possible lower bounds will take a look at the simple subgradient search. During these test runs no feasibility restoration was performed and attention was turned solely on the lower bound. The two remaining variables to examine were the value of α and the number of iterations until α was halved when no better lower bound was found. For these trials α was varied in the interval $]0, 2]$ and the second variable was tested within the interval $[2, 32]$.

Figure 5.1 shows the value of α on the x axis, and the number of iterations until α is halved is located on the y axis. A bigger circle implies a higher lower bound. We can see that the bigger the value of both α and the iterations until halving α are, the greater the lower bound gets. While this conclusion is apparent from this figure, the difference between the achieved lower bounds is rather small.

The same tendency can easily be seen in Figure 5.2a, where again a tendency to higher upper bounds is seen as the iterations until α is halved increase. In this figure each of the six vertical facets corresponds to the number of iterations until α is halved, the x axis indicates the lower bounds and the subdivision of the facets shows the values of α. Each dot in the figure codes one test run each with a different value for the two control variables, and the vertical lines are the arithmetic means for each facet. Although an increase of lower bounds can be observed, it is of minimal degree and the averages of the 5 lower facets differ in no more than 0.5 in value. While this figure suggests that changing α as seldom as possible is the way to go, numerical test on other problems show no improvement after the number of these iterations reaches 25.

The third figure 5.2b is the complement to the second with the roles of α and iterations until halving α interchanged. This means that each facet corresponds to a value of α and the subdivision of facets is by values of iterations until α was halved. As before each dot depicts one test run, each with different initial setting and the grey lines indicate the mean in every facet. This figure shows that once the value of α exceeds 1 there is

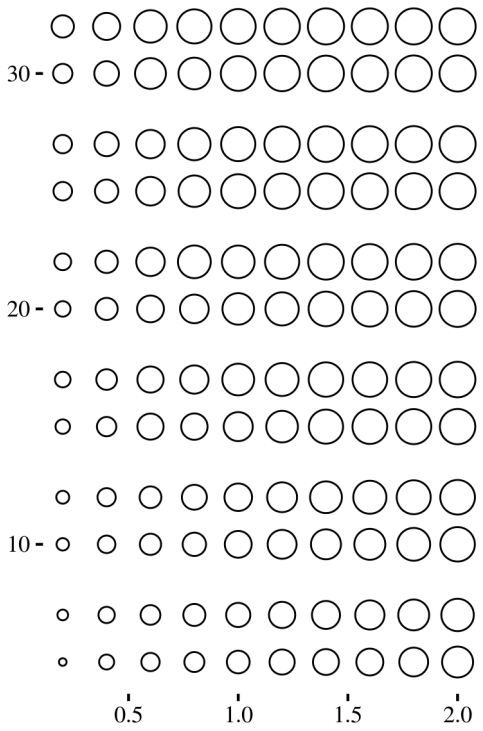
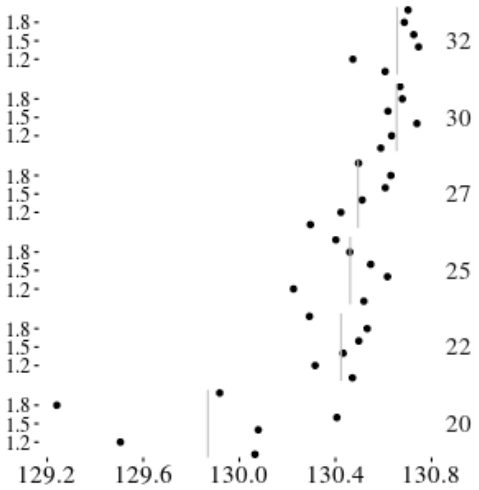


Figure 5.1: The best lower bounds for Mk02 loose, where a bigger circle means that a higher lower bound was achieved.

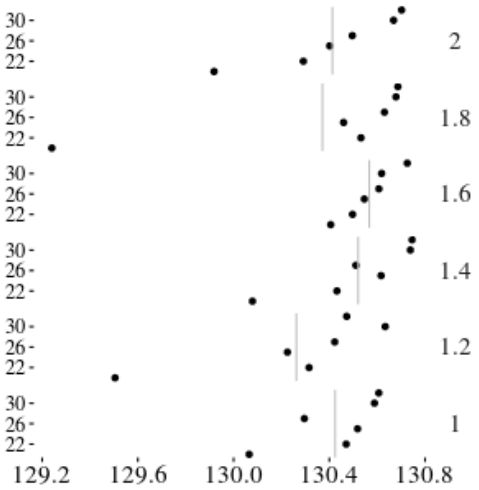
no trend indicating a superior configuration. From these three figuress one can conclude that our simple subgradient search provides the best lower bounds when the value of α lies in the interval $[1, 2]$ and the number of iterations until α is halved is larger than 25.

Surrogate Subgradient Search

Let us turn our attention to the surrogate subgradient search and the lower bounds it provides. The first insight concerning the surrogate subgradient search is that to fulfill the requirement of equation 2.15, it is necessary to optimize enough subproblems in one iteration. Only when optimizing at least 5 subproblems in a single iteration the surrogate subgradient search satisfies the lower bound condition. The parameters in the surrogate subgradient search are r, M, the initial step size and the number of subproblems to be optimized in one iteration. The bounds for r where set as 0.2 and 1, values considered for M were between 16 and 24, at least 5 and at most 9 subproblems were solved in one iteration. This last bound was chosen so that less than half of the subproblems of the largest instance WT2 are solved in one iteration. The initial step size was examined



(a) Facets to the right correspond to iterations until α is halved and scaling on the left side indicates the values of α.



(b) Facets to the right correspond to values of α whereas the values to the left show the number of iterations until α is halved.

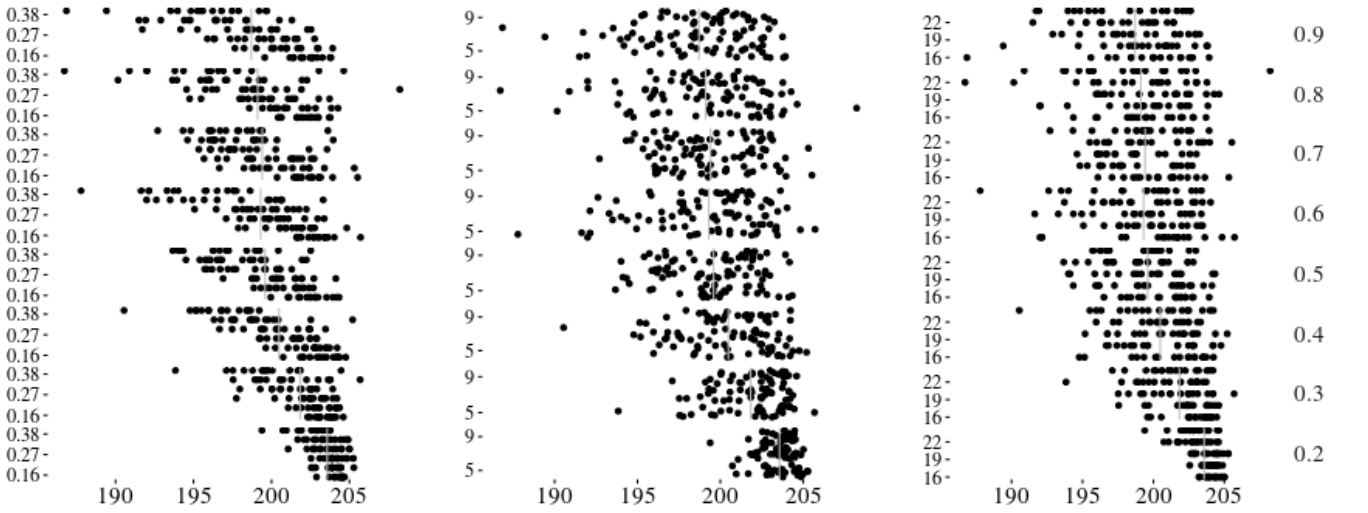
Figure 5.2: The best lower bounds for WT2 with respect to the values of both α and the number of iterations until halving α.

between 0.1 and 0.4.

The first three figures 5.3 are again faceted and each facet, as indicated on the far right, corresponds to a value of r. The subdivisions are, from left to right, the initial step size, the number of subproblems solved in an iteration and the values of M.

It is evident when looking at these three figures that the best choice for r is smaller that 0.3. We can restrict the value of r and will henceforth only include test runs adhering to this restriction into our considerations. Lets now turn to the best value for the variable M. We will again show three figures where the facets indicate the values of M and the subdivision correspond to the other three variables.

When looking at these three figures one finds that the best choice for M is the value 16. Although the maxima of lower bounds for different values of M are very similar, the minimal lower bounds, and therefore also the mean value, achieved are higher when the value 16 is chosen for M. As before we will from now on exclude test runs with other values for M which will ease our tasks substantially.

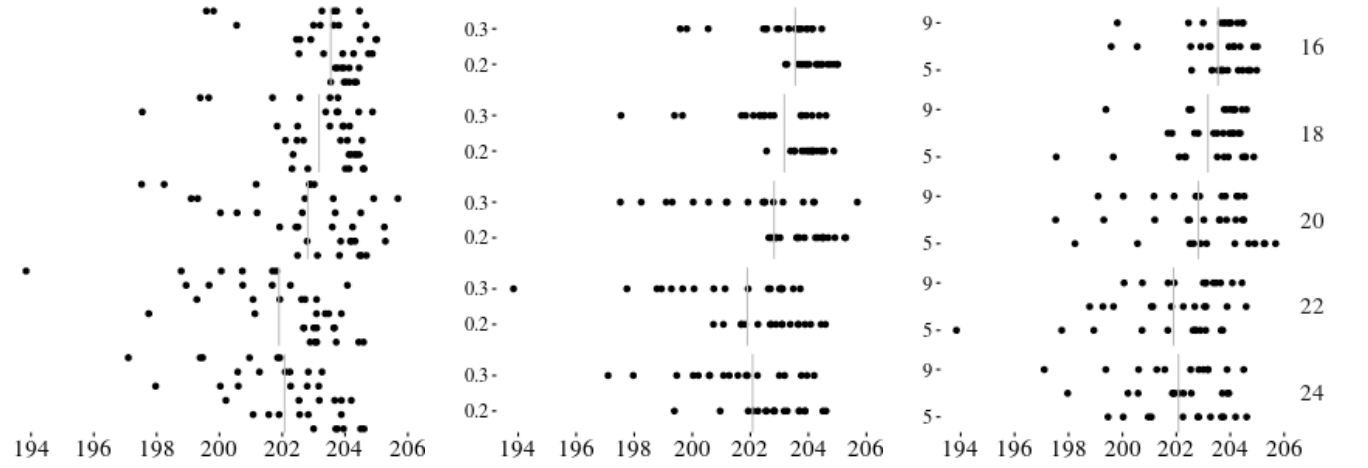


(a) subscales on the left indicate the initial step size

(b) subscales on the left indicate the number of subproblems solved each iteration

(c) subscales on the left show the values of M

Figure 5.3: The lower bounds achieved by the surrogate subgradient search for the problem instance Mk01 loose A. Each figure is faceted by values of r which are indicated to the far right. The sub scales at the left side of each graphic show values of a different variable.



(a) subscales on the left show the values of the initial stepsize

(b) subscales on the left denote values of r

(c) subscales on the left indicate the number of subproblems solved in each iteration

Figure 5.4: The lower bounds achieved by the surrogate subgradient search for the problem instance Mk01 loose A. The facets indicated to the far right show the values of M. Each graphic's subdivision corresponds to a different variable.

The last figure concerning the lower bounds in the surrogate subgradient search looks at the contribution of the initial step size. Although there seems to be substantial difference between the results for the initial step size equalling 0.2 and 0.25 investigation in results from other problems indicate that the outcomes for any initial step size smaller or equal 0.25 and larger than 1.5 are similar. For larger values a drop can be observed and therefore this limitation seems sensible for the variable of the initial step size.

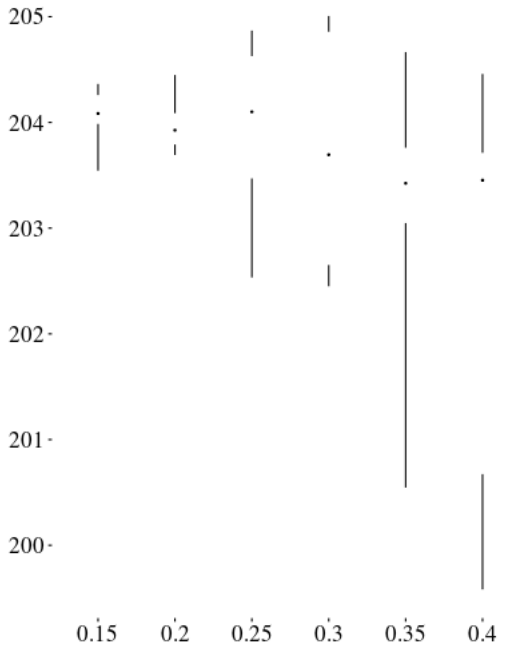


Figure 5.5: The lower bounds achieved by the surrogate subgradient graphic for the problem instance Mk01 loose A with the initial stepsize indicated on the x axis. In this boxplot the values of the initial step size are mapped to the x axis whereas the lower bounds achieved correspond to the y axis. The point in the middle symbolizes the median of the values and the lines run from the maximum and the minimum to the upper respectively lower quartile.

In conclusion, the best configuration for the surrogate subgradient search satisfies the following constraints

$$r \leq 0.3, \quad M = 16, \quad \text{initial step size} \leq 0.25, \tag{5.1}$$

while solving at least 5 subproblems in each iteration.

5.3.2 Lower Bounds with Price Discrimination

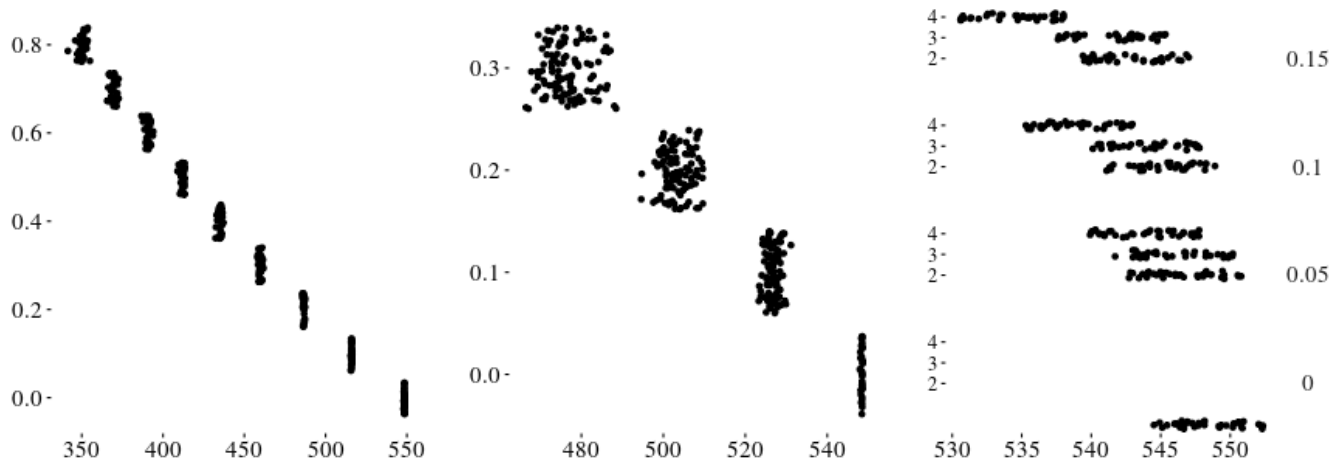
Once the underlying subgradient searches were configured, the next step was to test the effects of price discrimination on the lower bounds. The results of these test runs with price discrimination, which were performed for each of the aforementioned discrimination strategies for chapters 3.3.4 and 3.3.5, were sobering. A large problem seemed to be that the estimation to retain the property of achieving a lower bound from Theorem 1 was imprecise.

The following figures Figure 5.6 and Figure 5.7 show that the lower bounds deteriorate with increasing scaling factors for each of the methods of price discrimination in conjunction with both simple and surrogate subgradient search. The three figures from the first group pertain to the simple subgradient search. In each, the value of the achieved lower bound is recorded on the x axis and each point corresponds to a test run with different settings.

The left figure indicates the impact of the weighted tardiness discrimination from 4.2.3 on the search and the y axis plots the values of the factor $\frac{1}{\kappa}$. One immediately sees that the bigger the contribution of the price discrimination is the poorer the lower bound gets. The figure in the middle depicts the effect of the demand dependent price update from 4.2.2. The y axis shows values of the factor ρ and we see that its increase leads to worse lower bounds. Finally the right figure shows the ramification of the augmented price update as in 4.2.1 and the y axis shows the number of time slots joint to one time zone and the facetting to the right shows the values of μ. Again the same conclusion as above can be drawn.

The Figure 5.7 conveys the same information as the above but with regard to the surrogate subgradient search. The order of the pricing methods is the same as above and the meaning of each axis and its divisions are equal. The deterioration of the lower bounds is not as visible as in the case of the simple subgradient search. Especially when looking at the figure for the demand dependent discrimination in the middle, the effect is very little.

All in all, the demand dependent discrimination had varying influence on the surrogate subgradient search. While there are problem instances in which an improvement of the lower bounds is visible, other instances show worsening of the results and a third group of problem instances indicate a scattering of lower bounds. Therefore, it is not easy to come to a final conclusion concerning the co-action of the surrogate subgradient search and the demand dependent discrimination.

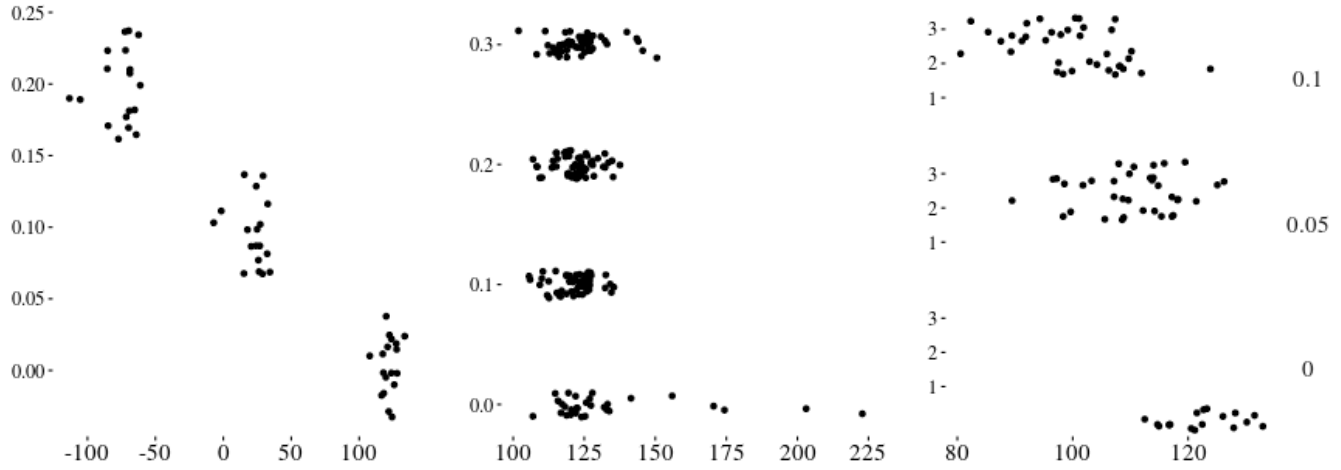


(a) Effect of weighted tardiness discrimination. The y axis shows the values of $\frac{1}{\kappa}$.

(b) Effect of demand dependent discrimination. The scale on the y axis denotes the values of ρ.

(c) Effect of augmented price update. The facets on the right correspond to the values of μ and the subscales on the left indicate the number of time slots joined to form a time zone.

Figure 5.6: The effects of three different strategies for price discrimination on lower bounds of the simple subgradient search for problem instance WT1B.



(a) Effect of weighted tardiness discrimination. The y axis shows the values of $\frac{1}{\kappa}$.

(b) Effect of demand dependent discrimination. The scale on the y axis denotes the values of ρ.

(c) Effect of augmented price update. The facets on the right correspond to values of μ and the subscales on the left show the number of time slots joined to form a time zone.

Figure 5.7: The effects of three different strategies for price discrimination on lower bounds of the surrogate subgradient search for problem instance WT1B.

5.3.3 Upper Bounds

Next to the quest for a good lower bound for a specific problem, the best feasible schedule produced is of even more importance as a benchmark for algorithmic performance. Thus, we will first present two tables which will show the best solutions the algorithm found for each problem. One for the simple subgradient search Table 5.3 and one for the surrogate subgradient search Table 5.4. Test runs were made with the parameters within the bounds obtained from the test runs for the lower bounds and with all price discrimination methods.

At first glance it might seem as if the price discrimination strategies had a positive effect on the subgradient searches. For example the augmented price update in conjunction with the surrogate subgradient seem to work together quite well. Yet, when investigated more thoroughly it seemed that there was no consistent effect from any form of price discrimination on the subgradient searches. While they helped produce some of the best feasible solutions they did not contribute in a consistent or predictable manner. There was no visible positive effect on the median feasible schedule, nor the average nor the worst case scenario. On the other hand, there was also no negative implication on any of these parameters. Another remarkable insight gained from this table is that it was easy to reach the best feasible schedule in the problem instance Mk02 tightB. Adding extensive travel times to the problem seems to restrict the job agents in their choices and almost always forces the algorithm to a very good result.

One could ask the question why there was no visible significant impact, neither positive nor negative on the performance of the basic algorithm. The first possibility is that there is no such impact and the price discrimination just contributes in a minor, mostly chaotic way. The second possibility is that the implementation of the pricing mechanisms was inadequate or that other price mechanisms would have been a better choice. The augmented price update is the price discrimination with the most solid basis in pure mathematics and one can therefore hope that the results together with the surrogate subgradient search could point out that an improvement could be facilitated with the help of this method. But it seems that the result from these test runs do not give a conclusive answer to the question of usefulness of augmented price update.

Another insight in reference to the best feasible schedule obtained was that not only was there no visible relation between values of α, the number of iterations until α was halved, there was also no relation between test runs which achieved high lower bounds and those which produces favourable feasible solutions.

As one can see in the following figure 5.8 in which the lower bound of a test run is indicated on the x axis and the total weighted tardiness of the best feasible solution is shown on the y axis. Additionally one can see in 5.9a and 5.9b that the choice of both α and the number of iterations until α is halved is in no visible correlation to the ratio of the best feasible schedule and the best lower bound.

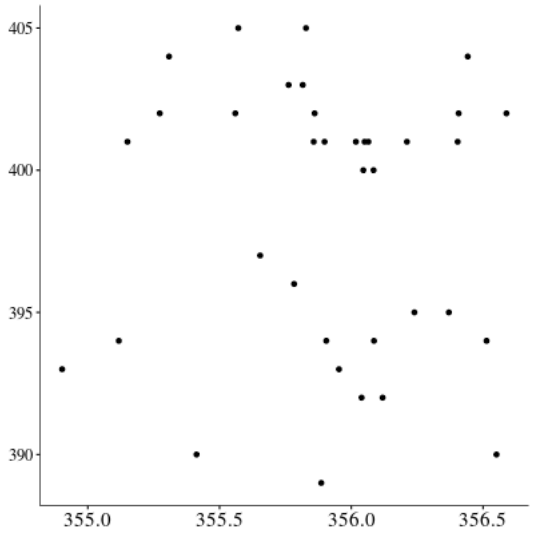
Table 5.3: The best results obtained by the simple subgradient search. The column 'Moenchs Best Schedule' shows the weighted tardiness of the best solution previously known and is taken from the website Moench (2016). The column 'Occurence' indicates how many test runs produced a schedule with the weighted tardiness from column 'Best Feasible Solution'. For each problem instance 360 test runs were perfomed. The last four columns display the same information as the column Occurence with regard to the kinds of discrimination.

Instance	Moenchs Best Schedule	Best Feasible Solution	Best Lower Bound	Occurrence	Price Discrimination			
					no Discrimination	DDD	WTD	APU
WT1	57	54	43,9408	9			3	6
WT1A		291	258,0356	80	36	42		2
WT1B		583	545,4973	3	1		2	
WT2	252	259	130,6117	1		1		
WT2A		1349	1177,4989	1		1		
WT2B		2408	2305,4489	1		1		
Mk01 loose	55	47	40,4944	28	3		12	13
Mk01 looseA		226	205,2234	2		1		1
Mk01 looseB		314	304,2023	24	1	1	11	11
Mk01 tight	393	386	356,5888	1			1	
Mk01 tightA		608	574,1061	33	4	11	8	10
Mk01 tightB		701	672,2655	1		1		
Mk02 loose	18	13	4,6806	1			1	
Mk02 looseA		185	176,9332	10			8	2
Mk02 looseB		366	352,6809	11			11	
Mk02 tight	361	343	318,6809	1				1
Mk02 tightA		581	572,0986	28				28
Mk02 tightB		763	749,9488	311	36	24	107	144

Table 5.4: The best results achieved with the surrogate subgradient search. The column 'Moenchs Best Schedule' shows the weighted tardiness of the best solution previously known and is taken from the website Moench (2016). The column 'Occurence' indicates how many test runs produced a schedule with the weighted tardiness from column 'Best Feasible Solution'. For each problem instance 200 test runs were perfomed. The last four columns display the same information as the column 'Occurence' with regard to the kinds of discrimination.

Instance	Moenchs Best Schedule	Best Feasible Solution	Best Lower Bound	Occurrence	Price Discrimination			
					no Discrimination	DDD	WTD	APU
WT1	57	59	42,635	1				1
WT1A		291	272,5864	140	18	32	68	22
WT1B		580	548,2526	1				1
WT2	252	271	133,5641	6		4	2	
WT2A		1384	1172,3936	2			2	
WT2B		2423	2305,0701	1	1			
Mk01 loose	55	50	39,7682	8	1	2	4	1
Mk01 looseA		225	204,4785	1				1
Mk01 looseB		314	304,1741	15		2	4	9
Mk01 tight	393	388	355,4702	1	1			
Mk01 tightA		608	573,5346	48	4	8	16	20
Mk01 tightB		701	673, 1979	1				1
Mk02 loose	18	20	4,2356	3	1			2
Mk02 looseA		185	175,9072	5				5
Mk02 looseB		366	352,4512	49	2	20	10	17
Mk02 tight	361	344	318,0011	1	1			
Mk02 tightA		581	571, 4261	3				3
Mk02 tightB		763	749,7659	170	16	30	60	64

Figure 5.8: The relationship between good lower bounds indicated on the x axis and good upper bounds on the y axis. Each point depicts the results of a test run on the problem Mk02 tight.



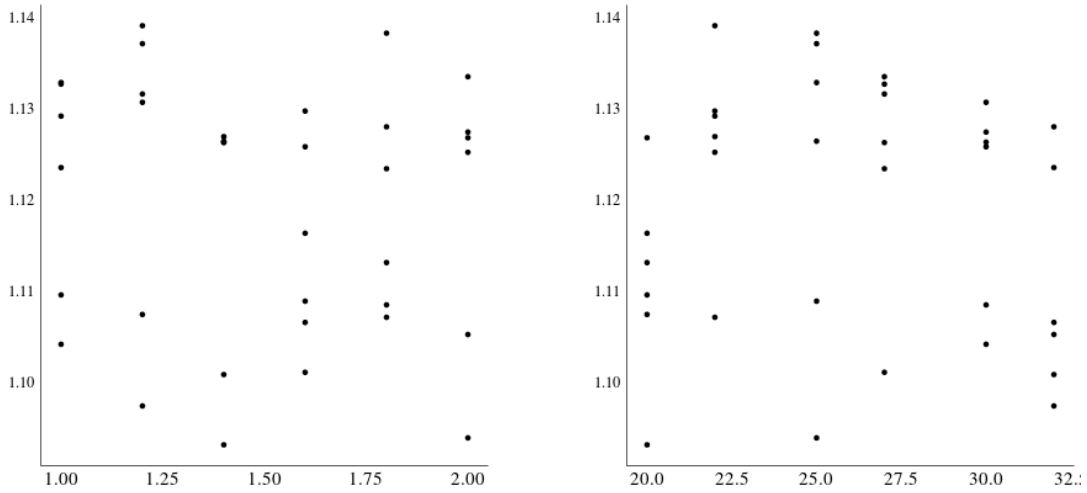
There has been no connection between different values for parameters in both the simple subgradient search and the surrogate subgradient search and the feasible schedule produced. Since this behaviour is rather unsatisfying another way of examining the algorithm was conducted. Meta optimization was carried out in order to get the best values for the input variables.

This was done by using the algorithm Snobift from Huyer and Neumaier (2008). The results of this test runs confirmed the findings from the grid search. Looking at Figure 5.10 no clear connection between the values of variables and the performance of the algorithm can be found. In this figure each point corresponds to the results of the meta optimization pertaining to one problem instance. The choice of variables that secured the best output are shown here. The different shapes indicate different types of instances.

5.3.4 List Scheduling

Two different kinds of list scheduling were tested. The basic version and the extended variant, in which all machines are searched prior to scheduling an operation on a machine. The ordinary list scheduling method first searches the machine suggested by the infeasible schedule and assigns the operation to this machine if the slots are free. In this case no other machines are searched.

Comparing these two methods the results show that, when the initial upper bound needed for the simple subgradient search is provided by the list scheduling method and not as



(a) The values of α are shown on the x axis and the ratio of the best feasible schedule and the best lower bound is indicated on the y axis.

(b) The number of iterations until α is halved are indicated on the x axis and the ratio of the best feasible schedule and the best lower bound is displayed on the y axis.

Figure 5.9: The ratio of the best feasible schedule and the best lower bound with respect to the values of both α and the number of iterations until halving α.

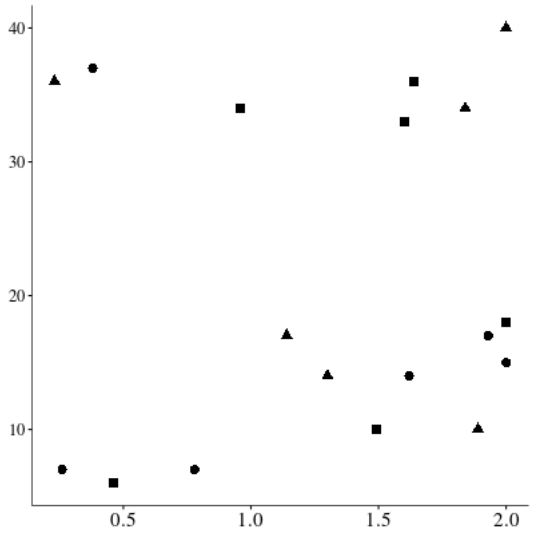
an input from the user, it is better to use the extended version. While the performance in view of best upper bound generated was comparable between these different methods, the extended version is a major improvement over the basic version in terms of worst attained upper bound, median and average attained upper bound. Since this initial list scheduling was performed on a randomly generated infeasible schedule, it seems that the basic version is more dependent on this infeasible schedule than the more intricate search. This connection was also visible when the feasibility restoration was performed during iterations of the subgradient searches, in which case the basic version produced better results. Although not better by a great margin it is still consistently better in terms of both best, worst, average and median feasible schedule.

Concerning the two different kinds of priority functions and the scaling factor in the ATC function, the ATC function with a scaling factor of about 5 proved to be best suitable, albeit the difference was minor.

5.4 Conclusion

Looking at the results from this chapter both positive and negative conclusions can be drawn. Each of the subgradient searches produced good lower bounds and a relation between parameters and the results can be identified. It can also be seen that the

Figure 5.10: The results of the meta optimization using the Snobfit algorithm. The values for α are indicated on the x axis and the numbers of iterations until α is halved are shown on the y axis. Each point corresponds to the choice of variables producing the best output for a single problem instance. The dots are basic problem instances, the triangles belong to the group A with moderate travel times and the squares to problems belonging to the group B.



methods for price discrimination do not improve the performance of the algorithms, with the exception of the demand dependent discrimination in conjunction with the surrogate subgradient search. In this case the effect is ambiguous and further testing could clarify the situation. Whether the worsening of the lower bounds is due to the calculation of the lower bound or a deteriorating effect of the price discrimination is hard to say.

Regarding upper bounds the outcome is conflicting. On the one hand the total weighted tardiness of the feasible schedules is comparable to previous results and sometimes even better. On the other hand a relation between parameters and the quality of the results can not be seen. It seems that the feasibility repair is to disruptive and changes to much to leave any connection between infeasible and feasible schedule. This may also be the reason why the effect of price discrimination on the total weighted tardiness can not be determined. Thus, the jury is still out on the usefulness of price discrimination regarding feasible schedules. Further research might bring an expedient method of price discrimination to light or proof the uselessness of price discrimination strategies.

Appendix A

Appendix

A.1 Example of a Problem Initialization

This section depicts the code for initializing a home made fjsstt-problem with commentary. The following problem is rather small, as the initialization of a problem within java is quite cumbersome.

```
/**
* This code generates a test instance with 3 jobs,
* 3 machines, 2 operations per job with moderate flexibility.
* It also initializes the process times of each operation,
* the matrix of travel times between machines as well as
* the due dates and weights for each job.
*/

public void testInstanceE() {

//This line engenders the HashMap that maps operations represented
//as strings to machines which are encoded as integers.
final HashMap<String, List<Integer» altMachines = new HashMap<String, List<Integer»();


//The following code generates an ArrayList of the type Integer for
//every operation.  These list will be filled with integer corresponding
//to suitable machines for this operation.  The String "altM01" stands
//for alternative machines for operation 1 of job 0.
final List<Integer> altM00 = new ArrayList<Integer>();
final List<Integer> altM01 = new ArrayList<Integer>();
final List<Integer> altM10 = new ArrayList<Integer>();
final List<Integer> altM11 = new ArrayList<Integer>();
```

```
final List<Integer> altM20 = new ArrayList<Integer>();
final List<Integer> altM21 = new ArrayList<Integer>();


//We now add the suitable machines to the operations of job 0 and
//operation 0 can be processed on machine 0 and machine 1, while
//operation 1 can only be scheduled on machine 2.
altM00.add(0);
altM00.add(1);
altM01.add(2);


//Analogously to above we now initialize the alternative machines
//for job 1.  One can see that operation 1 of this job can be
//processed on any of the three machines.
altM10.add(1);
altM10.add(2);
altM11.add(0);
altM11.add(1);
altM11.add(2);


//Contrary to the job above, this one is not flexible at all and
//every operation has only one suitable machine.
altM20.add(2);
altM21.add(0);


//Now we generate the keys for the HashMaps, where k21 is the key
//for operation 1 of job 2.
final String k00 = 0 + "-" + 0;
final String k01 = 0 + "-" + 1;
final String k10 = 1 + "-" + 0;
final String k11 = 1 + "-" + 1;
final String k20 = 2 + "-" + 0;
final String k21 = 2 + "-" + 1;


//Here the keys for the HashMap and the ArrayList of the type Integer
//representing operations and suitable machines respectively are //conjoined.
altMachines.put(k00, altM00);
altMachines.put(k01, altM01);
altMachines.put(k10, altM10);
altMachines.put(k11, altM11);
altMachines.put(k20, altM20);
```

```
altMachines.put(k21, altM21);


//We initialize the matrix of process times.  The first index indicates
//the job, the second the operation and the third the machine.
final int[][][] processTimes = new int[3][2][3];


//The third line implies that if operation 1 of job 2 is scheduled
//on machine 2, it takes 3 time slots to be processed.
processTimes[0][0][0] = 2;
processTimes[0][0][1] = 1;
processTimes[0][1][2] = 3;


processTimes[1][0][1] = 2;
processTimes[1][0][2] = 4;
processTimes[1][1][0] = 2;
processTimes[1][1][1] = 2;
processTimes[1][1][2] = 2;


processTimes[2][0][2] = 3;
processTimes[2][1][0] = 2;


//The matrix of travel times is subsequently generated.  Each index
//corresponds to a machine.  In this example the travel time between
//machines 0 and 1 and machine 2 amount to 2 time units.
final int[][] travelTimes = new int[3][3] {
{0, 0, 2}, {0, 0, 2}, {2, 2, 0}};


//The Array of Integer indicating the operations per job is generated
//here.
final int[] operationsPerJob = {2, 2, 2};


//The due dates for each job is specified, i.e.  job 0
//is due on the 9th time slot.
final int[] jobDueDates = {9, 10, 7};


//The weights for each job are given, i.e.  if job 2 is completed
//at time slot 9, it entails a tardiness penalty of 4.
final int[] jobWeights = {1, 2, 2};
```

```
//Now we use the constructor FJSSTT_ problem to generate the problem
//with the specifications above.  The inputs given directly to the
//constructor are in order of occurrence:  3; the number of jobs,
//2; the maximum number of operations per job, 3; the number of machines,
//30; the number of time slots the algorithm offers as well as the
//objective function, in this case FJSSTT_ problem.Objective.TARDINESS.
//The other possible choice for objective function is FJSSTT_ problem
//.Objective.COMPLETION_ TIME.
final FJSSTT_ problem problem = new FJSSTT_ problem(3, operationsPerJob, 2,
3, 30, altMachines, processTimes, travelTimes, jobDueDates, FJSSTT_ problem.Objective.TARD
jobWeights);


//The next line uses the SimpleSubgradientSearch constructor to produce
//said search.  The inputs are the problem above as well as the upper
//bound, where we insert our conservative guess of 20, our choice
//for α, in the form of a double and for the number of iterations
//with no improvement of the lower bound until α is halved, which
//has to be an integer and we set at 25.  //The variable initialized is the
number of iterations between calls
//to the list scheduling routine.  To achieve the best feasible schedule
//this has to be set at 1.
final SubgradientSearch subgradientSearch = new SimpleSubgradientSearch
(problem, 20, 2, 25, 1);

//If one wants to use the surrogate subgradient search, it can be
//done by replacing the command above by the following.  The inputs
//are again in order of occurrence, the boolean whether a fixed stepsize
//will be used, which was set to true in this example, the estimated
//dual cost of the problem, which we guessed to be 12, the value
//of the initial stepsize, the number of subproblems solved in each
//iteration, the value of the variables r and M as well as the number
//of iterations between calls to the feasibility repair.
final SubgradientSearch subgradientSearch = new SurrogateSubgradientSearch
(problem, true, 12, 0.2, 1, 0.1, 20, 1);

//Now we call the solve() method in the class SimpleSubgradientSearch
//respectivly SurrogateSubgradientSearch and specify that the algorithm
//should run for 800 iterations.
subgradientSearch.solve(800);


//An auction algorithm with price discrimination has to be initialized
```

```
//as follows.  First a subgradient search of your choice is initiazlizead,
//then this is used as an input into the auction algorithm.
final Auction auction = new AugmentedPriceUpdate(problem, subgradientSearch,
0.1, 2);
final Auction auction = new DemandDependentDiscrimination(problem, subgradientSearch,
2, 0.2);
final Auction auction = new WeightedTardinessDiscrimination(problem, subgradientSearch,
0.3);
//For the first kind of auction there are two variables to adjust,
//which are the factor μ and the lenght of the time zones τ.
//For the second type the to variables are again a scaling factor,
//namely ρ and the number of preceding iterations ι to be considered
//in your price discrimination.  The last choice has only one variable
//to adjust, which is the factor κ.  Further information can be found
//in chapter 4.4.1.
}

//Once the auction is generated one again calls the solve() method
//to run algorithm with the number of iterations as an input.
auction.solve(800);

//The last possibility is to only run the list scheduling algorithm.
//This is by first using the constructor ListScheduling().
ListScheduling listScheduling = new ListScheduling(problem);


//To ultimately run the algorithm one can use either the method
//listScheduling() or the method listSchedulingExtended().  Both
//have as inputs the kind of prioritizing function, which offer the
//options of the above mentioned ATC function (4.3.2) as well as
//the quadratic function mentioned ibidem.  These two function
//correspond to the integer values 1 and 2.  In case of the
//ATC function one also has to choose the variable K, which is
//usually selected between 4 and 6.  If the second function is
//chosen, the the second input is without effect and can therefore
//be chosen arbitrarily.
listScheduling.listScheduling(1,5); //or
listScheduling.listSchedulingExtended(2,0);
```

References

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job-shop scheduling. *Management Science*, *34*, 391-401.

Adibi, M., Zandieh, M., & Amiri, M. (2010). Multi-objective scheduling of dynamic job shop using variable neighborhood search. *Expert Systems with Applications*, *37*(1), 282–287.

Akers, S. B. J., & Friedman, J. (1955). A non numerical approach to scheduling problems. *Operations Research*, *3*, 429-442.

Amiri, M., Zandieh, M., Yazdani, M., & Bagheri, A. (2010). A variable neighborhood search algorithm for the flexible job shop scheduling problem. *International Journal of Production Research*, *48*(19), 5671–5689.

Ausubel, L. M., & Cramton, P. (2002). *Demand reduction and inefficiency in multi-unit auctions* (Manuscript). College Park, MD: Department of Economics, University of Maryland.

Balas, E. (1965). Machine scheduling via disjuctive graphs: An implicit enumeration algorithm. *Operations Research*, *17*, 941-957.

Banks, J. S., Ledyard, J. O., & Porter, D. P. (1989). Allocating uncertain and unresponsive resources: An experimental approach. *RAND Journal of Economics*, *20*, 1-25.

Bikhchandani, S., & Huang, C.-F. (1993). The economics of treasury securities markets. *Journal of Economic Perspectives*, *7*, 117-134.

Bikhchandani, S., & Ostroy, J. M. (2002). The package assignment model. *Journal of Economic Theory*, *107*, 377 – 406.

Binsse, H. L. (1887). A short way to keep time and cost. *Transactions of the American Society of Mechanical Engineers*, *9*, 380-386.

Boutilier, C., & Hoos, H. (2001). Bidding languages for combinatorial auctions. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence.*

Bragin, M. A., Luh, P. B., Yan, J. H., Yu, N., & Stern, G. A. (2014). Convergence of the Surrogate Lagrangian Relaxation Method. *Journal of Optimization Theory and Applications*, *164*(1), 173–201.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations research*, *41*(3), 157–183.

Buchmeister, B. (2013). *Advanced Job Shop Scheduling.* Vienna: DAAAM International Publishing.

Caplice, C. G. (1996). *An optimization based bidding process: A new framework for shipper-carrier relationships* (Unpublished doctoral dissertation). Department of Civil and Environmental Engineering, Massachusetts Institute of Technology, Cambridge, MA.

Cook, S. A. (1971). The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium on the Theory of Computing* (p. 151-158). New York.

Dai, B., Chen, H., & Yang, G. (2014). Price-setting based combinatorial auction approach for carrier collaboration with pickup and delivery requests. *Operational Research*, *14*, 361-386.

Dantzig, G. B., & Thapa, M. N. (1997). *Linear Programming 1: Introduction.* New York, Berlin, Heidelberg: Springer-Verlag.

Davenport, A., & Kalagnanam, J. (2002). Price negotiations for procurement of direct inputs. In B. Dietrich & R. V. Vohra (Eds.), *Mathematics of the Internet: E-Auction and Markets* (Vol. 127, p. 27-44). New York: Springer.

de Vries, S., & Vohra, R. V. (2007). Combinatorial auctions: A survey. *INFORMS Journal on Computing*, *15*, 284-309.

Dewan, P., & Joshi, S. (2002). Auction-based distributed scheduling in a dynamic job shop environment. *International Journal of Production Research*, *40*(5), 1173–1191.

Dorndorf, U., & Pesch, E. (1995). Evolution based learning in a job-shop scheduling environment. *Computers and Operations Research*, *22*, 25-40.

Falkenauer, E., & Bouffoix, S. (1991). A genetic algorithm for the job-shop. In *Proceedings of the IEEE International Conference on Robotics and Automation.* Sacramento, California, USA.

Fisher, M. L. (2004). The lagrangian relaxation method for solving integer programming problems. *Management Science*, *50*(12 Supplement), 1861–1871.

Fujishima, Y., Leyton-Brown, K., & Shoham, Y. (1999). Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of International Joint Conference on Artificial Intelligence* (Vol. 1, pp. 548–553).

Gantt, H. L. (1919). *Organizing for Work* (2nd ed.). New York: Harcourt, Brace and Howe.

Gao, J., Sun, L., & Gen, M. (2008). A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computer and Operations Research*, *35*, 2891–2907.

Glover, F. (1989). Tabu search - Part I. *ORSA Journal on Computing*, *1*, 190-206.

Glover, F. (1990). Tabu search - Part II. *ORSA Journal on Computing*, *2*, 4-32.

Grabot, B., & Geneste, L. (1994). Dispatching rules in scheduling: a fuzzy approach. *International Journal of Production Research*, *32*, 903-915.

Grefenstette, J. J. (1987). Incorporating problem specific knowledge into genetic algo-

rithms. In L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing* (p. 42-60). London: Pitman.

Hämmerle, A., Weichhart, G., & Vorderwinkler, M. (2015, August). *Next–generation multi–purpose production systems* (Tech. Rep.). Steyr: Profactor.

Held, M., Wolfe, P., & Crowder, H. P. (1974). Validation of subgradient optimization. *Mathematical Programming*, 62–88.

Herrmann, J. W. (Ed.). (2006). *Handbook of Production Scheduling*. New York: Springer.

Hoitomt, D. J., Luh, P. B., & Pattipati, K. R. (1993). A practical approach to job-shop scheduling problems. *Robotics and Automation, IEEE Transactions on*, *9*(1), 1–13.

Huyer, W., & Neumaier, A. (2008). SNOBFIT - Stable noise optimization by branch and fit. *ACM Transactions on Mathematical Software*, *35*(2).

Jackson, C. (1976). *Technology for spectrum markets* (Unpublished doctoral dissertation). Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, MA.

Jackson, J. (1956). An extension of Johnson's result on job lot scheduling. *Naval Research Logistics Quarterly*, *3*, 201-203.

Jain, A. S., & Meeran, S. (1999). Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, *113*(2), 390-434.

Jennergren, P. (1973). A price schedules decomposition algorithm for linear programming problems. *Econometrica*, *41*(5), 965–980.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with set-up times included. *Naval Research Logistics Quarterly*, *1*, 61-68.

Jones, A., Rabelo, L. C., & Sharawi, A. T. (1999). Survey of job shop scheduling techniques. *Wiley Encyclopedia of Electrical and Electronics Engineering*.

Jose, R. A., Harker, P. T., & Ungar, L. H. (1997). *Coordinating locally constraind agents using augmented pricing* (Tech. Rep.). University of Pennsylvania.

Kalagnanam, J., & Parkes, D. C. (2004). Auctions, bidding and exchange design. In D. Simchi-Levi, D. S. Wu, & Z.-J. Shen (Eds.), *Handbook of Quantitative Supply Chain Analysis: Modeling in the E-Business Era* (pp. 143–212). Boston, Dodrecht, London: Kluwer Academic Publishers.

Klemperer, P. (2002). What really matters in auction design. *The Journal of Economic Perspectives*, *16*, 169–189.

Kumar, V., Kumar, S., Tiwari, M. K., & Chan, F. T. S. (2006). Auction-based approach to resolve the scheduling problem in the steel making process. *International Journal of Production Research*, *44*(8), 1503–1522.

Kutanoglu, E., & Wu, S. D. (1999). On combinatorial auction and Lagrangean relaxation for distributed resource scheduling. *IIE Transactions*, *31*, 813–826.

Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1982). Recent developments in deterministic sequencing and scheduling: A survey. In M. A. H. Dempster, J. K. Lenstra, & A. H. G. Rinnooy Kan (Eds.), *Deterministic and Stochastic Scheduling* (p. 35-73). Dordrecht: Reidel.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., & Shmoys, D. B. (1993). Sequencing

and scheduling: Algorithms and complexity. In S. C. Graves, A. H. G. Rinnooy Kan, & P. H. Zipkin (Eds.), *Handbook in Operations Research and Management Scinece* (Vol. 4, chap. 9). North-Holland.

Lawler, E. L., Lenstra, J. K., Rinnoy Kan, A. H., & Shmoys, D. B. (1993). Sequencing and scheduling: Algorithms and complexity. In *Handbooks in operations research and managment science* (Vol. 4, pp. 445–522).

Ledyard, J. O., Olson, M., Porter, D., Swanson, J. A., & Torma, D. P. (2002). The first use of a combined–value auction for transportation services. *Interfaces*, *32*(5), 4–12.

Liu, N., Abdelrahman, M. A., & Ramaswamy, S. (2007). A complete multiagent framework for robust and adaptable dynamic job shop scheduling. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, *37*(5), 904–916.

Lourenço, H. R. D. (1993). *A computational study of the job-shop and the flow-shop scheduling problems* (Unpublished doctoral dissertation). School of Operations Research and Industrial Engineering, Cornell University, Ithaca, New York. (14853-3801)

Lourenço, H. R. D. (1995). Job-shop scheduling: computational study of local search and large-step optimization methods. *European Journal of Operational Research*, *83*, 347-364. (8th June)

Moench, L. (2016, September). *Downloads scheduling.* `http://p2schedgen.fernuni-hagen.de/index.php?id=174`.

Moscato, P. (1989). *On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms* (C3P Report 826). California, USA: Caltech. (Caltech Concurrent Computation Program)

Muth, J. F., & Thompson, G. L. (Eds.). (1963). *Industrial Scheduling.* Prentice-Hall.

Nakano, R., & Yamada, T. (1991). Conventional genetic algorithm for job-shop problems. In M. K. Kenneth & B. L. B. (Eds.), *Proceedings of the 4th international conference on genetic algorithms and their applicatons* (pp. 474–479). San Diego, California, USA.

Nisan, N. (2000). Bidding and allocation in combinatorial auctions. In *Proceedings ACM Conference on Electronic Commerce* (pp. 1–12). Minneapolis MN.

Nowicki, E., & Smutnicki, C. (1996). A fast taboo search algorithm for the job–shop problem. *Management Science*, *42*, 797–813.

Panwalkar, S. S., & Iskander, W. (1977). A survey of scheduling rules. *Operations Research*, *25*(1), 45–61.

Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms and Systems* (3rd ed.). New York: Springer.

Rassenti, S. J., Smith, V. L., & Bulfin, R. L. (1982). A combinatorial auction mechanism for airport time slot allocation. *Bell Journal of Economics*, *13*, 402–417.

Rothkopf, M. H., Pekec, A., & Harstad, R. M. (1998). Computationally Manageable Combinational Auctions. *Management Science*, *44*(8), 1131–1147.

Roy, B., & Sussman, B. (1964). Les problémes d'ordannancement avec contraintes

disjoctives. *Note D.S.*, *9*.

Sadeh, N., & Nakakuki, Y. (1996). Focused simulated annealing search – An application to job–shop scheduling. *Annals of Operations Research*, *63*, 77–103.

Sandholm, T. (1999). An algorithm for optimal winner determination in comibnatorial auctions. In *Proceedengs of International Joint Conference on Artificial Intelligence* (Vol. 1, pp. 542–547). Stockholm, Sweden.

Sevkli, M., & Aydin, M. E. (2006). A variable neighbourhood search algorithm for job shop scheduling problems. *Evolutionary Computation in Combinatorial Optimization*, 261 – 271.

Srinivasan, S., Stallert, J., & Whinston, A. B. (1998). *Portfolio trading and electronic networks* (Working paper). Austin, Texas: Center for Research in Electronic Commerce, The University of Texas at Austin.

Strevell, M., & P., C. (1985). Gambling on vacation. *Interfaces*, *15*, 63–67.

Taillard, E. (1994). Parallel taboo search techniques for the job–shop scheduling problem. *ORSA Journal on Computing*, *16*, 108–117.

Van Laarhoven, P. J. M., Aarts, E. H. L., & Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. *Operations Research*, *40*, 113–125.

Vepsalainen, A. P. J., & Morton, T. E. (1987). Priority rules for job shops with weighted tardiness costs. *Management Science*, *33*, 1035–1047.

Walras, L. (1954). *Elements of Pure Economics.* London: George Allen and Unwin LTD.

Wang, J., Luh, P. B., Zhao, X., & Wang, J. (1997). An optimization-based algorithm for job shop scheduling. *Sadhana*, *22*(2), 241–256.

Wellman, M. P., Walsh, W. E., Wurman, P. R., & MacKie-Mason, J. K. (2001). Auction protocols for decentralized scheduling. *Games and Economic Behavior*, *35*(1), 271–303.

White, K. P., & Rogers, R. V. (1990). Job–shop scheduling: limits of the binary disjunctive formulation. *International Journal of Production Research*, *28*, 2187–2200.

Zhao, X., Luh, P. B., & Wang, J. (1999). Surrogate gradient algorithm for lagrangian relaxation. *Journal of Optimization Theory and Applications*, *3*, 699–712.

Zhou, R., Lee, H. P., & Nee, A. Y. (2008). Applying Ant Colony Optimization (ACO) algorithm to dynamic job shop scheduling problems. *Int. J. Manufacturing Research,*, *3*(3), 301–320.

List of Figures

List of Tables

List of Algorithms