CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Measurement

Master's Thesis

Single machine scheduling minimizing the weighted number of tardy jobs assuming strongly correlated instances

Bc. Lukáš Hejl
Open Informatics, Computer Engineering

January 2020
Supervisor: doc. Ing. Přemysl Šůcha, Ph.D.

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Hejl Lukáš** Personal ID number: **434736**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Measurement**

Study program: **Open Informatics**

Branch of study: **Computer Engineering**

II. Master's thesis details

Master's thesis title in English:

Single machine scheduling minimizing the weighted number of tardy jobs assuming strongly correlated instances

Master's thesis title in Czech:

Rozvrhování na jednom stroji s minimalizací váženého počtu zpožděných úloh pro silně korelované instance

Guidelines:

This thesis addresses a scheduling problem of minimizing the weighted number of tardy jobs on a single machine. The stress is put on strongly-correlated problem instances, i.e. instances where weight of a job equals its processing time plus a constant which is the same for all jobs. The aim is to make an attempt to improve algorithm described in [1] for strongly-correlated instances. The particular objectives of the thesis are:
1) Review the existing works in the scheduling domain and analyze results described in [1].
2) Based on [1], devise an scheduling algorithm for 1|wi = pi + K|sum wi Uj where K is a constant.
3) Implement the scheduling algorithm.
4) Compare the devised algorithm with results published in [1].

Bibliography / sources:

[1] P. Baptiste, Federico Della Croce, Andrea Grosso, Vincent T'Kindt:
Sequencing a single machine with due dates and deadlines: an ILP-based approach to solve very large instances. J. Scheduling 13(1): 39-47 (2010).
[2] Muminu O. Adamu, Aderemi O. Adewumi: A survey of single machine scheduling to minimize weighted number of tardy jobs In: Journal of Industrial & Management Optimization. 10, 12(4):1465-1493 (2014).
[3] Michael L. Pinedo: Scheduling: Theory, Algorithms, and Systems (3rd ed.). Springer Publishing Company, Incorporated. 2008.

Name and workplace of master's thesis supervisor:

doc. Ing. Přemysl Šůcha, Ph.D., Department of Control Engineering, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **21.04.2019** Deadline for master's thesis submission: _____

Assignment valid until:
by the end of winter semester 2020/2021

doc. Ing. Přemysl Šůcha, Ph.D. Supervisor's signature	Head of department's signature	prof. Ing. Pavel Ripka, CSc. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student's signature

Acknowledgement / Declaration

First of all, I would like to express my gratitude to my supervisor doc. Ing. Přemysl Šůcha, Ph.D. for his support and guidance.

My thanks also go to Ing. Antonín Novák for his valuable suggestions and comments.

Last but not least, I would like to thank my family for their support throughout my study.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, 7. 1. 2020

.....

Abstrakt / Abstract

V této práci se zaměřujeme na silně korelované instance problému minimalizace váženého součtu zpožděných úloh s termínem dokončení, které jsou zpracovávané jedním strojem. Pro tyto instance představujeme vylepšení algoritmu představeného v článku Baptiste *et al.* [1]. Hlavním vylepšením je nový ILP model pro silně korelované instance využívající dekompozici podle počtu zpožděných úloh. Dále jsme představili těsnější horní a dolní meze pro nekorelované, slabě korelované a silně korelované instance. Společně tyto vylepšení umožňují vyřešit do optima silně korelované instance až do velikosti 5 000 úloh a také výrazně zkrátit dobu výpočtu u nekorelovaných a slabě korelovaných instancí.

Klíčová slova: rozvrhování na jednom stroji, vážený počet zpožděných úloh, silně korelované instance, celočíselné lineární programování, metoda větví a mezí

Překlad titulu: Rozvrhování na jednom stroji s minimalizací váženého počtu zpožděných úloh pro silně korelované instance

In this thesis, we focus on strongly correlated instances of the problem minimizing the weighted sum of the number of tardy jobs with deadlines on a single machine. We introduce improvements to the algorithm proposed by Baptiste *et al.* [1]. The main improvement is a new ILP model for strongly correlated instances, with decomposition according to the number of tardy jobs. Other introduced improvements are a tighter lower bound and upper bound for non-correlated, weakly correlated, and strongly correlated instances. These all our improvements allow solving strongly correlated instances to the optimum for up to $5,000$ jobs, and significantly reduced the computational time on non-correlated and weakly correlated instances.

Keywords: single machine scheduling, weighted number of tardy jobs, strongly correlated instances, integer linear programming, branch and bound

Contents /

Tables / Figures

Listings /

Chapter 1
Introduction

In this thesis, we deal with a special case of instances of the problem minimizing the weighted sum of the number of tardy jobs with deadlines that are strongly correlated. The problem is formally defined by a set of jobs $N = \{1, \ldots, n\}$, where each job is defined using four integer parameters: processing time p_j, weight w_j, due date d_j and deadline $\tilde{d}_j$. Jobs are scheduled on a single machine without preemption. The solution to the problem is a schedule, where the completion time of job $j \in N$ is denoted as C_j. The goal is to find a schedule minimizing $\sum w_j U_j$ where $U_j = 1$ if the job is tardy, i.e., $C_j > d_j$. In Graham's scheduling notation, the problem is denoted as $1|\tilde{d}_j|\sum w_j U_j$ and is known to be $\mathcal{NP}$-hard.

It is well-known that certain classes of instances of $1||\sum w_j U_j$ and $1|\tilde{d}_j|\sum w_j U_j$ are significantly harder to solve than the others. The same phenomenon is observed, e.g., with Knapsack Problem [2]. Potts and Van Wassenhove [3] defined three classes of instances for $1||\sum w_j U_j$ regarding the relation between the weights and processing times as: (i) *strongly correlated*, (ii) *weakly correlated*, and (iii) *uncorrelated*. Strongly correlated instances are those where $w_j = p_j + 20$. For weakly correlated instances, an integer weight w_j is drawn from the uniform distribution $w_j \sim [p_j, p_j + 20]$, while uncorrelated instances does not have any specific relation between p_j and w_j. Strongly correlated instances were observed as significantly harder to find an optimal schedule for, compared to uncorrelated or weakly correlated instances. The results in the paper of Baptiste *et al.* [1] show that their state-of-the-art algorithm for $1|\tilde{d}_j|\sum w_j U_j$ and $1||\sum w_j U_j$ problems can solve uncorrelated instances with up to $30,000$ and $50,000$ jobs, respectively. In contrast, the same algorithm is not able to solve strongly correlated instances with just 200 jobs to optimum (4.5% of instances were not solved to optimum within one hour).

Based on our experiments, the limiting factor for solving uncorrelated and correlated instances by the algorithm published in [1] is not the same. The principal limiting factor in the case of uncorrelated instances is the memory limit. It is caused by the quadratic size of the ILP (Integer Linear Programming) model used inside the algorithm. On the other hand, strongly correlated instances are significantly harder to solve than uncorrelated ones for several different reasons. For these instances, the size of memory is not limiting, but the CPU time grows enormously with the size of the instance. Another reason why strongly correlated instances are harder lies in a certain job's dominance property used in the above-mentioned algorithm. The property states that if a certain conditions for a pair of jobs are met, then it can be concluded that the two jobs are either both early or both tardy. Thus, the property can significantly reduce the number of jobs in the lower bound calculation depending on for how many pairs of jobs the conditions are satisfied. However, for the case of strongly correlated instances, the conditions of job's dominance property translates to the requirement that the jobs have to have the identical processing times and weights, which is far more restrictive than the original version for uncorrelated instances. For this reason, it is not possible to significantly reduce the number of jobs involved in the lower bound calculation, and,

hence, the ILP model used inside the algorithm tends to be larger. Therefore, even smaller instances remain intractable for the algorithm.

The third reason is similar to the previous one. The algorithm uses variable-fixing techniques from Integer Linear Programming to *a priori* decide whether some job is early or tardy [4]. The technique requires the knowledge of upper and lower bounds on the objective function. Even though the gap between the bounds is mostly narrow, the variable-fixing technique does reduce far less jobs than in the case of uncorrelated instances.

Lastly, we note that an empirical evidence suggests the higher complexity of strongly correlated instances as well. The same ILP model employed inside the algorithm of [1] used just alone with a state-of-the-art solver can handle uncorrelated instances to size up to $5,000$ jobs, wheres for the strongly correlated it is less than 200 jobs.

In this thesis, we suggest several improvements of the algorithm by Baptiste *et al.* [1] proposed for problems $1||\sum w_jU_j$ and $1|\tilde{d}_j|\sum w_jU_j$. We particularly focus on strongly correlated instances; however, the results demonstrate significant improvement concerning the other two classes as well. This thesis presents improvements of the lower and upper bounds, described in sections 5.2 and 5.3 respectively. The new bounds, besides other things, allow to reduce more decision variables using variable-fixing techniques, and thus the algorithm is less memory demanding. Furthermore, we reformulate the ILP model of [1] for strongly correlated instances (Section 5.1). The reformulated model together with our problem decomposition allows to transform instances having $w_j = p_j + 20$ to instances with $w_j = p_j$ for which an empirical evidence suggests that are much easier to solve. These improvements led to a more efficient algorithm capable of solving larger problem instances than ever before as it is documented in Section 6.

The thesis is structured as follows. The next section provides an overview of existing work addressing problems $1||\sum w_jU_j$ and $1|\tilde{d}_j|\sum w_jU_j$. Section 3 introduces some basic notions regarding scheduling and tree different scheduling problems that are related to the one tackled in this thesis. Section 4 summarizes the important parts of the algorithm proposed by Baptiste *et al.* [1]. The core of the thesis, i.e., the improvements of the algorithm, are described in Section 5, while their assessments can be found in Section 6. The last section concludes the work.

Chapter 2
Literature review

The problem of minimizing the weighted number of tardy jobs on a single machine is studied for many years now. It is known that this problem is $\mathcal{NP}$-hard. For the deadline-free variant, i.e., $1||\sum w_jU_j$, Karp [5] proved that the problem is $\mathcal{NP}$-hard even if all jobs have a common due date. Besides that, [6] proves that if the job's processing times or weights are equal to a constant, then the problem can be solved in polynomial time. The problem $1|\tilde{d}_j|\sum w_jU_j$ remains $\mathcal{NP}$-hard even if all jobs have the same weight, e.g., $w_i = 1$ [7].

A related deadline-free variant of the problem $1||\sum w_jU_j$ is studied in more papers. Authors in papers [8] and [9] propose dynamic programming-based algorithms with pseudopolynomial time complexity. The few subsequent papers use branch-and-bound based algorithms to find the exact solution. Villarreal *et al.* [10] reduce the size of the problem instances by the application of a dominance theorem. This approach enabled solving instances having up to 50 jobs. Tang in [11] introduces some new job's dominance rules. The algorithm can solve instances of up to 85 jobs. Authors of [3] propose an algorithm for solving linear relaxation of $1||\sum w_jU_j$ with time complexity $\mathcal{O}(n \cdot \log n)$. Their algorithm can solve instances with $1,000$ jobs. M'Hallah and Bulfin [12] show an algorithm that can solve instances having up to $2,500$ jobs. The algorithm uses Knapsack problem to compute a bound on the objective function of $1||\sum w_jU_j$.

Only a few papers study exact algorithms for the problem $1|\tilde{d}_j|\sum w_jU_j$. One of the first exact algorithms for this problem is introduced in paper [13]. The algorithm is based on the branch-and-bound with dynamic programming for computing bounds and can solve instances up to 300 jobs. The state-of-the-art algorithm is introduced by Baptiste *et al.* [1]. The algorithm is capable to solve very large instances. It is based on a branch-and-bound technique that uses variable-fixing techniques to reduce the problem size. The authors also introduce methods for computing very tight lower and upper bound. The lower bound is based on a transformation to the maximum profit flow problem; the upper bound computation uses a dominance theorem for reducing jobs set. The algorithm solves uncorrelated instances for both problems $1|\tilde{d}_j|\sum w_jU_j$ and $1||\sum w_jU_j$ up to $30,000$ and $50,000$ jobs respectively.

Recent research papers addressing the single machine total number of (weighted) tardy jobs problem concentrate on more specific variants of this problem with the additional job's characteristics. The authors of paper [14] focus on a periodic maintenance (PM) problem modeled as $1|PM|\sum U_j$. For this problem, they present an exact algorithm based on an improved branch-and-bound method with effective lower bound and several dominance properties. They also show that this problem is $\mathcal{NP}$-hard in a strong sense. Wang et al. [15] developed two different heuristic algorithms for the problem $1|p_{j,r} = (1 + p_{[1]} + p_{[2]} + ... + p_{[r-1]})^a|\sum U_j$, denoting a single machine scheduling problem with time dependent learning effect. Both heuristics are capable find near-optimal solutions. The authors also show an exact branch-and-bound algorithm for which they present lower bound and two dominance properties. Paper [16] focuses on

two problems $1|cos, prec|\sum w_j U_j$ and $Q_m|cos, p = 1|\sum w_j U_j$, where *cos* means common operation scheduling, i.e., situation when jobs share operations. They propose for both problem formulations using the set cover problem with an exponential number of constraints, and use the branch-and-cut method to solve these formulations. Zhao and Yuan [17] present an improved algorithm with time complexity $\mathcal{O}(n \cdot \log n)$ for a problem dealing with a trade-off between the number of tardy jobs and start time of a machine, denoted as $1||^{\#}(\sum U_j, A)$. They also present a new algorithm for solving problem $1||\sum U_j$, with time complexity $\mathcal{O}(n \cdot \log n)$. The authors in paper [18] address single machine scheduling problem with machine unavailability periods and a common due date denoted as $1, h_{m-1}|nr-a, d_j = d|\sum w_j U_j$. They use binary multiple knapsack problem to formulate the problem, and show that some large instances can be easily solved with an off-the-shelf-solver for binary multiple knapsack problem. They also developed a heuristic based on variable neighborhood search technique for instances that are difficult for an off-the-shelf solver.

Chapter 3
Scheduling overview

This chapter focuses on introducing the reader to scheduling problems. The first part of the chapter summarizes the most basic concepts. In the next part, the reader is acquainted with Graham's notation [19]. Several uses of problem $1|\tilde{d}_j|\sum w_j U_j$ and related problems are presented in Section 3.4. The rest of the section focuses on problems related to problem $1|\tilde{d}_j|\sum w_j U_j$, algorithms that solve these problems, and demonstrating how they work on examples.

3.1 Definition of scheduling problems

This section, with a definition of scheduling problems, is based on book [20]. To define a scheduling problem, we introduce two sets. Set of n jobs $J = \{J_1, J_2, ..., J_n\}$; these jobs J_i are each defined by several parameters such as processing time, deadline, and many others. These most basic parameters are summarized and described in Section 3.2. The second set is set of m machines $P = \{P_1, P_2, ..., P_m\}$. These machines can be dedicated to specific jobs, or allow the execution of all jobs, which are called parallel machines. These machines can also have different processing speeds of jobs and many other parameters. Part of these parameters is listed in Section 3.3.

In the definition of a scheduling problem, Blazewicz [20] defines a set of additional resources R, which we have excluded from the definition since, in this thesis, we consider only the two sets J and P defined above.

In scheduling, we deal with assigning jobs from set J to set of machines P and time points so that all constraints specified by parameters of jobs from set J can be met. All jobs from set J must be completed, making scheduling different from planning. Furthermore, it is also assumed that each machine processes at most one job at a time, and likewise, each job can be processed by only one machine at a time.

The outcome of scheduling is a schedule in which it is determined which jobs from set J are processed on which machine from set P, including the time points when the job processing starts and completed. In cases where we are not trying to find only a feasible schedule, but a feasible schedule that minimizes (or maximizes) a certain function, e.g., $\sum w_j U_j$, then this function is denoted as an objective function.

The following section describes the basic parameters of jobs.

3.2 Common parameters of job

In this section, we describe the most common parameters of jobs based on book [21].

3.2.1 Processing time

The processing time of job J_i is denoted as p_i, and it is one of the most common parameters of a job, which models the time needed to process the specific jobs. Processing time may be arbitrary for each job. There are also scheduling problems in which jobs have a common processing time or unit processing time.

3.2.2 Deadline

The deadline for job J_i is denoted as $\tilde{d}_i$. It is a date to which job J_i must be completed. If the job is placed in the schedule so that it is not completed before its deadline, it means that this schedule is infeasible. Therefore, it is sometimes called a hard deadline. For these reasons, it is also necessary to ensure that deadline $\tilde{d}_i$ of job J_i is not less than processing time p_i. Otherwise, it would not be possible to find the schedule in which the job is completed before its deadline.

3.2.3 Due date

The due date of job J_i is denoted as d_i. It is a date to which job J_i can be completed, but if it is not completed to this date, the schedule is still feasible. Therefore, it is sometimes called a soft deadline. The due date is often incorporated into a objective function. For example, if we try to minimize the number of jobs that are completed before its due date.

3.2.4 Release time

The release time of job J_i is denoted as r_i. When using the release time parameter, it is not considered that all jobs are ready to proceed from the beginning (at the time 0). The time when the job is ready to proceed is specified as r_i, which may be arbitrary for each job. A job cannot run before its release time, because it is not ready yet, and such a schedule is not acceptable, which is similar to the deadline.

3.2.5 Weight

We denote the weight of job J_i as w_i. This parameter is usually a part of the objective function, e.g., $\sum w_j U_j$.

3.2.6 Completion time

The completion time of job J_i is denoted as C_i. Compared to the previous parameters of jobs, completion time is not a parameter that would typically be specified for each job in advance. This parameter depends on the position of a job in a schedule. Their positions determine the time when a job is completed. The completion time is also affected by other jobs and their parameters. This parameter often appears in the objective function, such as minimizing the schedule length, minimizing the sum of jobs completion times, and many more.

3.3 Graham notation

This section, which describes Graham notation, is based on book [20]. Graham notation is a form of notation of scheduling problems. It is one of the most commonly used forms for scheduling problems.

Graham notation is composed of three fields α, β and γ, which are separated by the character $|$. Each problem which is described by this notation is in the form of $\alpha|\beta|\gamma$. The first field α describes machine properties. The second field β describes the properties of the jobs, and the last field γ describes the used objective function.

3.3.1 Field α

This field of the notation, denoted as α, defines the parameters of machines. These parameters are the type of machine and its count. This field is divided into two other subfields. The first subfield denotes the type of machine, which includes:

- $\emptyset$ - denotes a single machine,
- P - denotes parallel machines that are identical,
- Q - denotes parallel machines, where each has its speed, and this speed determines the time needed to process the job,
- R - denotes parallel machines, where each job is processed with different speeds on each machine. This relationship is defined using a matrix that determines the speed of a particular job on a particular machine.

Also, several other types of machines are used, but these are not listed because they are not related to the scheduling problems in this thesis.

The second subfield shows the number of used machines:

- $\emptyset$ - denotes that the number of machines is not fixed,
- 1 - denotes single machine,
- k - denotes a fixed number of machines equal to k.

3.3.2 Field β

This field denoted as β, defines parameters of jobs and their relationships between them. A list of several used parameters of jobs:

- Preemption - $\emptyset$ denotes that jobs must be processed without interruption, while $pmtn$ denotes that the job can be interrupted while it is processing and then can be resumed at a later time,
- Release time - $\emptyset$ denotes that jobs are ready at the beginning of a schedule, while r_j denotes that jobs are ready only after a defined time has elapsed,
- Processing time - $\emptyset$ denotes that jobs have arbitrary processing times, while, for example, $p_j = 1$ denotes that all jobs have a processing time equal to 1,
- Deadline - $\emptyset$ denotes jobs that do not have a deadline, while d_j denotes that jobs are constrained by deadlines.

The list includes only the essential parameters we encounter in this thesis, others that determine, for example, dependencies between jobs, we have omitted for keeping this subsection simple.

3.3.3 Field γ

This field defines what objective function is used in the scheduling problem. $\emptyset$ or $-$ are used to define problems where we only need to find an acceptable solution. There is a large number of objective functions, but only a few examples are given:

- $\sum U_j$ - denotes maximizing the number of early jobs,
- $\sum w_j U_j$ - denotes maximizing the weighted sum of early jobs,
- C_{max} - denotes minimizing the completion time of the last job in the schedule,
- $\sum C_j$ - denotes minimizing the sum of completion times for all jobs,
- $\sum w_j C_j$ - denotes minimizing the weighted sum of completion times of jobs.

In case that objective function contains U_j, it is assumed that jobs have a due date, then the due date is not listed in the field β unless it has additional constraints.

3.4 Examples of usage problems $1||\sum U_j$, $1||\sum w_jU_j$ a $1|\tilde{d}_j|\sum w_jU_j$

In this section, we show some examples of using scheduling problems $1||\sum U_j$, $1||\sum w_jU_j$ and $1|\tilde{d}_j|\sum w_jU_j$. These examples show possible uses of problems and also demonstrate differences between these problems.

Book [21] presents an example of using the problem $1||\sum U_j$ in a situation, where a student is preparing for exams. This student was devoting to other activities and was not preparing for exams. The student is in the situation when he/she is unable to prepare for all the exams. This student is trying to plan preparation for exams and maximize the number of exams, which he/she will pass. This problem can be formulated using the problem $1||\sum U_j$, where each exam is represented by one job J_i. The time required to prepare for the exam is represented by processing time p_i of job J_i, and a due date d_i of job J_i represents the date of exam i.

The following examples illustrate the use of the problem $1|\tilde{d}_j|\sum w_jU_j$ in several areas of industry.

One of possible applications of this problem, according to the paper [13], can be found in the logistics of supplied products. Suppose we have two different types of transport. One of them may be, for example, road transport, which is cheaper but even slower. Another type of transport can be, for example, air transport, which is faster, but significantly more expensive than road transport. We have contracted the date with the customer by which the product will be delivered. Our goal is to create a schedule of product manufacturing to minimize shipping costs, and all products will be delivered on time.

This problem can be formulated using the problem $1|\tilde{d}_j|\sum w_jU_j$, where each produced product is represented by one job J_i. The time required to produce the product is represented by the processing time p_i of job J_i. The due date d_i of job J_i represents the date by which a product must be produced to be able to be delivered on time by road. The deadline $\tilde{d}_i$ for job J_i represents the date by which the product must be produced in order to be able to be delivered on time but using more expensive air transport. The difference in the cost of shipping a product by air transport instead of road transport is expressed as weight w_i of job J_i.

Paper [13] mentions another possible use of the problem $1|\tilde{d}_j|\sum w_jU_j$, this time in the field of agriculture in crops harvesting. Suppose we have one harvester that is used to harvest crops in different areas. The harvester always harvests the entire area. After harvesting of this area is finished, it begins to harvest the next area. Harvested crops are then sold in two markets, local and foreign. In the local market, the crop is sold at a lower price, while when the crop is exported abroad, it is sold at a higher price. Exported abroad takes time, so in this case, it is necessary to take into account ripening or spoilage of the crop during transport.

This problem can be formulated using the problem $1|\tilde{d}_j|\sum w_jU_j$„ where each harvested area is represented by one job J_i. The time needed for harvesting the area with a harvester is represented by processing time p_i of job J_i. The date by which the crop must be harvested in order to be exported to the foreign market is represented by due date d_i of job J_i. The deadline $\tilde{d}_i$ for job J_i represents the date by which the crop must be harvested in order to be sold on the local market. Weight w_i of job J_i represents the difference in the price when we are selling a crop on a foreign market compared to the local market.

In the following subsection we present an example showing the differences between problems $1||\sum U_j$, $1||\sum w_j U_j$ and $1|\tilde{d}_j|\sum w_j U_j$.

3.4.1 Example showing the differences between all tree problems

Based on the above example from book [21], we have created a similar example, also coming from the student environment, in which in three different variations, we demonstrate the differences between the individual problems. Again, as in the original example, the student solves the problem of lacking time to complete all assignments due to too many activities.

In the first variant, all assignments are awarded the same number of points for their submission on time, submitting all assignments is not mandatory and the student tries to maximize the number of assignments. This problem can be formulated as problem $1||\sum U_j$, where each assignment i is represented as one job J_i. The deadline for submitting an assignment for receive points is modeled as due date d_i of job J_i. The time required to complete an assignment is represented by processing time p_i of job J_i.

In another variant, it is considered that all assignments are awarded by a different number of points for submitting on time. Submission of assignments is again not mandatory, but this time the student tries to maximize the gain of point for assignments. This problem can be formulated as problem $1||\sum w_j U_j$, where each assignment i is represented as one job J_i whose parameter processing time p_i, and due date d_i are assigned in the same sense as in the previous case. A new parameter of job J_i is weight w_i, which is used for modeling points for assignments.

In the last variant, it is again considered that the assignments are awarded by a different number of points for submitting on time. Also, it is mandatory to submit an assignment to some date (e.g., before the end of a semester). The students receive points only for submitting on time. Once again, the student tries to maximize the gain of point for the assignments. This problem can be formulated as problem $1|\tilde{d}_j|\sum w_j U_j$, where each assignment i is represented as one job J_i whose parameters the processing time p_i, due date d_i, and weight w_i are assigned as in the previous case. A new parameter of job J_i is the deadline $\tilde{d}_i$, which is used to represent the date until the assignment must be submitted in order to complete the course.

3.5 Problem $1||\sum U_j$

The $1||\sum U_j$ problem can be seen as a simplification of the problem $1|\tilde{d}_j|\sum w_j U_j$ that occurs by removing deadlines $\tilde{d}_i$ for all jobs and weight w_i from the objective function. If only deadlines $\tilde{d}_i$ are removed, the complexity of new problem $1||\sum w_j U_j$ is still $\mathcal{NP}$-hard, which was proved in paper [5]. If only weight w_i is removed from the objective function, the complexity of new problem $1|\tilde{d}_j|\sum U_j$ is also $\mathcal{NP}$-hard, which was proved in paper [7]. Only after removing both deadlines and weights, new problem $1||\sum U_j$ become solvable in polynomial time. One algorithm that solves problem $1||\sum U_j$ is the Moore-Hodgson algorithm, which solves this problem with $\mathcal{O}(n \cdot \log n)$ time complexity. This algorithm is described in the following subsection.

3.5.1 Description of Moore-Hodgson algorithm

The algorithm first orders all jobs in the instance according to their due dates from the earliest to the latest ($d_1 \leq d_2 \leq ... \leq d_n$). These jobs are processed sequentially from the job with the earliest due date to the job with the latest due date. The algorithm maintains two variables. Variable S represents a set of jobs that can be completed

before their due date. The second variable t determines the current time point in the constructed schedule. During the processing of each job, this job is inserted in set S, and variable t is increased by its processing time. A check follows to determine if the last added job will be completed before its due date. If this job succeeds in completing before its due date, the algorithm continues processing to the next job. Otherwise, the algorithm iterates through all the jobs in set S and selects job J_k with the longest processing time. It removes job J_k from set S and also subtracts its processing time from t. This step replaced job J_k with the currently processed job. The algorithm continues processing to the next job until all jobs are processed. At the end of the algorithm, we get a set S containing the maximum number of jobs to be completed before their due date [22].

The algorithm described in the previous paragraph can be expressed as pseudocode as follows:

```
1  Sort J = {J_1, J_2, ..., J_N} by due dates in ascending order
2  Let S be an empty set and t := 0
3  For i := 1 to N
4      Add J_i to set S
5      t := t + p_i
6      If t > d_i
7          Find job J_j in S with the largest p_j
8          Remove J_j from set S
9          t := t - p_j
```

3.1. Moore-Hodgson algorithm for problem $1||\sum U_j$ taken from book [22].

In order to create a schedule from the obtained set S, we must first order all jobs from set S according to their due dates from the earliest to the latest. According to this order, jobs are subsequently placed to the schedule. The remaining jobs expressed as $N \setminus S$, are jobs that are tardy because they are not completed before their due date. These jobs can be placed in the schedule after the jobs from set S. We can choose their order arbitrary because these jobs cannot be completed before their due date (in terms of placement after jobs from set S), and there are no other constraints.

For a better understanding of the algorithm, we will show how the Moore-Hodgson algorithm works on a simple example in the following subsection.

3.5.2 Example

The operation of the above algorithm is demonstrated on a simple instance of this problem with five jobs. These jobs and their parameters are listed in Table 3.1. Since the algorithm initially orders jobs in the instance from the earliest due date to the latest, the jobs in Table 3.1 are ordered in this manner.

job	p_i	d_i
J_1	1	2
J_2	4	7
J_3	3	8
J_4	3	8
J_5	2	9

Table 3.1. The instance with five jobs of problem $1||\sum U_j$

The algorithm first process job J_1 because it proceeds from the job with the earliest due date to the job with the latest due date. It adds this job to set $S = \emptyset$ and adds processing time $p_1 = 1$ to variable $t = 0$. After this step, set $S = \{J_1\}$ and variable $t = 1$. The next step is to check if the job being processed will be completed before its due date, which is $d_1 = 2$. Since $t \leq 2$, J_1 is completed before its due date, and the algorithm can continue processing other jobs. In the same way, it processes the other two jobs J_2 and J_3. After processing these jobs, set $S = \{J_1, J_2, J_3\}$ and variable $t = 8$. Figure 3.1 shows a partial schedule based on set S.

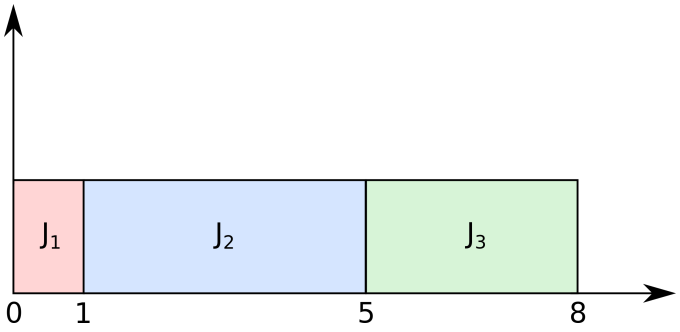

Figure 3.1. Partial schedule of the example of problem $1||\sum U_j$ created from S.

When the algorithm is processing job J_4, the job is again added to S, and variable t is increased by processing time $p_4 = 3$, so $t = 11$. By checking if job J_4 will be completed before its due date ($d_4 = 8$), it is found that $t > 8$, therefore job J_4 would not be able to complete before its due date and would be flagged as tardy. In this case, the algorithm selects job J_2 from set $S = \{J_1, J_2, J_3, J_4\}$ because it has the greatest processing time ($p_2 = 4$). It removes this job from S and decreases value of t from 11 to 7 ($p_2 = 4$). After this step, job J_4 will be completed before its due date $d_4 = 8$.

Subsequently, the algorithm processes the last job J_5. This job is added to S, and variable $t = 7$ is increased by processing time $p_5 = 2$ to $t = 9$. The next check shows that job J_5 can be completed before its due date $d_5 = 9$ because of $t \leq 9$. Thus, the algorithm ends and returns set $S = \{J_1, J_3, J_4, J_5\}$ containing jobs, that are early.

To create the schedule, we order jobs from set $S = \{J_1, J_3, J_4, J_5\}$ by due dates from the earliest to the latest. This gives the following order of jobs J_1, J_3, J_4, J_5. In this order, we put the jobs into the schedule. We place the remaining job J_2 behind these jobs. The created schedule, with the maximum value of $\sum U_j$, is shown in Figure 3.2.

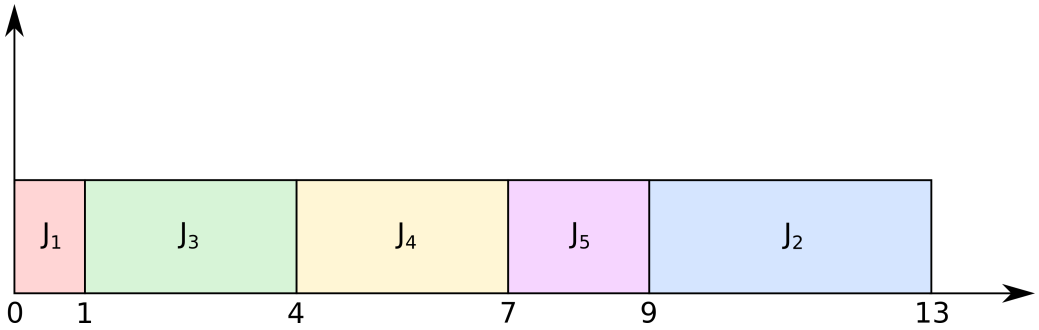

Figure 3.2. The final schedule of the example of problem $1||\sum U_j$ created from S.

It is good to note that removing job J_2 from set S allowed not only job J_4 but also job J_5 to be added to set S. If job J_2 remained in set S, the total number of early jobs would be only 3, compared to the optimal 4.

3.5.3 Connection with problem $1||\sum w_jU_j$

Problem $1||\sum U_j$ is related to problem $1||\sum w_jU_j$. Both of these problems share solutions space. This means that if we consider an instance of problem $1||\sum w_jU_j$ and create an instance of problem $1||\sum U_j$ by merely omitting the weight of the jobs, it is possible to say that a feasible solution to the first problem is also a feasible solution to the second problem and vice versa. This is because the problems differ only in the objective function. Moreover, if we do not consider the deadlines, any permutation of jobs in the schedule is a feasible solution.

There is the same connection between problems $1|\tilde{d}_j|\sum U_j$ and $1|\tilde{d}_j|\sum w_jU_j$, which also share the solution space. Also, these two problems differ only in the objective function, which, as in the previous case, does not impose any constraints on jobs. Adding the deadline parameter means that any permutation of jobs in the schedule is no longer a feasible solution, and also that both $1|\tilde{d}_j|\sum U_j$ and $1|\tilde{d}_j|\sum w_jU_j$ problems are $\mathcal{NP}$-hard, as demonstrated in papers [7] and [5].

We use the fact that the above pairs of problems share a solution space to determine the maximum number of early jobs for a given instance, as described in later Section 5.2.

3.6 Problem $1|p_i = 1|\sum w_jU_j$

One of the further simplifications of problem $1|\tilde{d}_j|\sum w_jU_j$ is problem $1|p_i = 1|\sum w_jU_j$, which does not include deadlines compared to the original problem, but only due dates and all jobs have unit processing time.

After removing the deadlines for all jobs and restricting the job processing time to $p_i = 1$, this problem can be solved in polynomial time. In contrast, the original problem $1|\tilde{d}_j|\sum w_jU_j$ is $\mathcal{NP}$-hard, as demonstrated in paper [5].

The algorithm solving problem $1|p_i = 1|\sum w_jU_j$ with time complexity $\mathcal{O}(n \cdot \log n)$ is described in the following subsection.

3.6.1 Description of the algorithm

The algorithm first orders all jobs in the instance according to their due dates from the earliest to the latest ($d_1 \leq d_2 \leq ... \leq d_n$). The algorithm maintains two variables. It uses variable S to register a set of jobs that can be completed before its due date. The second variable is t, which holds the current time point in the constructed schedule and it is initialized to 1. The algorithm proceeds jobs from the job with the earliest due date to the job with the latest due date. During the processing of each job J_i, the job is checked if it satisfies the condition $d_i \geq t$ alternatively, if this job could be completed before its due date if the algorithm adds it to set S. If job J_i meets this condition, it is inserted into set S, and variable t is increased by 1, which corresponds to the processing time of each job. If a job cannot be added to set S because it would not be completed before its due date (would be marked as a tardy job), then the algorithm will try to find a job $J_k \in S$ that satisfies the condition $w_k < w_i$. If the algorithm does not find any such job J_k, then job J_i is not added to set S, and the algorithm continues to process other jobs. Otherwise, $J_k \in S$ with the smallest w_k is selected and removed from S and replaced by job J_i. The algorithm then continues processing the next job, until all jobs are processed. After the algorithm processes all jobs, we get set S containing jobs that will be completed before their due dates and maximizing the weighted sum $\sum w_jU_j$ [22].

The algorithm described in the previous paragraph can be expressed as pseudocode as follows:

```
Sort J = {J_1, J_2, ..., J_N} by due dates in ascending order
Let S be an empty set and t := 1
For i := 1 to N
    If d_i >= t
        Add J_i to set S
        t := t + 1
    Else if there exists a job J_k in set S with w_k < w_i
        Find job J_k in set S with the smallest w_k
        Remove J_i from set S
        Add J_i to set S
```

3.2. The algorithm for problem $1|p_i = 1|\sum w_j U_j$ taken from book [22].

If we want to create a schedule from the obtained set S, we must first order all jobs from S according to their due dates from the earliest to the latest. Subsequently, we will place these jobs in the schedule according to this order from the earliest due date to the latest. Remaining jobs that are not included in set S are jobs that are tardy because they are not completed before their due date. These jobs can be placed in the schedule after the jobs from set S in arbitrary order.

For better understanding of the algorithm, we will show how the algorithm works on a simple example in the following subsection.

3.6.2 Example

The operation of the above-described algorithm for problem $1|p_i = 1|\sum w_j U_j$ is demonstrated on a simple instance of this problem with five jobs. These jobs and their parameters are listed in Table 3.2. Jobs in Table 3.2 are ordered from the earliest due date to the latest to skip the step when the algorithm initially performs ordering of jobs in the instance.

job	p_i	d_i	w_i
J_1	1	1	2
J_2	1	2	5
J_3	1	3	3
J_4	1	3	4
J_5	1	4	3

Table 3.2. The instance with five jobs of problem $1|p_i = 1|\sum w_j U_j$.

The algorithm firstly initializes variable S to $\emptyset$ and variable t to 1. Then the algorithm first processes job J_1 because it has the earliest due date $d_1 = 1$. The algorithm first performs testing of condition $d_1 \geq t$ if job J_1 is completed before its due date. The job satisfies the condition $d_1 \geq t$, therefore, it is added to S, and variable t is increased by processing time $p_1 = 1$ of job J_1. After this step, $S = \{J_1\}$ and variable $t = 2$. The algorithm then continues to process the next job. It will process two other jobs J_2 and J_3, in the same way as described above. After processing these jobs, set $S = \{J_1, J_2, J_3\}$ and variable $t = 4$. Based on the current state of set S, we can calculate the value of the objective function $\sum w_j U_j = 10$. Figure 3.3 shows a partial schedule created based on S.

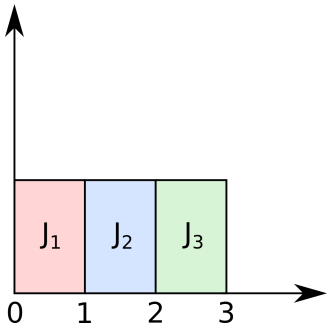

Figure 3.3. Partial schedule created from S with the current value of the objective function 10.

When processing J_4, there is a change, when testing of the condition $d_4 \geq t$, it is found that job J_4 does not meet this condition and would not be completed before its due date $d_4 = 3$. In this case, the algorithm iterates through set $S = \{J_1, J_2, J_3\}$ from which it selects job J_1 because it has the smallest weight w_1, for which $w_1 < w_4$ also holds. This job is removed from set $S = \{J_1, J_2, J_3\}$ and replaced by job J_4. Subsequently, the algorithm continues to process job J_5, which is the last job. Based on the testing of condition $d_5 \geq t$, this job will be completed before its due date. Therefore it is inserted into set $S = \{J_2, J_3, J_4\}$, and the value of variable $t = 4$ is increased by 1. After the algorithm is finished, variable $t = 5$ and set $S = \{J_2, J_3, J_4, J_5\}$. Also, the algorithm returns set $S = \{J_2, J_3, J_4, J_5\}$.

After completing the algorithm, we have a set $S = \{J_2, J_3, J_4, J_5\}$ of jobs that are early. To create a schedule, we order this set by due dates from the earliest to the latest. This gives us the following order of jobs J_2, J_3, J_4, J_5. And we put the jobs into the schedule in this order. Subsequently, we place the remaining job J_1 into the schedule after the jobs from set S. The resulting schedule with the maximum value of the objective function $\sum w_j U_j = 15$ is presented in Figure 3.4.

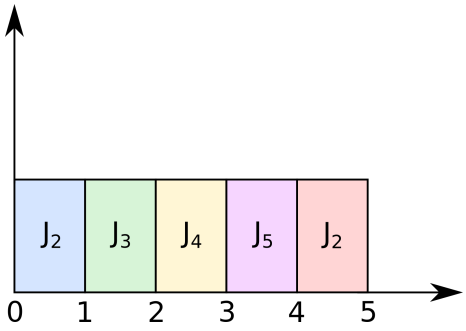

Figure 3.4. The final schedule created from S with the value of the objective function 15.

3.6.3 Special case $1|p_j = 1|\sum U_j$

By removing the weights from problem $1|p_j = 1|\sum w_j U_j$, we obtain problem $1|p_j = 1|\sum U_j$, which is a special case of problem $1||\sum U_j$ differing in fixed processing times $p_i = 1$.

For this problem, we can use the original algorithm presented in the previous Subsection 3.6.1, or we can use the algorithm described in this subsection for problem $1|p_i = 1|\sum w_j U_j$. This algorithm should be modified by omitting the `else` branch on line 7 because the all jobs do not have a weight parameter (or all jobs have the same weight), so this part is redundant.

This modified algorithm is written using the pseudocode as follows:

```
1  Sort J = {J_1, J_2, ..., J_N} by due dates in ascending order
2  Let S be an empty set and t := 1
3  For i := 1 to N
4      If d_i >= t
5          Add J_i to set S
6          t := t + 1
```

3.3. The reduced algorithm for problem $1|p_j = 1|\sum U_j$ based on book [22].

The modified algorithm has the same time complexity as the original variant for problem $1|p_i = 1|\sum w_jU_j$, which is $\mathcal{O}(n \cdot \log n)$ because only redundant parts of the algorithm are removed.

Assuming that jobs are already ordered by due date from the earliest to the latest, it would be possible to remove the ordering of the input on line 2. After removing this line and assuming that jobs are ordered by the due date, then the algorithm would have time complexity only $\mathcal{O}(n)$.

3.7 Problem $1||\sum w_jU_j$

Jobs of problem $1||\sum w_jU_j$ do not have deadlines opposed to jobs of problem $1|\tilde{d}_j|\sum w_jU_j$, which is defined in Section 1. By removing the deadlines, each permutation of jobs in the schedule is a feasible solution, differing only by the value of the objective function. This omission of the deadlines do not affect the complexity class, so the problem $1||\sum w_jU_j$ is also NP-hard, as shown in paper [5].

This problem is also very close to the problem $1|\tilde{d}_j|\sum w_jU_j$, at least by being formulated using the same ILP model, and the algorithm presented in paper [1] solves instances for both of these problems.

The algorithm for solving the problem $1||\sum w_jU_j$ with pseudo-polynomial time and space complexity $\mathcal{O}(n\sum p_i)$ is described in the next subsection.

3.7.1 The algorithm for solving problem $1||\sum w_jU_j$

The authors of paper [8] presented this algorithm. The algorithm only works under the assumption that jobs $J_1, J_2, ..., J_n$ are ordered according to the due date on the earliest to the latest ($d_1 \leq d_2 \leq ... \leq d_n$). The algorithm uses a method of dynamic programming, which is based on the function $f_j(t)$, where $j = 0, 1, ..., n$ and $t = 0, 1, ..., \sum_{j=0}^{n} p_j$. This function $f_j(t)$ represents the optimal value for jobs $J_1, J_2, ..., J_j$, with a constraint on processing time to a maximum of t. The initial values are defined as follows: $f_j(t) = 0$ when $j = 0 \vee t = 0$ and $f_j(t) = -\infty$ when $t < 0$. The following recursive relation defines the function:

$$f_j(t) = \begin{cases} -\infty & \text{if } t > d_j, \\ \max\{f_{j-1}(t), f_{j-1}(t - p_j) + w_j\} & \text{otherwise.} \end{cases} \quad (1)$$

This described algorithm can be written as pseudocode as follows:

```
Sort J = {J_1, J_2, ..., J_N} by due dates in ascending order
Let P be the sum of all p_i
Let T and W be (N+1)x(P+1) matrices
For t := 0 to P
    T[0][t] = 0
For i := 1 to N
    T[i][0] = 0
For i := 1 to N
    For t := 1 to P
        T[i][t] := T[i - 1, t]
        W[i][t] := 0
        If t > d_i
            T[i][t] := negative infinity
        Else if T[i - 1, j - p_i] + w_i > T[i - 1, t]
            T[i][t] := T[i - 1, j - p_i] + w_i
            W[i][t] := 1
Let S be an empty set
Let M be an index of the maximal element in the last row of T
For i := N to 0
    If W[N][M] = 1
        Add J_i to set S
    M := M - p_i
```

3.4. The algorithm for problem $1||\sum w_j U_j$ created from recursive relation (1).

3.7.2 Limitations of the algorithm

The algorithm described in Subsection 3.7.1, which is based on the principle of dynamic programming, has one major limitation. The runtime of this algorithm not only depends on the size of a problem but also on the sum of processing time of all jobs. Thus, if the processing time of jobs is chosen improperly, the runtime and memory consumption of the algorithm can be extremely large, even for small instances. For example, if a processing time of jobs is uniformly selected from range $1 - 10$, then approximately 0.5GB of RAM will be needed to store this table for $5,000$ jobs. On the contrary, if processing times of jobs are uniformly selected from range $10,000 - 100,000$, the same amount of memory will be needed for instances with 50 jobs.

Despite this limitation, the dynamic programming with pseudo-polynomial time and memory complexity is the best option if there is no other exact algorithm, or it has a better time complexity for given job parameters.

3.7.3 Example

The tree created from the recursive relation above is shown for a simple instance with five jobs. These jobs and their parameters are listed in Table 3.3. Since the algorithm requires ordering jobs by due date from the earliest to the latest, the jobs in Table 3.3 are ordered like that.

We compute the value of the recursive function for $f_5(8)$, which we intentionally selected because, for $j = 5$ and $t = 8$, the function returns the optimal value for these arguments, for the instance in Table 3.3. In Figure 3.5, the same subtrees $f_3(5)$ can be observed. These subtrees are computed twice, although they have the same result. For larger instances, this occurs considerably more often, and through dynamic programming, these repeated computations of the same subtrees are eliminated.

job	p_i	d_i	w_i
J_1	1	4	2
J_2	3	4	5
J_3	2	5	4
J_4	2	7	3
J_5	3	6	7

Table 3.3. The instance with five jobs of problem $1||\sum w_jU_j$.

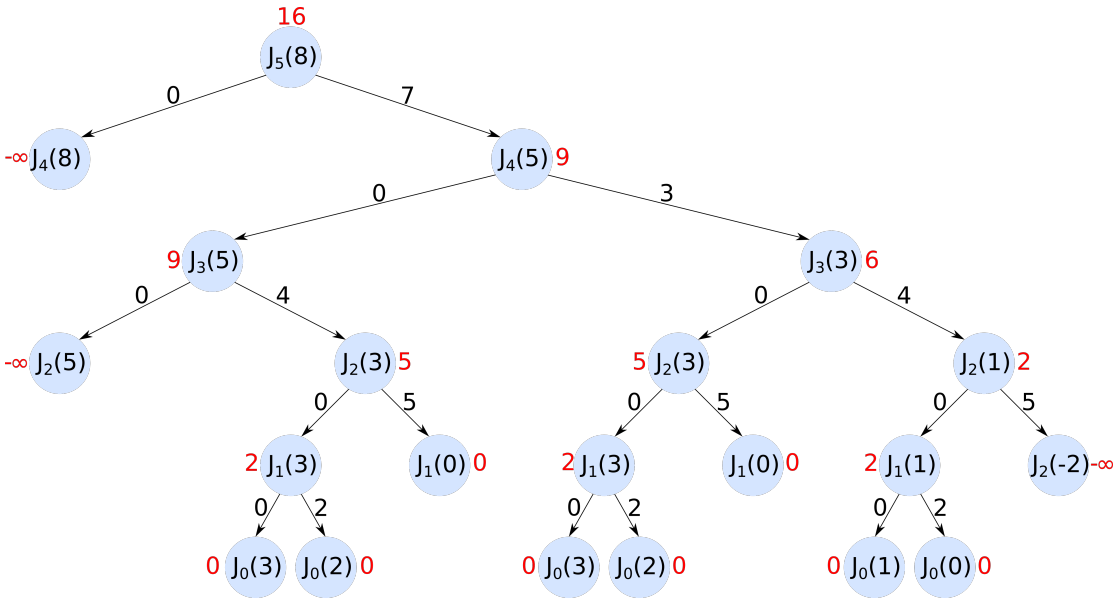

Figure 3.5. Tree of the recursive relation (1) for $f_3(5)$. The red numbers next to the nodes represent the optimal value of the corresponding subtree. The numbers next to the arrows represent the increment to the objective function.

Using the previously described algorithm for the instance listed in Table 3.3, we obtain two values for each position in the dynamic programming table. Both values are shown in Table 3.4. The first is the value of the objective function, and the second is the binary information if the job at the index j is early (1) or tardy (0). This information is represented as a subscript. For the example given in Table 3.3, the result is in Table 3.4, where the subscript indicates whether the job is early or not. Since computing the Table 3.4 is only applying a recursive relation (1), we do not describe the creation of Table 3.4.

	$t=0$	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$	$t=11$
$j=0$	0_0	0_0	0_0	0_0	0_0	0_0	0_0	0_0	0_0	0_0	0_0	0_0
$j=1$	0_0	2_1	2_1	2_1	2_1	$-\infty_0$	$-\infty_0$	$-\infty_0$	$-\infty_0$	$-\infty_0$	$-\infty_0$	$-\infty_0$
$j=2$	0_0	2_0	2_0	5_1	7_1	$-\infty_0$	$-\infty_0$	$-\infty_0$	$-\infty_0$	$-\infty_0$	$-\infty_0$	$-\infty_0$
$j=3$	0_0	2_0	4_1	6_1	7_1	9_1	$-\infty_0$	$-\infty_0$	$-\infty_0$	$-\infty_0$	$-\infty_0$	$-\infty_0$
$j=4$	0_0	2_0	4_1	6_1	7_1	9_1	10_1	12_1	$-\infty_0$	$-\infty_0$	$-\infty_0$	$-\infty_0$
$j=5$	0_0	2_0	4_0	7_1	9_1	11_1	13_1	14_1	16_1	$-\infty_0$	$-\infty_0$	$-\infty_0$

Table 3.4. Data computed by Algorithm 3.4 for the example of problem $1||\sum w_jU_j$.

We create set of early S jobs from Table 3.4 as follows. In the row with index $j=5$, we select the maximum value, which is 16 at position $(j=5, t=8)$. The subscript, it

1, which determines that job J_5 is early and therefore belongs to set S. Therefore, it decrements index j and subtracts the processing time $p_5 = 3$ from t. Then, we continue at position $(j = 4, t = 5)$. At this position, there is a value 1 in the subscript again, so J_4 is added to set S, and we continue to position $(j = 3, t = 3)$. The situation is again the same, and job J_3 is added to set S, and we continue to position $(j = 2, t = 1)$. At this position, it is in subscript a value 0, so J_2 is tardy and does not belong to set S. Therefore, only index j is decremented, and index t remains the same. So when processing the last job J_1, we move to position $(j = 1, t = 1)$. According to value 1 in the subscript, job J_1 is early and thus belongs to set S. The resulting set of early jobs is $S = \{J_1, J_3, J_4, J_5\}$.

As in the previous examples, we create a schedule from set S by ordering the jobs in set $S = \{J_1, J_3, J_4, J_5\}$ from the earliest to the latest due date which is J_1, J_3, J_4, J_5. In this order, we insert them into the schedule and put J_2 at the end of the schedule. The resulting schedule is shown in Figure 3.6.

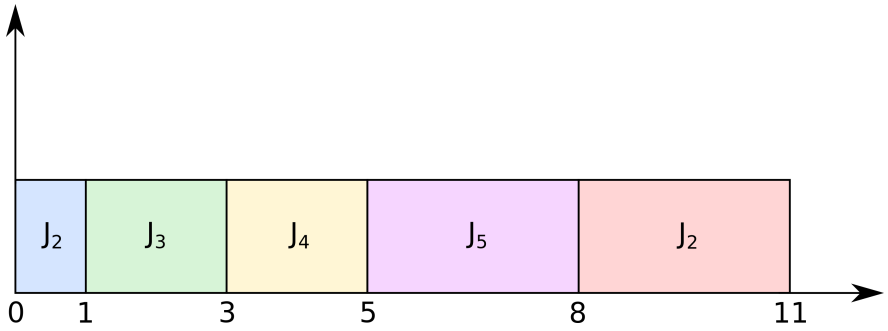

Figure 3.6. The final schedule created from S with the value of the objective function 16.

3.7.4 Special case $1|d_j = d|\sum w_j U_j$

If we add the constraint that all jobs have a common due date d, then we get a problem denoted as $1|d_j = d|\sum w_j U_j$. This problem is equivalent to the 0/1 knapsack problem. The common due date d corresponds to the backpack capacity, processing time p_i of job J_i corresponds to the weight of the item in the 0/1 knapsack problem. Weight w_i of the job J_i corresponds of the increase to the objective function when placing the item in the knapsack.

Based on this context, it is possible to use the knowledge from the 0/1 knapsack problem for the problem $1|d_j = d|\sum w_j U_j$, for example, it is possible to use some approximation algorithms.

Chapter 4
Preliminaries

An advantageous property of the problem $1|\tilde{d}_j|\sum w_jU_j$ ensures that the solution can be expressed by a set of *early jobs* $E \subseteq N$ with the meaning that each $j \in E$ is completed before due date d_j in the corresponding schedule (i.e., $U_j = 0 \iff j \in E$). If job j exceeds the due date in the solution, then $j \in N \setminus E$ and the job is called *tardy job*. Set E defines for each job its maximum completion time $D_j \geq C_j$ as

$$f(x) = \begin{cases} d_j & \text{if } j \in E, \\ \tilde{d}_j & \text{if } j \in N \setminus \text{ E}. \end{cases}$$

The schedule can be constructed by sequencing jobs N in a non-decreasing order of their maximum completion times D_j.

In this thesis, we improve the algorithm presented by the authors of [1]. Their algorithm is based on three essential components, which are a ILP model of the problem, an algorithm computing the upper bound to the objective function based on maximum profit flow problem, and a heuristic algorithm for computing the lower bound to the objective function. Additionally, their algorithm exploits two fundamental theorems capturing the structure of problem $1|\tilde{d}_j|\sum w_jU_j$.

The first theorem is the dominance theorem [1]. It states that in an optimal schedule a job must be early (or tardy) if another job is early (or tardy) provided that certain conditions hold:

Theorem 1. Let $p_i \leq p_j$, $d_i \geq d_j$, $\tilde{d}_i \leq \tilde{d}_j$ and $w_i \geq w_j$, and at least one inequality is strict. Then,

- if job i is tardy, then job j must be tardy too,
- if job j is early, then job i must be early too.

The second theorem is the reduction theorem [1], which is important for reducing the size of an instance. When the algorithm decides that job i is early or tardy, it implies that $D_i = d_i$ for early jobs and $D_i = \tilde{d}_i$ tardy jobs. Then, the theorem defines a *reduced problem* defined by set $N' = N \setminus \{i\}$ as

$$p'_j = p_j,\ w'_j = w_j,\ j \in N'$$

$$d'_j = \begin{cases} \min\{d_j, D_i - p_i\} & \text{if } d_j \leq D_i, \\ d_j - p_i & \text{if } d_j > D_i \end{cases} \quad j \in N',$$

$$\tilde{d}'_j = \begin{cases} \min\{\tilde{d}_j, D_i - p_i\} & \text{if } \tilde{d}_j \leq D_i, \\ \tilde{d}_j - p_i & \text{if } \tilde{d}_j > D_i \end{cases} \quad j \in N'.$$

The theorem allows removing job i from N and solve the reduced problem only, without loosing the optimal solution:

Theorem 2. There exists a feasible schedule with early set E if and only if there exists

a feasible schedule with early set $E' = E \setminus \{i\}$ for the reduced problem.

Therefore, a job that is identified as early or tardy in the algorithm is excluded from N by Theorem 2. Proofs of both theorems can be found in [1].

The algorithm proposed in [1] is a branch-and-bound algorithm with very efficient solution space pruning. Every partial solution of the branch-and-bound algorithm is processed in the following way:

- (Upper bound $\bar{z}$). The algorithm solves the LP relaxation of the problem using a transformation to a maximum profit flow problem. The flow problem is called *relaxed problem* and its solution defining the upper bound on the objective function of the original problem is denoted by $\bar{z}$.
- (Lower bound $\underline{z}$). The heuristic uses the dominance theorem (Theorem 1) to transform the relaxed solution to a feasible solution. The jobs that are not decided by Theorem 1 are resolved by the ILP formulation described in Section 4.1.
- (Fixing of decisions). Subsequently, the algorithm uses variable upfixing techniques from [1] and bounds $\bar{z}$ and $\underline{z}$ to decide whether $i \in E$ or $i \in N \setminus E$.
- (Branching). The algorithm selects job i for which it cannot decide whether it is early or tardy and recursively branches with $i \in E$ and $i \in N \setminus E$.

In the rest of this section, we summarize the key parts of the algorithm proposed in [1]. In the subsequent section, we describe improvements allowing to solve significantly larger instances not only for strongly correlated class of instances but also for uncorrelated and weakly correlated classes.

4.1 ILP model

In the paper [1], the authors introduce a ILP formulation of the problem $1|\tilde{d}_j|\sum w_j U_j$. Their ILP model decides whether job i is in E or $N \setminus E$, therefore, it introduces binary variable x_i which equals one if $i \in E$ and zero otherwise. The ILP formulation is

$$\max_{x} \sum_{i \in N} w_i x_i, \tag{2}$$

subject to

$$\sum_{i \in B_t} p_i + \sum_{i \in C_t} p_i x_i \leq t, \quad t \in T \tag{3}$$

$$x_i \in \{0,1\}, \quad i \in N. \tag{4}$$

Instead of minimizing the weighted number of tardy jobs, the objective (2) maximizes the weighted number of early jobs. To ensure that all jobs are completed before their deadlines with the jobs belonging to set E completed before their due dates, the ILP model contains constraints (3). The constraints are defined over a set of time points $T = \left\{t : t = d_i \vee t = \tilde{d}_i, \forall i \in N\right\}$. For each $t \in T$, the constraints define sets of jobs $B_t = \left\{i \in N : \tilde{d}_i \leq t\right\}$ and $C_t = \left\{i \in N : d_i \leq t \wedge \tilde{d}_i > t\right\}$, i.e., the set of jobs that must be completed before t and the set of jobs that will be early if they are scheduled before t, respectively. The term $\sum_{i \in B_t} p_i$ in (3) represents the sum of processing times of jobs that must be completed within the time t. Conversely, $\sum_{i \in C_t} p_i x_i$ represents the sum of processing times of jobs that can be completed within the time t, i.e., before their

due dates. These jobs may not be completed within time t because their deadlines are after $t \in T$.

As it is pointed out by [1], and our experiments confirm this also, the issue with this ILP formulation is that for large instances it encounters the lack of memory needed to solve it. This is because the constraints (3) contain a large number of non-zero coefficients, where the number increases in the worst-case with $\mathcal{O}(n^2)$. For example, Gurobi ILP solver consumes approximately 8 GB of memory for an instance containing $9,000$ jobs.

The size of ILP model affects all kinds of instances, but for strongly correlated instances its computational complexity becomes a significant problem. For example, Gurobi is unable to solve up to the optimality instances with 200 jobs in a time limit of 1 hour.

4.1.1 Upper bound

The LP relaxation of the ILP formulation (2)–(4) is a crucial part of the algorithm presented by [1] and it is used across the whole algorithm. In addition to being used as an upper bound, it is the basis for the lower bound calculation and is applied to reducing jobs using the variable-fixing technique.

The LP relaxation can be obtained from the ILP formulation by omitting the integer constraint of the x_i variables. Such LP relaxation suffers from the same issue with the model size as the original model. Therefore, the authors of [1] presented a compact linear relaxation formulated as the maximum profit flow problem. The main advantage of this formulation is memory complexity, which is $\mathcal{O}(n)$ compared to $\mathcal{O}(n^2)$ of the LP relaxation. This formulation allowed solving the relaxation even for instances with $50,000$ jobs, which would not possible with the LP model.

The maximum profit flow problem is defined on directed graph $G = (V, A)$, where V is the set of nodes and A is the set of edges. The graph G contains nodes that represent every job $i \in N$ and nodes that represent every time point $t \in T$, i.e., $V = N \cup T$. In the flow problem, time point $t_m \in T$ is used to denote the last time point. The set of edges A is given by three subsets defined as

$$A_d = \{(i, t_k) : i \in N, t_k \in T, t_k = d_i\}, \tag{5}$$
$$A_{\tilde{d}} = \{(i, t_k) : i \in N, t_k \in T, t_k = \tilde{d}_i\}, \tag{6}$$
$$A_T = \{(t_k, t_{k+1}) : t_k \in T \setminus \{t_m\}. \tag{7}$$

A_d is a set of edges between nodes representing jobs i and its due dates d_i represented by time point $t_k = d_i$. The same applies to the set $A_{\tilde{d}}$, where the edges lead to nodes $t_k = \tilde{d}_i$ representing deadlines. Edges between two nodes representing successive time points corresponds to set A_T.

In a solution, each edge $(i, d) \in A = A_d \cup A_{\tilde{d}} \cup A_T$ is associated with a flow $y_{i,d}$ representing a portion of a processing time executed in a specific time interval defined by G. The problem relaxation can be expressed as the maximum profit flow in $G = (V, A)$ as

$$\max_{\boldsymbol{y}} \sum_{(i,d_i) \in A_d} \frac{w_i}{p_i} y_{i,d_i}, \tag{8}$$

subject to

$$y_{i,d_i} + y_{i,\tilde{d}_i} = p_i, \quad i \in N, \tag{9}$$

$$y_{t_1,t_2} - \sum_{(i,t_1)\in A_d \cup A_{\tilde{d}}} y_{i,t_1} = 0, \tag{10}$$

$$y_{t_k,t_{k+1}} - y_{t_{k-1},t_k} - \sum_{(i,t_k)\in A_d \cup A_{\tilde{d}}} y_{i,t_k} = 0, \quad k \in \{2,\ldots,m-1\}, \tag{11}$$

$$y_{t_{m-1},t_m} + \sum_{(i,t_m)\in A_d \cup A_{\tilde{d}}} y_{i,t_m} = \sum_{i\in N} p_i, \tag{12}$$

$$y_{t_k,t_{k+1}} \leq t_k, \quad (t_k,t_{k+1}) \in A_T, \tag{13}$$

$$y_{i,d} \geq 0, \quad (i,j) \in A. \tag{14}$$

Each node representing job $i \in N$ is a source of flow in G. The requested inflow of source i equals to processing time p_i, which is defined by constraints (9). Conversely, the node representing the largest time point t_m is a sink in G. The requested outflow equals the sum of all jobs processing times, which is stated by the constraint (12). The constraints (13) sets edge capacity between two different time points to the value of the time point from which the edge arises and thus defines the capacity of the machine to be one. Finally, constraints (10) – (11) ensure that Kirchhoff's law applies for nodes representing time points, and non-negativity of flow is provided by constraints (14).

The formulation can be solved with a maximum profit (min cost) flow algorithm. We will denote the values of variables in an optimal relaxed solution as $\hat{y}_{i,d}$.

Chapter 5
Improved algorithm

In this chapter, we build on the algorithm introduced by [1]. Their algorithm is a state-of-the-art algorithm for solving both problems $1|\tilde{d}_j|\sum w_jU_j$ and $1||\sum w_jU_j$. The algorithm allows for the problem $1|\tilde{d}_j|\sum w_jU_j$ to solve uncorrelated instances up to $30,000$ jobs and weakly correlated instances up to $5,000$ jobs. On the other hand, strongly correlated instances are very difficult for the original algorithm. For example, in case of strongly correlated instances the original algorithm cannot solve some instances with just 200 jobs within one hour time limit.

We introduce three enhancements of the algorithm that enables solving strongly correlated instances of up to $10,000$ jobs compared to the original algorithm, which cannot solve all instance of the size of 200 jobs. The most significant enhancement is a modified ILP formulation, which allows performing a decomposition by the number of early jobs and a method for efficient solution of this decomposed model. The other two improvements relate to the computation of the lower bound and upper bound.

5.1 Improved ILP model

In the available literature, strongly correlated instances are referred to as instances where $\forall i \in N : w_i = p_i + 20$. In this thesis, we use this name for instances that meet $w_i = p_i + C$, where $C \geq 0$. We have performed extensive experiments to investigated whether the C value affects the complexity of an instance, measured in terms of the time needed to solve an instance with the ILP solver. Our experiments with ILP formulation (2)–(4) revealed an anomaly that occurs for $C = 0$. Instances where $C = 0$ (i.e., $w_i = p_i$) are significantly easier than instances that have $C > 0$, such as the widely used $C = 20$ (i.e., $w_i = p_i + 20$). The time needed for solving instances with $C = 0$ is comparable to uncorrelated instances. In contrast, the same ILP solver is not able to solve some instances having 200 jobs with $C = 20$ to optimum within one hour. With this observation, we propose simple but yet powerful reformulation of the original model.

5.1.1 Model reformulation

The difference between instances with $C = 0$ and instances with $C > 0$ is that in the second case the weight parameter of the job is the same as the execution time of the job ($w_i = p_i$). Thus, the objective function in this case can be written as $\max \sum_{i \in N} p_i x_i$. Base on that, we created a new ILP model for strongly correlated instances. This new ILP model is based on expressing solution for strongly correlated instances with $C > 0$, similarly to strongly correlated instances with $C = 0$.

The new ILP model takes over constraints (3) and (4) from the original ILP model. In addition, we introduce variable e, which specifies the number of early jobs via constraint (17). Using variable e, one can reformulate the original objective function for strongly correlated instances as $\max \sum_{i \in N} w_i x_i = \max \sum_{i \in N} (p_i + C)x_i =$

$\max C \cdot e + \sum_{i \in N} p_i x_i$. Then, the new formulation is stated as

$$\max_{e \in \{0,\ldots,n\}} \max_{\boldsymbol{x}} C \cdot e + \sum_{i \in N} p_i x_i, \tag{15}$$

subject to

$$\sum_{i \in B_t} p_i + \sum_{i \in C_t} p_i x_i \leq t, \quad t \in T, \tag{16}$$

$$\sum_{i \in N} x_i = e, \tag{17}$$

$$x_i \in \{0,1\}, \quad i \in N. \tag{18}$$

The advantage of the new ILP model is that it allows decomposition according to variable e. If we select some specific value of variable e, then the objective function can be written as $\max \sum_{i \in N} p_i x_i + C \cdot e$. The expression $C \cdot e$ is a constant and therefore only $\max \sum_{i \in N} p_i x_i$ can to be considered. The constant expression $C \cdot e$ is added after the solution is found. Thus, for each value of e and $C > 0$, we have a ILP model very similar to the one with $C = 0$. The decomposed ILP model differs in that it has one extra constraint (17) to limit the number of early jobs.

Our experiments confirm that if we decompose according to variable e and select a specific number of early jobs, the ILP solver for this selected number of early jobs is able to find a solution even for instances with $5,000$ jobs.

When decomposing the new ILP model according to variable e, up to n ILP models with different numbers of early jobs (variable e) may emerge. We need to solve in the worst case all of them to ensure that the optimal solution for the original model is found. In order to reduce the number of ILP models introduced by the decomposition, we have created a technique that accurately estimates the candidate range of the number of early jobs. This technique is described in the following subsection.

5.1.2 Problem decomposition

This subsection describes decomposition of the reformulated model based on fixing the number of early jobs e. This approach is based on an observation that LP relaxation of the original ILP model, even for instances with $5,000$ jobs, has a very low number of non-integer values in the optimal solution. The same property holds for the relaxed problem, described in Section 4.1.1, where the majority of $\hat{y}_{i,d}$ are equal to p_i or to zero. Therefore, the number of early jobs (having $\hat{y}_{i,d} = p_i$) or partly early (having $0 < \hat{y}_{i,d} < p_i$) provides a very accurate estimate on the number of early jobs $e = |E|$ in the optimal solution, and we utilize this information to solve the decomposed ILP model faster.

First, it is required to solve the relaxed problem (8) – (14). Then, we denote by $\underline{e}$ the number of early jobs having $\hat{y}_{i,d} = p_i$ in the relaxed solution. We denote by $\bar{e}$ the maximum number of early jobs. For computing the maximum number of early jobs, we use problem $1|\tilde{d}_j|\sum U_j$, which is a relaxation of the problem $1|\tilde{d}_j|\sum w_j U_j$, and it is easy to solve it using ILP solver. This ensures that no more than $\bar{e}$ early jobs can be.

With the values $\underline{e}$ and $\bar{e}$, we define a set $K = \{\underline{e}, \underline{e}+1, \ldots, \bar{e}\}$ of likely candidates for the number of early jobs $|E|$ in an optimal solution. Subsequently, for each $e_k \in K$, we solve the modified ILP model with the constraint $\sum_{i \in N} x_i = e_k$. Since it is not guaranteed that the number of early jobs of the optimal solution will fall within set K,

one additional ILP model is solved separately. This model have replaced constraint (17) with the boundary constraint $\sum_{i \in N} x_i \leq \underline{e} - 1$. This ensures that the optimal solution is not omitted even with a poor choice of K. The optimal solution is then obtained as the maximum over all optimal solutions of ILP models with a fixed value of e_k and the one model with the boundary constraints. We note that since the relaxed solution has typically a small number of non-integer values, the size of K is typically very small.

The efficiency of the decomposition algorithm is further improved by using the maximum value of the objective function of ILP models solved with previous values of $e_k \in K$ (including the boundary case) as the *cut up* parameter, i.e., the solver is told to search only for solutions with objectives better than the cut up. Hence, often it is not needed to thoroughly solves model with boundary condition as they are quickly cut off by an existing solution within K. Our experiments demonstrated that in most cases, the number of early jobs is indeed within the set K. Only in a very few rare cases, the optimal number of early jobs lies outside of K.

5.2 Improved lower bound

The tightness of the lower bound (and upper bound as well) has a crucial impact on the size of set N. The lower bound is used in the variable fixing technique which together with the reduction theorem (Theorem 2) allows to reduce size of jobs N which has to be solved. When it comes to solving strongly correlated instances of large sizes, the original algorithm of [1] fixes significantly fewer variables than in the case of uncorrelated instances. The smaller number of fixed variables results in extensive branching in the branch-and-bound method and the explosion of run time. Hence, the goal is to improve the lower bound in order to fix more variables and to prevent extensive branching.

The heuristic computing the lower bound proposed in [1] is based on the solution of the relaxed problem (8) – (14). According to the solution, the set of job N is split into a set of jobs L that will be optimally scheduled by (2) – (4), and the other jobs $N \setminus L$ are scheduled heuristically using the information from the solution of the relaxed problem. The quality of the heuristics depends on the jobs selected into set L. In [1], they propose that all jobs with $0 < \hat{y}_{j,d} < p_j$ are inserted into set L. Furthermore, they include all jobs j that are tardy (i.e., $\hat{y}_{j,d} = 0$) for which there is no job i dominating the job j by the dominance Theorem 1. Similarly, their set L includes all jobs j that are early (i.e., $\hat{y}_{j,d} = p_j$) for which there is no job i that is dominated by job j according to the same theorem.

Our approach uses the same structure but has three main differences. The first one is that the jobs in set L are scheduled by the decomposed ILP model from Section 5.1. The second one is a different definition of L. We have carried out several experiments to investigate which jobs must be in L in order to find a tighter lower bound. The results showed that the heuristics provides much tighter bound if the rule for adding early jobs to set L is modified to the following: set L includes all jobs j that are early (i.e., $\hat{y}_{j,d} = p_j$) for which there is no job i dominating the job j by the dominance theorem. In the other words, the condition was reversed. Set L also includes all jobs j that are tardy (i.e., $\hat{y}_{j,d} = 0$) for which there is no job i dominating the job j by the dominance theorem and all jobs with $0 < \hat{y}_{j,d} < p_j$, which is the same with the original heuristic. The last difference is that we do not use the additional local search. In the original heuristics, the local search neighborhood is defined as $\sum_{i \in N} |\bar{x}_i - \tilde{x}_i| = 2$, where $\bar{x}_i$ represents an original solution, and $\tilde{x}_i$ represents a new solution. This neighborhood

allows swapping only for pairs of jobs, which one of them is early and other is tardy. In our improved heuristic, this local search led only rarely to an improvement.

This adjustment leads to an improvement in the lower bound heuristic, which is significantly tighter than the original one. Modifying this rule increases the size of set L by approximately by 13%. The increase of the CPU time introduced by the larger job set L is compensated by leaving out the additional local search.

5.3 Improved upper bound

An another way to reduce the number of jobs with Theorem 2 and the variable fixing technique is to improve the upper bound. In paper [1], the authors use continuous relaxation of the ILP model as the upper bound. This relaxation is calculated using the formulation as a maximum profit flow problem. Although this upper bound is relatively tight, we present its improvement by a limited branching procedure.

In our approach, we first solve the problem relaxation (8) – (14). Then we tight up the upper bound obtained from the problem relaxation by branching on a selected job. In every node of this branch and bound procedure, we select job $i = \arg_{i' \in N} \max\{p_{i'} : 0 < \hat{y}_{i',d} < p_{i'}\}$, i.e., a job with maximal processing time that has non-integer variable value. For this job we assume two cases: (i) the job is early ($\hat{y}_{i,d} = p_i$), and (ii) is tardy ($\hat{y}_{i,d} = 0$). For each case, we solve the relaxed problem while fixing the flow $y_{i,d}$ accordingly, and apply the procedure for each branch again. The branching is repeated up to a small fixed depth of the branching tree. Finally, from all solutions in leaves of the limited branching tree, the algorithm selects the one with the highest objective value. This solution is then used as a new upper bound $\bar{z}$.

The limited branch-and-bound procedure selects jobs with the smallest deadline for instances with deadlines and the largest due date for instances without deadlines for two reasons. First, it is computationally cheap rule, and second, one can expect that the job with the earliest deadline(resp. the latest due date) can have the most significant impact on the objective function. Even though this method has increased computational complexity over solving just single relaxation, overall, it paid off. The main reason is the same as in lower bound computation. Tighter lower and upper bounds we provide to the variable fixing technique, the fewer nodes it is necessary to explore in the main branch-and-bound method. Especially for strongly correlated instances, a large number of explored nodes leads to time outs.

Chapter 6
Experiments

To compare the performance of the improved algorithm for all classes of instances with the original algorithm, we present the results of several performed experiments. All experiments are performed on a computer containing two Intel Xeon E5-2690 v4 CPUs with 512GB RAM running CentOS Linux 7. Both algorithms are implemented in C++ with Gurobi 8.1.1 ILP solver. For solving the minimum cost flow problem (maximum profit flow problem) our algorithm uses a dual ascent method RelaxIV [23], specifically its implementation in the MCFClass project [24], which proved to be the most suitable for implementation of both algorithms. The source code of our algorithm is available on the attached CD.

The algorithm was tested with the following parameters. The maximum depth of the branch and bound tree in the improved procedure computing the upper bound was 8. The threshold for solving instances directly by the ILP solver was $1.4 \cdot 10^7$ non zero elements in the ILP matrix of constraints, which is the same as used by Baptiste [1], and it corresponds to instances with approximately $4,000$ jobs.

This section has six parts. First, we describe the way we generate benchmark instances. Then we benchmark ILP problem formulations, i.e., ILP formulation from [1] and the formulation from Section 5.1. Our algorithm is compared with the algorithm published in [1] in Section 6.3.1, and individual improvements used in our algorithm are benchmarked in the subsequent section. The last section presents the experimental results on the heaviest problem instances defined in [1].

6.1 Instances generation

To compare the results of our experiments with the paper [1], we generate instances using the same method. For each job i, the processing time p_i is an integer randomly drawn from the interval $[1, 100]$. The weight for each job is created based on the instance type as follows:

- In the case of non-correlated instances, for each job i, the weight w_i is an integer randomly drawn from interval $[1, 100]$;
- In the case of weakly correlated instances, for each job i, the weight w_i is an integer randomly drawn from interval $[p_i, p_i + 20]$;
- In the case of strongly correlated instances, each job i has its weight $w_i = p_i + 20$ if the $C = 20$ or $w_i = p_i$ if $C = 0$, which is specified for each experiment. The value of $C = 20$ is chosen for the comparability of results with paper [1].

Let us denote $P = \sum_{i \in N} p_i$. Furthermore, we define a set of parameters $D = \{0.1, 0.3, 0.5, 0.7, 0.9\}$ which is used to create pairs $(u, v) \in D \times D$ that meet the condition $u < v$, which are used for generating due dates. Then, for each job i, and the selected pair (u, v) due date d_i is an integer randomly drawn from interval $[P \cdot u, P \cdot v]$. For instances in which jobs contain deadlines, a deadline $\bar{d}_i$ is an integer randomly drawn from interval $[d_i, P \cdot 1.1]$.

The benchmark instances are created such the for a given n there are always 20 randomly generated instances for each pair (u, v), i.e., 200 instances for the given n. All generated instances are guaranteed to be feasible. Feasibility of each generated instance assuming deadlines is tested by omitting the objective function and solving the feasibility problem by the earliest deadline first rule. Infeasible instances are disregarded as in paper [1].

All the experiments described below assume time limit 3600 seconds to solve one problem instance. The ILP solver is allowed to use only one physical CPU core. The reason why we limit the number of CPU cores is that in paper [1] the algorithm was benchmarked on a computer with a single core as well.

6.2 Comparison of ILP models

The following experiments analyse the runtime required to solve the original ILP model (2)–(4) depending on the size of the instance and its type. In the experiment, we focus on three different types of instances, which are non-correlated instances, strongly correlated instances with a constant $C = 20$, and strongly correlated instances with a constant $C = 0$; all cases without deadlines. We present only results on instances without deadlines since results on instances with deadlines are very similar. The last experiment shows the performance of the ILP model proposed in this thesis and compares it with the original one.

The results related to the original ILP model are summarized in tables 6.1 – 6.3. This data is primarily used to demonstrate the difficulty of strongly correlated instances versus non-correlated instances. In each table, the `unsolved` column indicates the number of instances that are not solved to the optimum within the time limit. The `avg` and `max` columns indicate the average and maximum runtime required to solve an instance to the optimum. Instances that are not solved to the optimum within the time limit are also included in these columns. These instances have the runtime equal to the time limit.

	CPU Time		
n	avg [s]	max [s]	unsolved [-]
50	0.02	0.12	0
100	0.04	0.16	0
150	0.07	0.22	0
200	0.11	0.31	0
250	0.16	0.37	0
500	0.61	1.29	0
1000	2.84	5.90	0
2000	11.70	23.84	0
3000	27.04	61.93	0
4000	48.63	126.78	0
5000	77.62	206.10	0

Table 6.1. Result computed using the original ILP model for non-correlated instances.

Table 6.1 contains the measured results for non-correlated instances. The results concerning strongly correlated instances with $C = 20$ and $C = 0$ are shown in tables 6.2 and 6.3, respectively.

	CPU Time		
n	avg [s]	max [s]	unsolved [-]
50	0.03	0.31	0
100	1.42	221.72	0
150	32.18	3600.00	1
200	174.81	3600.00	6
250	444.72	3600.00	22
500	579.14	3600.00	30
1000	1153.90	3600.00	63
2000	1123.90	3600.00	61
3000	1328.78	3600.00	72
4000	1642.63	3600.00	90
5000	1586.28	3600.00	86

Table 6.2. Result computed using the original ILP model for strongly correlated instances with $C = 20$..

When comparing the results in tables 6.1 and 6.2, we can observe a very significant difference in the runtime required to solve the instance to the optimum between strongly correlated instances with $C = 20$ in Table 6.2 and non-correlated instances in Table 6.1. We can also observe that for non-correlated instances, all instances up to $5,000$ jobs, are solved within the time limit. In contrast, for strongly correlated instances with $C = 20$, it can be seen that even instances with 150 jobs are not all solved within the time limit.

	CPU Time		
n	avg [s]	max [s]	unsolved [-]
50	0.01	0.09	0
100	0.03	0.13	0
150	0.05	0.16	0
200	0.09	0.23	0
250	0.13	0.33	0
500	0.42	1.21	0
1000	1.87	5.19	0
2000	7.45	12.68	0
3000	16.19	28.39	0
4000	28.97	74.33	0
5000	45.55	76.11	0

Table 6.3. Result computed using the original ILP model for strongly correlated instances with $C = 0$..

Conversely, if we compare the results in tables 6.2 and 6.3, we can observe that strongly correlated instances with $C = 0$ show an anomaly. This anomaly is that these instances are significantly easier to solve than other strongly correlated instances with $C > 0$. Even if we compare the results in Table 6.3 with the measured data for non-correlated instances in Table 6.1, we can observe that strongly correlated instances with $C = 0$ are even easier to solve to the optimum than non-correlated instances in the sense of runtime. The average runtime required to find the optimal solution for strongly correlated instances with $C = 0$ is, on average, about 60% of the time required for non-correlated instances. These observed anomalies for strongly correlated instances with

$C = 0$ are crucial for the development of the improved ILP model, which is explained in detail in Section 5.1.

The property of the studied problem illustrated in Table 6.3 was used to propose the improved ILP model, described in Section 5.1. Therefore, the next experiment demonstrates the capabilities of the improved ILP model without using other parts of the algorithm. This experiment was performed on strongly correlated instances with $C = 20$, which are the same as the instances used to measure the results for the original ILP model in Table 6.2. The results of this experiment are in Table 6.4. The table shows that our modified ILP model can solve all these instances to optimum within $3,600$ seconds. On the contrary, the original ILP model does not solve all these instances to the optimum within the time limit, even for the instances with 150 jobs. When we compare results in tables 6.2 and 6.4, we can see that the maximum run times for solved instances from size 100 to $3,000$ are significantly smaller for our improved ILP model.

	CPU Time		
n	avg [s]	max [s]	unsolved [-]
50	0.10	0.40	0
100	0.17	1.17	0
150	0.28	1.42	0
200	0.37	3.00	0
250	0.48	1.80	0
500	1.82	11.65	0
1000	9.01	64.01	0
2000	46.79	423.79	0
3000	153.44	1173.26	0
4000	285.79	2567.63	0
5000	432.04	3022.05	0

Table 6.4. Result computed using our new ILP model for strongly correlated instances with $C = 20$.

6.3 Comparison of full algorithms

The next set of experiments focuses on comparing the original algorithm and our improved algorithm. The experiments are primarily focused on strongly correlated instances with $C = 20$ with and without deadlines. Nevertheless, we also provide experimental results on weakly and non-correlated correlated instances with deadlines.

In each table presented in this section, the column "Nodes" indicates the number of visited nodes in the branch-and-bound method. The column `CPU time` indicates the runtime required to solve an instance. Column "root-GAP" indicates the relative difference between the objectives of the upper bound and lower bound algorithms. The value is computed as $\frac{UB-LB}{LB} \cdot 100$. Average and maximum values of this indicator are in percentages. The column `Unsolved` indicates the number of instances that are not solved to the optimum within the time limit of $3,600$ seconds. In tables 6.6 and 6.8 column "Red. prob. size" indicates the size of the reduced instance, i.e., the number of jobs after the application of the reduction theorem in the root node.

6.3.1 Strongly correlated instances with deadlines

The first part of the experiments is focused on strongly correlated instances with deadlines. Tables 6.5 and 6.7 summarize the results of experiments focused on the relation-

ship between the size of the instance and the time required to find the optimal value for both algorithms. On the other hand, tables 6.6 and 6.8 demonstrate the relationship between a pair of parameters (u, v) and the time required to find the optimal value for the largest instances for which our improved algorithm is able to solve the most of the instances within the time limit $3,600$ seconds.

		CPU Time		Nodes		root-GAP	
n	unsolved [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
1000	18	330.91	3600.00	1.00	1.00	0.0284	0.1196
2000	29	558.35	3600.00	1.00	1.00	0.0140	0.0789
3000	47	867.64	3600.00	1.00	1.00	0.0092	0.0345
4000	44	849.17	3600.00	1.00	1.00	0.0066	0.0299
5000	47	875.12	3600.00	1.06	7.00	0.0053	0.0239
6000	49	918.11	3600.00	5.96	477.00	0.0044	0.0150
7000	68	1249.09	3600.00	42.23	2236.00	0.0038	0.0215
8000	71	1318.14	3600.00	60.98	1921.00	0.0033	0.0132
9000	97	1775.35	3600.00	114.42	1987.00	0.0031	0.0135
10000	103	1870.18	3600.00	141.26	2696.00	0.0027	0.0103

Table 6.5. Results for the strongly correlated instances with $C = 20$ and deadlines solved using the original algorithm.

			Red. prob. size		CPU Time		Nodes		root-GAP	
u	v	unsolved [-]	avg [-]	max [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
0.1	0.3	8	268.28	5241	1487.76	3600.00	7.75	98.00	0.0115	0.0150
0.1	0.5	16	217.18	4257	2893.31	3600.00	1.10	3.00	0.0057	0.0088
0.1	0.7	16	248.00	3056	2990.03	3600.00	1.00	1.00	0.0040	0.0083
0.1	0.9	4	62.72	2222	743.56	3600.00	1.00	1.00	0.0024	0.0052
0.3	0.5	1	64.00	5213	224.44	3600.00	25.90	477.00	0.0049	0.0097
0.3	0.7	2	142.37	4256	377.63	3600.00	1.15	4.00	0.0038	0.0070
0.3	0.9	0	37.63	2757	7.64	16.61	1.00	1.00	0.0025	0.0053
0.5	0.7	1	120.20	4811	247.01	3600.00	18.70	353.00	0.0034	0.0062
0.5	0.9	0	149.00	3054	9.49	16.52	1.00	1.00	0.0029	0.0048
0.7	0.9	1	192.35	4168	200.28	3600.00	1.00	1.00	0.0037	0.0052

Table 6.6. Detailed results for the strongly correlated instances with $C = 20$ and deadlines with $6,000$ jobs, solved using the original algorithm.

From the results in tables 6.5 and 6.7, it can be seen that the original algorithm has a problem to solve to optimum all strongly correlated instances with $1,000$ jobs. On the other hand, the algorithm with improvements presented by us can solve to optimum instance having up to $5,000$ jobs and the most of the instances with $6,000$ jobs. For the remaining sizes of instances, the improved algorithm solves significantly more instances than the original algorithm.

Comparing the results in tables 6.6 and 6.8, it is apparent that instances with a pair of parameters $(0.1, 0.3)$ and $(0.1, 0.5)$ are the most difficult instances, because the most unresolved instances come from these two pairs of parameters, and also for these pairs, the instance sizes are the least reduced (see column "Red. prob. size"). This is consistent with the result in the Baptiste et al. paper, where this was observed for non-correlated and weakly correlated instances.

		CPU Time		Nodes		root-GAP	
n	unsolved [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
1000	0	1.57	16.15	1.00	1.00	0.0255	0.1087
2000	0	3.93	46.52	1.00	1.00	0.0133	0.0604
3000	0	7.38	48.37	1.00	1.00	0.0090	0.0357
4000	0	14.33	332.31	1.00	1.00	0.0067	0.0297
5000	0	17.53	94.21	1.05	5.00	0.0053	0.0239
6000	10	218.03	3600.00	20.84	1165.00	0.0045	0.0176
7000	32	598.27	3600.00	55.66	1540.00	0.0038	0.0148
8000	38	733.45	3600.00	89.36	1128.00	0.0033	0.0172
9000	64	1189.44	3600.00	140.18	1444.00	0.0030	0.0122
10000	68	1255.93	3600.00	164.66	2007.00	0.0027	0.0103

Table 6.7. Results for the strongly correlated instances with $C = 20$ and deadlines solved using the algorithm with all improvements.

			Red. prob. size		CPU Time		Nodes		root-GAP	
u	v	unsolved [-]	avg [-]	max [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
0.1	0.3	4	214.90	5392	844.67	3600.00	96.75	1165.00	0.0113	0.0176
0.1	0.5	4	119.40	4707	808.24	3600.00	68.75	464.00	0.0066	0.0107
0.1	0.7	0	182.40	3666	42.55	308.54	1.00	1.00	0.0042	0.0077
0.1	0.9	0	59.42	2363	9.24	16.27	1.00	1.00	0.0024	0.0052
0.3	0.5	1	63.60	5213	221.73	3600.00	22.15	406.00	0.0045	0.0087
0.3	0.7	0	142.37	4115	18.99	36.62	1.00	1.00	0.0034	0.0063
0.3	0.9	0	38.28	2757	8.53	20.14	1.00	1.00	0.0024	0.0051
0.5	0.7	1	120.20	4811	196.16	3600.00	14.70	275.00	0.0032	0.0062
0.5	0.9	0	149.00	3054	11.49	18.50	1.00	1.00	0.0029	0.0048
0.7	0.9	0	192.35	4168	18.70	31.28	1.00	1.00	0.0037	0.0052

Table 6.8. Detailed results for the strongly correlated instances with $C = 20$ and deadlines with $6,000$ jobs, solved using the algorithm with all improvement.

These pairs of parameters also have their root-GAP larger than the other pairs, and as a consequence of this, there is little ability to reduce the instance size by the variable fixing technique. This also leads to a large number of nodes being visited in the branch-and-bound method because the instances cannot be reduced to the size, which is solved directly using the ILP solver. These experiments also show that the variables fixing technique is not as effective with strongly correlated instances as with non-correlated instances as will be shown in Subsection 6.3.4.

6.3.2 Strongly correlated instances without deadlines

The second part of the experiments is focused on strongly correlated instances without deadlines. As in the previous experiments, tables 6.9 and 6.11 summarize the results of experiments analyzing the relationship between the instance size and time required to find the solution. Tables 6.10 and 6.12 demonstrate the relationship between a pair of parameters (u, v) and the time required to find the solution for the largest instances that our improved algorithm is able to solve within the time limit.

When we compare the results in tables 6.9 and 6.11 with the tables from the previous subsection, it can be observed that instances without deadlines are easier to solve to the optimum than instances with deadlines even for the original ILP model. This can

		CPU Time		Nodes		root-GAP	
n	unsolved [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
1000	0	0.51	6.38	1.00	1.00	0.0228	0.0848
2000	1	23.74	3600.00	1.00	1.00	0.0121	0.0395
3000	2	55.38	3600.00	1.00	1.00	0.0077	0.0319
4000	0	13.23	1176.60	1.00	1.00	0.0058	0.0183
5000	1	41.49	3600.00	1.00	1.00	0.0046	0.0177
6000	21	389.15	3600.00	42.27	574.00	0.0037	0.0125
7000	61	1090.58	3600.00	193.48	3389.00	0.0033	0.0135
8000	75	1341.55	3600.00	224.56	2358.00	0.0028	0.0114
9000	79	1433.09	3600.00	288.55	1875.00	0.0025	0.0110
10000	80	1453.85	3600.00	277.75	1556.00	0.0022	0.0100

Table 6.9. Results for the strongly correlated instances with $C = 20$ and without deadlines, solved using the original algorithm.

			Red. prob. size		CPU Time		Nodes		root-GAP	
u	v	unsolved [-]	avg [-]	max [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
0.1	0.3	3	144.16	5961	549.05	3600.00	65.90	499.00	0.0081	0.0125
0.1	0.5	6	292.74	5999	1091.57	3600.00	92.80	412.00	0.0053	0.0098
0.1	0.7	2	226.47	5811	370.88	3600.00	36.50	444.00	0.0037	0.0062
0.1	0.9	2	39.32	5998	366.30	3600.00	55.10	574.00	0.0023	0.0051
0.3	0.5	0	176.15	5191	11.40	23.84	1.00	1.00	0.0039	0.0085
0.3	0.7	2	220.67	5657	367.89	3600.00	34.50	388.00	0.0031	0.0059
0.3	0.9	2	90.10	5517	366.05	3600.00	39.75	419.00	0.0025	0.0048
0.5	0.7	2	162.85	5743	372.75	3600.00	45.10	451.00	0.0036	0.0061
0.5	0.9	1	125.17	5656	186.19	3600.00	25.40	487.00	0.0020	0.0048
0.7	0.9	1	157.26	5750	209.47	3600.00	26.60	475.00	0.0025	0.0050

Table 6.10. Detailed results for the strongly correlated instances with $C = 20$ and without deadlines with $6,000$ jobs, solved using the original algorithm.

		CPU Time		Nodes		root-GAP	
n	unsolved [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
1000	0	0.59	1.53	1.00	1.00	0.0218	0.0830
2000	0	2.40	41.45	1.00	1.00	0.0118	0.0377
3000	0	4.18	12.80	1.00	1.00	0.0076	0.0319
4000	0	7.44	22.34	1.00	1.00	0.0058	0.0183
5000	0	10.15	38.47	1.00	1.00	0.0045	0.0177
6000	21	392.34	3600.00	32.97	421.00	0.0037	0.0125
7000	61	1105.89	3600.00	162.75	2409.00	0.0033	0.0135
8000	75	1358.59	3600.00	201.24	1501.00	0.0028	0.0114
9000	79	1430.10	3600.00	236.11	1285.00	0.0025	0.0092
10000	80	1447.29	3600.00	226.62	1110.00	0.0022	0.0100

Table 6.11. Results for the strongly correlated instances with $C = 20$ and without deadlines, solved using the algorithm with all improvements.

be seen from a smaller number of instances remaining unsolved within a given time limit by both the original and the improved algorithm. This is because the absence of deadlines makes the ILP model smaller than in the case with deadlines.

			Red. prob. size		CPU Time		Nodes		root-GAP	
u	v	unsolved [-]	avg [-]	max [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
0.1	0.3	3	144.16	5961	552.74	3600.00	50.10	398.00	0.0081	0.0125
0.1	0.5	6	292.74	5999	1090.10	3600.00	83.30	321.00	0.0052	0.0098
0.1	0.7	2	226.47	5811	376.10	3600.00	36.45	357.00	0.0037	0.0062
0.1	0.9	2	39.32	5998	369.31	3600.00	40.30	421.00	0.0023	0.0051
0.3	0.5	0	176.15	5191	11.01	31.03	1.00	1.00	0.0039	0.0085
0.3	0.7	2	220.67	5657	370.72	3600.00	27.55	316.00	0.0031	0.0059
0.3	0.9	2	90.10	5517	369.46	3600.00	24.85	269.00	0.0025	0.0048
0.5	0.7	2	162.85	5743	371.01	3600.00	32.00	317.00	0.0036	0.0061
0.5	0.9	1	125.17	5656	189.23	3600.00	16.75	314.00	0.0020	0.0048
0.7	0.9	1	157.26	5750	223.74	3600.00	17.35	290.00	0.0025	0.0050

Table 6.12. Detailed results for the strongly correlated instances with $C = 20$ and without deadlines with $6,000$ jobs, solved using the algorithm with all improvement.

From the tables, we can see that the improved algorithm solved all instances of up to $6,000$ jobs and also reduced the average time required to solve an instance for instances of up to this size. In contrast, for instances bigger than $6,000$ jobs, the number of solved instances is the same. This is most likely because some instances after reduction have almost the same size as the original instances, which can be seen in Table 6.12. As a result, the branch-and-bound method searches a large number of nodes. Also, improved lower bound and upper bound for strongly correlated instances without deadlines improve root-GAP only slightly. Also, when comparing Table 6.12 and Table 6.8, we can observe that the maximum size of the reduced instance is smaller for instances with deadlines, compared to instances without deadlines.

It should also be pointed out that the original algorithm uses the ILP solver for instances having approximately less than $4,000$ jobs. Thus, if an optimal solution is not found for these instances within the time limit, this is not due to the algorithm itself, but because the ILP solver is not able to find the optimal solution within the time limit. It is the same for instances with or without deadlines.

6.3.3 Weakly correlated instances with deadlines

In this subsection, we present results comparing the original algorithm to the algorithm with improvement lower bound and upper bound on weakly correlated instances with deadlines. Both improved lower bound and improved upper bound work for all types of instances. For the weakly correlated instances, the original ILP model was used because, although the improved ILP model allows solving this type of instance, it needs to be solved several times due to decomposition according to the number of early jobs e. In the case of weakly correlated instances, solving the improved ILP module requires more time than the original ILP model.

When we compare the result from tables 6.13 and 6.14, we can observe that in the case of weakly correlated instances, both algorithms solved all instances up to the size of $15,000$ jobs. In particular, we can see that our improved algorithm achieves a significant decrease in root-GAP compared to the original algorithm. As a result, both the average time required to solve an instance, and the average number of visited nodes significantly decrease. A decrease in the number of visited nodes is particularly noticeable on instances with a large number of jobs.

		CPU Time		Nodes		root-GAP	
n	unsolved [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
1000	0	1.19	10.85	1.00	1.00	0.0115	0.0783
2000	0	2.83	22.47	1.00	1.00	0.0059	0.0468
3000	0	5.65	57.17	1.00	1.00	0.0038	0.0263
4000	0	9.79	89.65	1.00	1.00	0.0031	0.0220
5000	0	12.95	138.93	1.00	1.00	0.0023	0.0155
6000	0	20.29	212.91	1.03	3.00	0.0020	0.0142
7000	0	24.51	251.34	1.03	3.00	0.0016	0.0120
8000	0	33.12	576.17	1.05	3.00	0.0016	0.0089
9000	0	44.38	1033.08	1.12	13.00	0.0013	0.0117
10000	0	45.27	282.67	1.07	7.00	0.0010	0.0069
15000	0	114.13	2606.64	1.79	101.00	0.0007	0.0044

Table 6.13. Results for the weakly correlated instances with deadlines solved using the original algorithm.

		CPU Time		Nodes		root-GAP	
n	unsolved [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
1000	0	0.99	7.61	1.00	1.00	0.0061	0.0696
2000	0	2.27	21.44	1.00	1.00	0.0034	0.0463
3000	0	4.09	30.63	1.00	1.00	0.0024	0.0190
4000	0	6.39	77.99	1.00	1.00	0.0019	0.0155
5000	0	9.36	183.25	1.00	1.00	0.0015	0.0116
6000	0	12.60	172.04	1.03	3.00	0.0014	0.0142
7000	0	14.98	90.71	1.01	3.00	0.0011	0.0106
8000	0	19.40	254.58	1.04	3.00	0.0011	0.0089
9000	0	23.72	253.51	1.03	3.00	0.0009	0.0098
10000	0	25.01	209.48	1.04	3.00	0.0007	0.0066
15000	0	53.31	535.42	1.11	7.00	0.0005	0.0042

Table 6.14. Results for the weakly correlated instances with deadlines solved using the algorithm with improved upper bound and lower bound.

6.3.4 Non-correlated instances with deadlines

In this subsection, we present the results for comparing the original algorithm to the algorithm that contains improved lower bound and improved upper bound on non-correlated instances with deadlines.

When we compare the results from the tables 6.15 and 6.16, we can see that the results for non-correlated instances are similar to those for weakly correlated instances in the previous subsection. Again, we can see that root-GAP is smaller for our algorithm with improvement lower bound and upper bound than the original algorithm. Smaller root-GAP leads to a significant decrease in both the average time required to solve an instance and the average number of visited nodes.

Both algorithms can solve uncorrelated instances of up to $30,000$ jobs. In comparison to weakly correlated instances, the number of resolved instances is doubled.

When we compare the results for non-correlated, weakly correlated, and strongly correlated instances, there are significant differences in the difficulty of solving these instances, that are depending on the type of instance. For strongly correlated instances, our algorithm can solve all instances of up to $5,000$ jobs. In contrast to non-correlated

		CPU Time		Nodes		root-GAP	
n	unsolved [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
1000	0	0.62	5.06	1.00	1.00	0.0251	0.1303
2000	0	1.32	14.70	1.00	1.00	0.0109	0.0728
3000	0	1.81	8.65	1.00	1.00	0.0060	0.0312
4000	0	3.22	17.68	1.00	1.00	0.0046	0.0189
5000	0	4.66	31.23	1.00	1.00	0.0035	0.0198
6000	0	6.48	35.76	1.00	1.00	0.0032	0.0161
7000	0	9.30	64.25	1.00	1.00	0.0026	0.0158
8000	0	11.21	101.15	1.00	1.00	0.0021	0.0108
9000	0	14.88	159.75	1.00	1.00	0.0018	0.0104
10000	0	18.92	162.12	1.00	1.00	0.0018	0.0105
15000	0	40.04	633.84	1.02	3.00	0.0011	0.0068
20000	0	68.15	557.69	1.09	7.00	0.0009	0.0054
25000	0	116.49	1057.36	1.41	53.00	0.0007	0.0042
30000	0	163.75	822.44	1.25	13.00	0.0005	0.0037

Table 6.15. Results for the non-correlated instances with deadlines solved using the original algorithm.

		CPU Time		Nodes		root-GAP	
n	unsolved [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
1000	0	0.75	4.12	1.00	1.00	0.0102	0.0942
2000	0	1.41	6.04	1.00	1.00	0.0049	0.0604
3000	0	2.36	7.80	1.00	1.00	0.0024	0.0143
4000	0	3.96	18.33	1.00	1.00	0.0021	0.0162
5000	0	5.33	22.21	1.00	1.00	0.0017	0.0147
6000	0	7.67	30.15	1.00	1.00	0.0017	0.0146
7000	0	9.95	39.38	1.00	1.00	0.0014	0.0118
8000	0	11.81	60.25	1.00	1.00	0.0011	0.0103
9000	0	14.23	62.63	1.00	1.00	0.0011	0.0086
10000	0	18.33	105.97	1.00	1.00	0.0010	0.0057
15000	0	36.04	151.86	1.01	3.00	0.0006	0.0058
20000	0	61.88	318.09	1.02	3.00	0.0006	0.0047
25000	0	95.27	566.45	1.17	29.00	0.0004	0.0037
30000	0	132.90	605.11	1.04	3.00	0.0004	0.0037

Table 6.16. Results for the non-correlated instances with deadlines solved using the algorithm with improved upper bound and lower bound.

instances, both our and the original algorithm can solve all instances of up to $3,0000$ jobs. Differences in difficulty between types of instances are mainly due to the reduction of the size of instances by variable-fixing techniques. For strongly correlated instances, fewer jobs can be reduced then for non-correlated and weakly correlated instances.

6.4 Comparison of individual improvements

Next, experiments focus on comparing each algorithm enhancement with the original algorithm on strongly correlated instances with $C = 20$. The identical instances with deadlines, used in the previous section, are utilized for this experiment.

6.4.1 ILP model improvement

When we compare the results from the tables 6.17 and 6.5, we can see that the improved ILP allows solving all instances up to 5,000 jobs. For the remaining instance sizes, It increases the number of resolved instances by more than 27 instances for each instance size. From Table 6.18, it can also be observed that even for the improved ILP approach, instances with a pair of parameters $(0.1, 0.3)$ and $(0.1, 0.5)$ are some of the most difficult instances.

		CPU Time		Nodes		root-GAP	
n	unsolved [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
1000	0	2.16	27.70	1.00	1.00	0.0290	0.1196
2000	0	5.14	38.66	1.00	1.00	0.0144	0.0789
3000	0	9.51	71.71	1.00	1.00	0.0096	0.0357
4000	0	18.22	396.51	1.00	1.00	0.0072	0.0309
5000	0	21.50	135.03	1.06	7.00	0.0056	0.0246
6000	10	225.18	3600.00	19.46	847.00	0.0047	0.0176
7000	34	645.68	3600.00	70.31	2019.00	0.0041	0.0215
8000	40	786.65	3600.00	108.10	1839.00	0.0036	0.0172
9000	68	1252.85	3600.00	151.71	1842.00	0.0033	0.0135
10000	72	1337.61	3600.00	184.90	2553.00	0.0029	0.0115

Table 6.17. Results for the strongly correlated instances with $C = 20$ and with deadlines solved using the algorithm with the improved ILP model.

			Red. prob. size		CPU Time		Nodes		root-GAP	
u	v	unsolved [-]	avg [-]	max [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
0.1	0.3	4	241.45	5392	870.99	3600.00	79.65	847.00	0.0120	0.0176
0.1	0.5	4	119.45	4707	838.81	3600.00	71.55	452.00	0.0068	0.0107
0.1	0.7	0	182.40	3666	38.74	80.56	1.00	1.00	0.0046	0.0083
0.1	0.9	0	59.42	2363	11.97	19.68	1.00	1.00	0.0024	0.0052
0.3	0.5	1	64.05	5213	225.01	3600.00	22.40	407.00	0.0049	0.0097
0.3	0.7	0	142.37	4256	26.82	146.65	1.30	7.00	0.0038	0.0070
0.3	0.9	0	37.63	2757	9.81	23.47	1.00	1.00	0.0025	0.0053
0.5	0.7	1	120.20	4811	198.04	3600.00	14.70	273.00	0.0034	0.0062
0.5	0.9	0	149.00	3054	12.17	19.23	1.00	1.00	0.0029	0.0048
0.7	0.9	0	192.35	4168	19.48	31.94	1.00	1.00	0.0037	0.0052

Table 6.18. Detailed results for the strongly correlated instances with $C = 20$ and with deadlines with 6,000 jobs, solved using the algorithm with the improved ILP model.

6.4.2 Upper bound improvement

In this subsection, we benchmark the improved upper bound. The common reason for not solving instances within the time limit is caused by the ILP solver, which is unable to solve the original ILP model. Therefore, to better demonstrate the actual effect of the improved upper bound, the improved ILP model is also used in this experiment, and the results are compared with the results from the previous subsection, which describes the effect of the improved ILP model.

When we compare results from tables 6.19 and 6.17, it can be observed that the tighter upper bound leads to a significant decrease in the average number of visited

		CPU Time		Nodes		root-GAP	
n	unsolved [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
1000	0	2.35	26.72	1.00	1.00	0.0258	0.1087
2000	0	5.22	46.95	1.00	1.00	0.0133	0.0604
3000	0	9.09	53.50	1.00	1.00	0.0091	0.0357
4000	0	18.10	397.19	1.00	1.00	0.0069	0.0307
5000	0	21.99	137.00	1.05	5.00	0.0054	0.0246
6000	10	224.09	3600.00	16.95	636.00	0.0045	0.0176
7000	33	619.98	3600.00	55.72	1288.00	0.0039	0.0148
8000	38	742.92	3600.00	69.95	792.00	0.0034	0.0172
9000	64	1197.32	3600.00	116.41	1165.00	0.0031	0.0127
10000	69	1287.62	3600.00	136.38	1605.00	0.0028	0.0115

Table 6.19. Results for the strongly correlated instances with $C = 20$ and with deadlines solved using the algorithm with the improved upper bound and improved ILP model.

nodes. This is caused by a smaller root-GAP, which leads to a reduction of more jobs in instances and also to more frequent termination of the branch-and-bounds method because there is no better solution between the lower and upper bound. It is also possible to see that that the average time needed to solve an instance is reduced, even though the computation of the improved upper bound itself requires more time than the original upper bound. The other consequence of an improved upper bound is a few more solved instances than without this improvement.

			Red. prob. size		CPU Time		Nodes		root-GAP	
u	v	unsolved [-]	avg [-]	max [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
0.1	0.3	4	214.90	5392	872.22	3600.00	67.90	636.00	0.0113	0.0176
0.1	0.5	4	119.40	4707	830.46	3600.00	60.80	402.00	0.0066	0.0107
0.1	0.7	0	182.40	3666	39.89	84.51	1.00	1.00	0.0045	0.0083
0.1	0.9	0	59.42	2363	12.50	19.70	1.00	1.00	0.0024	0.0052
0.3	0.5	1	64.05	5213	223.66	3600.00	20.90	381.00	0.0045	0.0087
0.3	0.7	0	142.37	4115	21.69	39.49	1.00	1.00	0.0034	0.0063
0.3	0.9	0	38.28	2757	10.19	24.06	1.00	1.00	0.0024	0.0051
0.5	0.7	1	120.20	4811	197.63	3600.00	13.95	260.00	0.0032	0.0062
0.5	0.9	0	149.00	3054	12.74	20.15	1.00	1.00	0.0029	0.0048
0.7	0.9	0	192.35	4168	19.93	32.53	1.00	1.00	0.0037	0.0052

Table 6.20. Detailed results for the strongly correlated instances with $C = 20$ and with deadlines with $6,000$ jobs, solved using the algorithm with the improved upper bound and improved ILP model.

When we compare the results in tables 6.20 and 6.18 containing more detailed information for instances with $6,000$ jobs, we can see that the improved upper bound results in a better reduction in instance size for some pair of parameters (u, v) than without this improvement, e.g., $(0.1, 0.3)$.

6.4.3 Lower bound improvement

This subsection analyses the algorithm with the improved lower bound and also with the improved ILP model. The structure of the presented tables is the same as in the previous subsection.

		CPU Time		Nodes		root-GAP	
n	unsolved [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
1000	0	1.72	16.98	1.00	1.00	0.0288	0.1196
2000	0	3.86	38.87	1.00	1.00	0.0143	0.0789
3000	0	7.37	68.66	1.00	1.00	0.0095	0.0357
4000	0	13.96	332.87	1.00	1.00	0.0070	0.0299
5000	0	16.79	92.25	1.06	7.00	0.0055	0.0239
6000	10	216.26	3600.00	26.48	1997.00	0.0047	0.0176
7000	33	618.86	3600.00	77.28	3196.00	0.0040	0.0215
8000	40	773.41	3600.00	150.09	2461.00	0.0035	0.0172
9000	67	1246.16	3600.00	204.26	2614.00	0.0032	0.0135
10000	71	1309.17	3600.00	248.40	3557.00	0.0028	0.0103

Table 6.21. Results for the strongly correlated instances with $C = 20$ and with deadlines solved using the algorithm with the improved lower bound and improved ILP model.

When we evaluate data present in tables 6.21 and 6.17 for the improved lower bound, it can be observed that the improved lower bound has a smaller root-GAP compared to the results without this improvement. As a result of smaller root-GAP, the average time required to solve the instance is smaller than without this improvement, and also, the number of solved instances is slightly higher than without this improvement. This only small increase in the number of solved instances is because the original lower bound is relatively tight.

We can also see that there is an increase in the average number of nodes visited. When we compare tables 6.22 and 6.18, it can be observed that this happens only for parameters (u, v) for which not all instances are solved. The increase in the number of nodes visited is because the improved lower bound omits the local search, which in some cases reduces the time required to calculate the lower bound. Secondly, it is also because the improved lower bound has a different set of jobs that are solved using the ILP solver, then original lower bound. Although this set of jobs has a bigger size, in some cases, the ILP solver finds the optimal solution faster.

			Red. prob. size		CPU Time		Nodes		root-GAP	
u	v	unsolved [-]	avg [-]	max [-]	avg [s]	max [s]	avg [-]	max [-]	avg [%]	max [%]
0.1	0.3	4	241.45	5392	836.95	3600.00	140.00	1997.00	0.0120	0.0176
0.1	0.5	4	119.45	4707	796.11	3600.00	78.95	529.00	0.0068	0.0107
0.1	0.7	0	182.40	3666	41.89	308.74	1.00	1.00	0.0043	0.0077
0.1	0.9	0	59.42	2363	8.67	16.09	1.00	1.00	0.0024	0.0052
0.3	0.5	1	64.00	5213	221.95	3600.00	23.90	437.00	0.0049	0.0097
0.3	0.7	0	142.37	4256	23.51	130.99	1.30	7.00	0.0038	0.0070
0.3	0.9	0	37.63	2757	8.09	18.73	1.00	1.00	0.0025	0.0053
0.5	0.7	1	120.20	4811	196.32	3600.00	15.60	291.00	0.0034	0.0062
0.5	0.9	0	149.00	3054	10.90	18.10	1.00	1.00	0.0029	0.0048
0.7	0.9	0	192.35	4168	18.25	30.18	1.00	1.00	0.0037	0.0052

Table 6.22. Detailed results for the strongly correlated instances with $C = 20$ and with deadlines with $6,000$ jobs, solved using the algorithm with the improved lower bound and improved ILP model.

When we compare the results in tables 6.17, 6.19 and 6.21, it can be seen that the most important improvement is the improvement of the ILP model, which allows

instances up to 5,000 to be solved to the optimum. Another very import improvement is the improvement of upper bound heuristic, which allows solving more instances than without this improvement.

6.5 Upper and lower bounds

In this section, we focus on comparing the original lower bound and upper bound with their improved variants. Therefore, in this experiment, we made a comparison between the original algorithm that uses the improved ILP model and our algorithm with all improvements. It should be noted that the improved ILP model does not affect the quality of found bounds but allows us to perform this experiment on instances with a bigger number of jobs and ensure that all instances are resolved to the optimum.

To achieve a fair comparison, it is necessary to ensure that the same instances are solved to the optimum using both algorithms. Therefore, in this experiment assumes instances with 4,000 jobs. The results for strongly correlated instances are summarized in tables 6.23 and 6.24. Column "root-UB-GAP" indicates the relative difference between the upper bound and optimal value, which is computed as $\frac{root-UB-OPT}{OPT} \cdot 100$. In the same way, column "root-LB-GAP" indicates the relative difference between the lower bound and the optimal value, which is computed as $\frac{OPT-root-LB}{OPT}$. Values in both columns are in percentages.

		root-LB-GAP		root-UB-GAP	
u	v	avg [%]	max [%]	avg [%]	max [%]
0.1	0.3	0.0001478	0.0009857	0.0167	0.0299
0.1	0.5	0.0011834	0.0127308	0.0102	0.0208
0.1	0.7	0.0004803	0.0096056	0.0072	0.0117
0.1	0.9	0.0003891	0.0077826	0.0045	0.0082
0.3	0.5	0.0000317	0.0006339	0.0075	0.0147
0.3	0.7	0	0	0.0051	0.0103
0.3	0.9	0	0	0.0044	0.0069
0.5	0.7	0	0	0.0059	0.0100
0.5	0.9	0	0	0.0040	0.0081
0.7	0.9	0	0	0.0042	0.0075

Table 6.23. Detailed results for the strongly correlated instances with $C = 20$, and deadlines having 4,000 jobs, solved using the algorithm with improved ILP model.

When we compare the results from the tables 6.23 and 6.24, we can see that for all pairs of parameters (u, v), our improved algorithm always finds a better upper bound than the original algorithm. On average, the most significant improvement is achieved for the pair of parameters $(0.1, 0.3)$.

In the case of lower bound, the situation is a bit more complicated. The original lower bound is very tight and often finds an optimal solution. In the tables, this case is denoted by value 0, as can be observed, i.e., in the last five rows in both tables. From the results, we can see that the improved lower bound, in most cases, finds the same or better solution than the original lower bound. In the case of parameters $(0.1, 0.5)$, it even found an optimal solution for all instances compared to the original lower bound. On the other hand, for the parameters $(0.3, 0.5)$, we can see that the original lower bound found a better solution. On average, the improved lower bound is tighter than the original one.

		root-LB-GAP		root-UB-GAP	
u	v	avg [%]	max [%]	avg [%]	max [%]
0.1	0.3	0.0000491	0.0009822	0.0157	0.0297
0.1	0.5	0	0	0.0100	0.0207
0.1	0.7	0.0000241	0.0004834	0.0071	0.0117
0.1	0.9	0.0000388	0.0003890	0.0043	0.0081
0.3	0.5	0.0000320	0.0006405	0.0067	0.0141
0.3	0.7	0	0	0.0048	0.0103
0.3	0.9	0	0	0.0044	0.0069
0.5	0.7	0	0	0.0057	0.0098
0.5	0.9	0	0	0.0039	0.0076
0.7	0.9	0	0	0.0041	0.0075

Table 6.24. Detailed results for the strongly correlated instances with $C = 20$, and deadlines having $4,000$ jobs, solved using the algorithm with all improvements.

6.6 The heaviest strongly correlated instances

The authors of the paper [1] point out the special type of strongly correlated instances ($p_i = w_i + 20$) that have only two possible due dates for all the jobs. According to their results, these instances prove to be the heaviest correlated instances. They also give an example of such instance with 200 jobs, that none of the ILP solvers was able to solve within $3,600$ seconds. However, current ILP solvers are more efficient, therefore ILP solver Gurobi 8.1.1 can solve their attached instance in less than 7 seconds with the original ILP model.

We performed experiments for this kind of instance. We generated strongly correlated instances of 200 jobs without deadlines, where the first half of the jobs have one common due date, and the second half of the jobs have a common second due date. For each pair of parameter (u, v) it was generated 20 instances which is together 200 instances.

Instances with 200 jobs were all solved by the original algorithm within $3,600$ seconds except two instances. Based on this observation, we perform the next experiment comprising also larger instances than 200 jobs and having two due dates. The largest instances have $5,000$ jobs since instances having less then or $4,0000$ are solved directly by an ILP formulation. In this experiment, we compared the original ILP model with our improved ILP models on this strongly correlated instances.

The results are summarized in tables 6.25 and 6.26. From the results, it can be seen that the original model cannot solve all instances within the time limit of $3,600$ seconds, even for instance with 150 jobs. On the other hand, our improved ILP model has no problem solving all $5,000$ instances within the same time limit. Moreover, it can also be observed that our improved ILP model has relatively small differences between avg and max CPU time, indicating a relatively stable run time for these strongly correlated instances compare to the original ILP model.

n	unsolved [-]	CPU Time avg [s]	CPU Time max [s]
50	0	0.01	0.13
100	0	0.14	24.18
150	1	26.71	3600.00
200	2	36.33	3600.00
250	3	54.05	3600.00
500	2	36.04	3600.00
1000	13	234.09	3600.00
2000	15	283.02	3600.00
3000	12	216.05	3600.00
4000	11	198.25	3600.00
5000	11	198.10	3600.00

Table 6.25. Results for the strongly correlated instances with $C = 20$ and with two due dates, solved using the original ILP model.

n	unsolved [-]	CPU Time avg [s]	CPU Time max [s]
50	0	0.06	0.11
100	0	0.06	0.13
150	0	0.07	0.17
200	0	0.05	0.11
250	0	0.06	0.10
500	0	0.06	0.10
1000	0	0.08	0.25
2000	0	0.09	0.21
3000	0	0.11	0.57
4000	0	0.14	0.84
5000	0	0.15	0.30

Table 6.26. Results for the strongly correlated instances with $C = 20$ and with two due dates solved using the improved ILP model.

Chapter 7
Conclusion

The goal of this thesis was to extent results published in paper [1], where the authors present an algorithm for problem $1|\tilde{d}_j|\sum w_jU_j$. We focus on strongly correlated instances that are very difficult for their algorithm. Our goal was to achieve better results for this type of instances.

Based on the experiments in Section 6.2, we found an anomaly for strongly correlated instances with $C = 0$. This type of strongly correlated instances can be solved even faster than non-correlated instances using an ILP solver. Based on this anomaly, we created a modified ILP model with the decomposition according to the number of early jobs. This improved ILP model allows solving strongly correlated instances of up to $5,000$ jobs, as is shown in the experiments in Section 6.2.

We also introduced improvements for both lower bound and upper bound, which are tighter than in paper [1]. Their improvements make it possible to solve a bigger number of instances, which is caused by reducing more jobs using variable-fixing techniques. The experiments in sections 6.3.3 and 6.3.4 confirmed that these improved bounds are also useful for non-correlated and weakly correlated instances. For these types of instances, improved bounds reduce the number of visited nodes in the branch-and-bound method.

In Section 6.3, we described results of many experiments comparing the original algorithm from paper [1] with the algorithm that contained all our improvements. The original algorithm had a problem to solve all instances with 150 jobs to the optimum before the time limit. In contrast, our improvements made it possible to solve all instances of up to $5,000$ jobs. For instances with $6,000$ jobs, our improved algorithm solved 190 instances to optimum out of a total of 200.

It should be mentioned that we initially dealt with a possible use of machine learning for strongly correlated instances of problem $1|\tilde{d}_j|\sum w_jU_j$. We based our research on paper [25], which uses decision diagrams and deep reinforcement learning to achieve relatively tight bounds for both maximum independent set problem and maximum cut problem. Our experiments led to the fact that the use of decision diagrams and machine learning would not perform well for problem $1|\tilde{d}_j|\sum w_jU_j$. We think that this was because problem $1|\tilde{d}_j|\sum w_jU_j$ has very few infeasible solutions besides a huge number of feasible solutions, but also because of its objective function.

7.1 Future work

One option is to explore methods and results for Knapsack problem presented by Pisinger in paper [26]. In paper [26], Pisinger presents an algorithm that allows solving strongly correlated instances of Knapsack problem up to $100,000$ items. Before Pisinger introduced its algorithm, strongly correlated instances were considered more difficult than uncorrelated instances for general Knapsack problem solvers.

References

[1] Philippe Baptiste, Federico Della Croce, Andrea Grosso, and Vincent T'kindt. Sequencing a single machine with due dates and deadlines: an ILP-based approach to solve very large instances. *Journal of Scheduling*. 2010, 13 (1), 39–47.

[2] Silvano Martello, and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990. ISBN 0-471-92420-2.

[3] Chris N Potts, and LN Van Wassenhove. Algorithms for scheduling a single machine to minimize the weighted number of late jobs. *Management Science*. 1988, 34 (7), 843–858.

[4] J. T. Linderoth, and M. W. P. Savelsbergh. A Computational Study of Search Strategies for Mixed Integer Programming. *INFORMS Journal on Computing*. 1999, 11 (2), 173-187.

[5] Richard M Karp. *Reducibility among combinatorial problems*. 1972.

[6] Danny Hermelin, Shlomo Karhi, Michael Pinedo, and Dvir Shabtay. New algorithms for minimizing the weighted number of tardy jobs on a single machine. *Annals of Operations Research*. 2018, 1–17.

[7] Eugene L Lawler. Scheduling a single machine to minimize the number of late jobs. 1983,

[8] Eugene L Lawler, and J Michael Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*. 1969, 16 (1), 77–84.

[9] Sartaj K Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM (JACM)*. 1976, 23 (1), 116–127.

[10] Francisco J Villarreal, and Robert L Bulfin. Scheduling a single machine to minimize the weighted number of tardy jobs. *IIE Transactions*. 1983, 15 (4), 337–343.

[11] Guochun Tang. A new branch and bound algorithm for minimizing the weighted number of tardy jobs. *Annals of Operations Research*. 1990, 24 (1), 225–232.

[12] Rym M'Hallah, and RL Bulfin. Minimizing the weighted number of tardy jobs on a single machine. *European Journal of Operational Research*. 2003, 145 (1), 45–56.

[13] AMA Hariri, and CN Potts. Single machine scheduling with deadlines to minimize the weighted number of tardy jobs. *Management Science*. 1994, 40 (12), 1712–1719.

[14] Ming Liu, Shijin Wang, Chengbin Chu, and Feng Chu. An improved exact algorithm for single-machine scheduling to minimise the number of tardy jobs with periodic maintenance. *International Journal of Production Research*. 2016, 54 (12), 3591-3602.

[15] Zhenyou Wang, Cai-Min Wei, and Linhui Sun. Solution algorithms for the number of tardy jobs minimisation scheduling with a time-dependent learning effect. *International Journal of Production Research*. 2017, 55 (11), 3141-3148.

[16] "Claudio Arbib, Giovanni Felici, and Mara Servilio". "Common operation scheduling with general processing times: A branch-and-cut algorithm to minimize the weighted number of tardy jobs". *"Omega"*. "2019", "84" "18 - 30".

[17] "Qiulan Zhao, and Jinjiang Yuan". "A note on single-machine scheduling to trade-off between the number of tardy jobs and the start time of machine". *"Operations Research Letters"*. "2019", "47" ("6"), "607 - 610".

[18] "Y. Laalaoui, and R. M'Hallah". "A binary multiple knapsack model for single machine scheduling with machine unavailability". *Computers & Operations Research*. "2016", "72" "71 - 82".

[19] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. *Optimization and approximation in deterministic sequencing and scheduling: a survey*. 1979.

[20] J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Handbook on Scheduling: From Theory to Applications*. Springer Berlin Heidelberg, 2007. ISBN 9783540322207.

[21] J.Y.T. Leung. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, 2004. ISBN 9780203489802.

[22] P. Brucker. *Scheduling Algorithms*. Springer Berlin Heidelberg, 2007. ISBN 978-3540695165.

[23] Dimitri P Bertsekas, and Paul Tseng. *RELAX-IV: A faster version of the RELAX code for solving minimum cost flow problems*. 1994.

[24] Antonio Frangioni, and Claudio Gentile. *The MCFClass project*. 2001. `http://www.di.unipi.it/optimize/Software/MCF.html`.

[25] Quentin Cappart, Emmanuel Goutierre, David Bergman, and Louis-Martin Rousseau. *Improving optimization bounds using machine learning: decision diagrams meet deep reinforcement learning.* In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2019. 1443–1451.

[26] David Pisinger. A fast algorithm for strongly correlated knapsack problems. *Discrete Applied Mathematics*. 1998, 89 (1-3), 197–212.

Appendix A
Content of the attached CD

- `DP_2020_Hejl_Lukas.pdf` — This thesis in PDF file,
- `source_code.zip` — Source files of implemented algorithms,
- `source_text.zip` — Source files of the thesis.