Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Management and Production Engineering (36^{th}cycle)

Single Machine Rescheduling for New Orders

Properties, complexity and solution algorithms

By

Elena Rener

******

Supervisor(s):
Prof. Fabio Salassa, Supervisor
Prof. Vincent T'kindt, Co-Supervisor

Doctoral Examination Committee:
Prof. Alessandro Agnetis, Referee, Università di Siena
Prof. Roel Leus, Referee, University KU Leuven
Prof. Federico Della Croce, Politecnico di Torino
Prof. Joanna Józefowska, Poznań University of Technology
Prof. Giuseppe Lancia, Università di Udine

Politecnico di Torino
2024

Abstract

Rescheduling adds to deterministic scheduling the ability to respond to the occurrence of an unexpected event. While scheduling proposes methods to optimize the execution of tasks on machines, rescheduling proposes methods to best *update* known schedules when they become suboptimal for some reason. The unexpected triggering event studied in this dissertation is the arrival of new jobs. The new goal is to optimize the execution of all jobs, but also to avoid excessive deviations from the original schedule on which other work may rely. Methods for rescheduling involve the consideration of an objective function to be minimized and a disruption constraint that limits the deviation from the original solution. The goal of this dissertation is to study, from both a theoretical and algorithmic perspective, single-machine rescheduling problems to minimize classical scheduling objective functions given a disruption constraint measured as a function of the absolute deviation of completion times. Since rescheduling for new orders has only been studied in a fragmented and case-by-case manner in the literature, the work in this dissertation attempts to provide a comprehensive analysis of the nature of rescheduling problems and ways to solve them. First, the analysis considers the formulation of structural properties of the problem and focus on demonstrating the computational complexity of the problems. The structural analysis is followed by a detailed discussion of the timing problem arising from those problems where the problem formulation leads to optimal solutions that may include unforced idle time on the machine. Algorithms for the problems are proposed and their polynomiality is shown. The remainder of the dissertation studies how to approach two computationally hard rescheduling problems. The first problem is solved to optimality in pseudo-polynomial time using dynamic programming, and then using approximation algorithms to obtain solutions with a bounded performance ratio. For the second problem, a branch and memorize algorithm is developed that exploits a generic structure and specific properties. The algorithm is the first exact search tree algorithm for a hard rescheduling problem

that exploits an efficient exploration of the solution space. Computational experiments comparing the algorithm with a commercial solver for optimization problems show that the algorithm does indeed succeed in exploiting problem-specific features for solving the problem in a more efficient way. This concludes the discussion of rescheduling problems analyzed and studied as combinatorial problems, with the aim of providing a starting point for solving them from an application perspective.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Machine scheduling

The practical importance of scheduling in the field of production planning and in resource management has made scheduling theory an important area of Operations Research. The literature on scheduling theory is vast and has been abundant since the 1950s. Some introductory reference works are those by Baker (1974), Lawler et al. (1993), Tanaev et al. (1994b), Tanaev et al. (1994a). Pinedo (2012) gives an updated version of most used definitions, problem descriptions and directions considered in scheduling. A review on existing algorithms used in scheduling is given by Brucker (2007).

Scheduling refers to the allocation of resources over time for the purpose of executing a collection of tasks. A job consists of an ordered list of operations, each of which requires a specific processing time. In general, each machine can only process one job at a time and is available continuously from instant 0 onwards. Each job can only run on one machine at a time. A complete scheduling plan indicates for each operation of each job when and on which machine this operation is executed. The objective is to determine a scheduling plan that optimizes some criterion normally denoted by a function of the completion times of each job. In deterministic scheduling problems, it is assumed that all specifications are unambiguously predetermined.

We introduce some standard notation and definitions used in scheduling in the next section. In Section 1.2, we give an introduction to combinatorial optimization, which studies solutions to complex problems, such as scheduling problems. In Sec-

tion 1.3, we introduce the machine rescheduling problem, that adds to deterministic scheduling the ability to respond to the occurrence of an unexpected event, which is the topic of this dissertation.

Notation and definitions

We introduce in this section notations and definitions commonly used in scheduling. There is a set J of jobs and a job represents a task to be processed on some machines to minimize a certain objective function. Each job consists of a number of operations that have to be performed on a specific machine or on a set of parallel machines. The processing time of the $j-th$ job on the $i-th$ machine is p_{ji} and is the time requirement the job operation needs in order to be completed. Preemption is allowed if a job operation can be interrupted at a certain point and completed for the the remaining time afterwards. If each job is made of a single operation and there is one available machine, we have single machine scheduling problems. If the jobs have one operation per machine and the sequence of operations is the same for all jobs, we have a flow shop. If the sequence of operations is not fixed, we call it an open shop. Lastly, if the sequence of operations is fixed but different among jobs, then we have a job shop.

Beside the processing time, each job j is usually characterized by a due date d_j and/or a weight w_j. A job may additionally have a release date r_j if it is available only from r_j onwards and a deadline $\tilde{d}_j$ if the job has the strict requirement of being completed by $\tilde{d}_j$. A schedule is an allocation of a time interval to each job. When all time intervals allocated on a machine are consecutive, we talk of a compact schedule with no machine idle time or a non-delay schedule; whereas when there is some empty interval left, we call this interval an idle time. A schedule can be represented as a Gantt chart that shows how operations are carried on with respect to time on each machine. Given a schedule σ, $C_j(\sigma)$ (C_j when there is no ambiguity) is the completion time of job j, i.e. the time by which all operations of the job are completed.

Each job in a schedule is associated with a function f_j that depends on job completion time. In standard scheduling, there are two types of total cost functions: bottleneck objective functions and sum objective functions. Bottleneck objective

functions are in the form $f_{max} = \max_{j \in J} f_j$, while sum objective functions are in the form $\sum f = \sum_{j \in J} f_j$.

Common bottleneck objective functions are:

- makespan $C_{max} = \max_{j \in J}(C_j)$.
- maximum lateness $L_{max} = \max_{j \in J}(C_j - d_j)$.

Common sum objective functions are:

- total completion time $\sum C_j = \sum_{j \in J} C_j$,
- total number of late jobs $\sum U_j = \sum_{j \in J} U_j$,
- total tardiness $\sum T_j = \sum_{j \in J} \max(0, C_j - d_j)$,
- total earliness $\sum E_j = \sum_{j \in J} \max(0, d_j - C_j)$,

The above objective functions are often considered also in their weighted versions, where each job-related function f_j is additionally multiplied by the job weight w_j.

An objective function that is non decreasing with respect to job completion times is said to be regular (e.g. all objectives above except the total earliness), otherwise it is said to be non regular (e.g. the total earliness).

Scheduling problems are specified using the three-field classification $\alpha|\beta|\gamma$ introduced by Graham et al. (1979). The α field denotes the machine environment (e.g., 1 for single machine, P2 for two parallel machines, F3 for a three-machines flow shop, ...). The β field gives additional information on job characteristics and/or additional constraints, such as the existence of time windows and/or precedence constraints. In particular, if jobs have time windows, we write r_j and/or $\tilde{d}_j$ to denote release dates and/or deadlines. If there are general precedences between jobs, we write *prec*, if the precedence between jobs is defined as a fixed sequence of jobs, we denote it by *seq*. If schedules must be non-delay, i.e. machine idle times are prohibited, we write *no-idle*. If preemption is allowed, then this is denoted with *pmtn*. In the γ field, there is the objective function to be minimized.

Let us now define the following ordering rules for the jobs in a schedule:

- *Earliest Due Date* (*EDD*). With this rule, jobs are assumed to be ordered in non-decreasing order of their due dates, i.e. $d_1 \leq d_2 \leq \cdots \leq d_n$.

- *Shortest Processing Time* (*SPT*). With this rule, jobs are assumed to be ordered in non-decreasing order of their processing times, i.e. $p_1 \leq p_2 \leq \cdots \leq p_n$.
- *Weighted Shortest Processing Time* (*WSPT*). With this rule, jobs are assumed to be ordered in non-increasing order of the ratio $\frac{w_j}{p_j}$, i.e. $\frac{w_1}{p_1} \geq \frac{w_2}{p_2} \geq \cdots \geq \frac{w_n}{p_n}$.

In the dissertation, we will make an extensive use of the following well-known results for single machine scheduling:

1. Scheduling jobs according to EDD optimally solves the single machine scheduling problem to minimize maximum lateness $1||L_{max}$.
2. Scheduling jobs according to SPT optimally solves the single machine scheduling problem to minimize total completion time $1||\sum C_j$.
3. Scheduling jobs according to WSPT optimally solves the single machine scheduling problem to minimize total completion time $1||\sum w_j C_j$.

1.2 Combinatorial optimization and computational complexity

Combinatorial optimization is the broad discipline that studies how to find solutions to complex problems that can be expressed in mathematical models and where there are a finite number of solutions. Formally, combinatorial optimization deals with those problems where it is possible to uniquely define the set S of finite cardinality of all solutions and, for each solution $s \in S$, an objective function with values in R. In these terms, the objective is to find the element $s \in S$ for which the value $f(s)$ is maximum or minimum.

Examples of classical combinatorial optimization problems are the knapsack problem, the maximum flow problem, and the transportation problem. In the knapsack problem, there are a number of items with a weight and a profit, and the objective is to fill the knapsack so as to maximize the total profit value of the items taken, subject to the constraint that a given capacity is not exceeded. In the maximum flow problem, there is a directed network with nodes connected by arcs, each with a given capacity, and the objective is to determine the maximum amount of flow that

can be sent from a given source node to another given sink node. In the transportation problem, there is a network of supply and demand nodes, each with a given amount of goods being supplied and demanded, respectively, and the objective is to find the minimum transportation cost to match the supply and demand requests, given that each pair of nodes corresponds to a cost per unit flow.

Solving an optimization problem can be computationally very expensive. Therefore, when providing algorithms for solving a problem, an important goal of combinatorial optimization is to study the efficiency of such algorithms, which is often referred to as the *time complexity*, or computational complexity. The time complexity of an algorithm gives an indication of the time it will take to solve a problem, given its size. Instead, the complexity of a problem is given by the time complexity of the best algorithm that solves the problem. The effort spent into improving the efficiency of algorithms to solve a specific problem is thus motivated by the information that we get in return about that problem. The more we reduce the time complexity of an algorithm, the more we tighten the complexity of the problem. Computational theory, as we know it today, is mainly due to the contributions of Cook (1971), Karp (1972), Garey and Johnson (1979), Ausiello et al. (1999).

The solution of a computational problem can be viewed as a function f that maps, to each input x, an output $f(x)$. Let be $|x|$ the size of a vector containing all input values given in some encoding. A problem is *polynomial* if there exists an algorithm that finds an optimal solution to the problem with time complexity in $O(|x|^k)$, for some constant k. The class of polynomial problems is denoted by P.

Let a decision problem be a problem that admits as a solution only a "yes" or a "no" answer, for instance a problem that asks whether $f(x) \leq K$. All decision problems for which, given an input $\bar{x}$, the answer to the decision problem $f(\bar{x}) \leq K$ is found in polynomial time define the class of problems NP. Clearly, $P \subseteq NP$. Let an optimization problem be a problem that requires finding the solution that maximizes/minimizes a function $f(x)$. If we consider any value K of its possible output values, an optimization problem can be seen as a set of decision problems that answer to whether or not exists a solution with $f(x) \leq K$.

A decision problem π is NP-complete if $\pi \in NP$ and if any other problem in NP can be reduced to it polynomially. A decision problem π can be reduced to a decision problem π' if there exists a polynomial algorithm that transforms the input values of π into the input values of π', and if a "yes" answer to problem π corresponds to a

"yes" answer to problem π'. The class *NP*-complete contains the hardest problems of *NP*. Given the reducibility of any problem in *NP* to any other *NP*-complete problem, if a polynomial-time algorithm existed for solving one of the *NP*-complete problems, then all problems in *NP* would be solvable in polynomial time. The *P* vs. *NP* problem is a major unsolved problem in theoretical computer science, but it is a common assumption that $P \neq NP$ and no polynomial algorithm exists for problems in *NP*. Many decision problems studied in combinatorial optimization are *NP*-complete, in practice they are fast to check but slow to solve.

We can distinguish two types of problems: problems that are *NP*-complete in the weak, or ordinary, sense, and problems that are *NP*-complete in the strong sense. A problem is said to be *NP*-complete in the weak sense if it is *NP*-complete with respect to the binary encoding of the input values, but there exists an algorithm that solves the problem in polynomial time with respect to the unary encoding of the input values. A problem is said to be *NP*-complete in the strong sense if the problem is also *NP*-complete with respect to the unary encoding of the input values. The difference lies in the fact that there is an exponential difference between the length of a unary and a binary encoding.

This can be translated into the following definition (T'kindt and Billaut (2006), Def. 11). Given the set of instances, with maximum input value x_{max} and input length $|x|$, of an *NP*-complete subproblem π' of π, the problem π is *NP*-complete in the strong sense if there exists a polynomial λ such that

$$x_{max} \leq \lambda(|x|)$$

for any of the instances of π'. If there is no polynomial λ, π' is a number problem and we cannot decide whether π is strongly *NP*-complete.

A decision problem can be associated with an optimization problem by searching for a solution which has a better value than a given bound K. If this decision problem is *NP*-complete, then at least so is the optimization problem. A problem is *NP*-hard if any problem in *NP* reduces to it, but is not necessarily in *NP* i.e., if its belonging to the *NP* class cannot be verified. In fact, an optimization problem may have a large number of solutions, and checking whether a given solution is optimal is often non-trivial. A problem is at least weakly (resp. strongly) *NP*-hard if its decision version is *NP*-complete in the weak (resp. strong) sense.

1.3 Machine rescheduling

Generating high-quality production schedules is important in most manufacturing facilities for the benefits that can be obtained in terms of productivity and cost efficiency. Schedules specify the allocation of tasks to the available resources and thus define the production plan, depending on production requests, system's capabilities or material availability. Schedules are built taking in account the amount of work that needs to be carried out to satisfy the requests, and these may be known in advance or estimated based on system-specific information. However in practice, schedules must be often updated because of the inaccuracy of predictions or unexpected changes in the available data. Typical disturbances frequently encountered in manufacturing facilities include machine breakdowns, rush orders, order cancellations, changes in priorities and due dates, labour unavailability, material arrival delays, raw material shortages, transportation delays, rework, process time variation, setup time variation, outsourcing, machine performance degradation, etc. If data at hand change, schedules may become infeasible or inefficient. The scheduler must be able to react quickly and in the most efficient manner in order not to lose too much in the cost-efficiency of the schedule. *Rescheduling* refers to the dynamic approaches used to update predetermined schedules to respond to such disruptions.

Rescheduling differs from *predictive* scheduling, as well as from *online* scheduling methods (refer, for instance, to Aytug et al. (2005)). On the one hand, predictive scheduling (e.g., robust or stochastic scheduling) considers a range of scenarios representing the outcomes of different problem realizations to build schedules with the goal of performing relatively well under a wide range of possible problem realizations. A drawback of these approaches is that they do not explicitly consider issues that may raise during execution. On the other hand, online scheduling considers providing schedules according to the currently available information. This approach has many practical advantages: it has low computational cost, it is intuitive, and easy to explain. However, these methods tend to provide localized or myopic scheduling. Instead, rescheduling, also known as *reactive* scheduling, integrates changes in the data based on the initial state to ensure a certain level of stability. The initial state, which is assumed to be known periodically, takes into account global information. Then, the reaction takes into account the updated information and combines the two. Notice that this approach also differs from solving a production optimization problem in a rolling horizon fashion, which aims to repeatedly solve a "local" problem of a

larger time horizon, where the global information, which is aggregated to the current information, does not represent in itself a fixed schedule.

Methods and policies for rescheduling allow flexibility and a proper capability in the system's reactiveness. In the literature, rescheduling methods have been developed to respond to unforeseen disruptive events, such as the arrival of new jobs, the delay of some, machine breakdowns, or periods of unavailability of machines due to maintenance activities. This thesis studies, in particular, rescheduling strategies when new orders, called *new jobs*, have to be scheduled in an already given optimal solution made up of known jobs, called *old jobs*. This is typically called *rescheduling for new orders*. These strategies seek a trade-off between limiting the perturbation of the original schedule, called *disruption* of the schedule, and integrating the new jobs to optimize an objective function.

1.3.1 Rescheduling classes

Variability in industrial systems may caused by several intrinsic and extrinsic factors. For instance, consider machine breakdowns that disrupt planned activities and preventive maintenance can reduce the failure rate, but it is almost impossible to completely eliminate this type of disturbance from the system. Similarly, other parameters such as material availability and market demand are likely to change, so it is essential to react quickly and produce new schedules that take into account the new system parameters.

In the literature, several differences between theoretical models and real-life scheduling problems when dealing with data uncertainty (Pinedo (2012)) are pointed out.

1. New jobs are constantly added to the system and are not predetermined at any point in time.

2. Assumptions, such as material or work force availability, on which a schedule is based on may change over time.

3. The priorities of jobs may fluctuate over time.

4. It is assumed that machines are available at all times, however, in practice machines are usually not continuously available due to breakdowns or repair operations.

5. Processing times may be subject to change due to learning or deterioration effects.

Therefore, when designing a scheduling system, uncertainties need to be taken into account, since, as time goes by, production schedules become suboptimal or even infeasible and eventually a new updated schedule will be needed. Depending on the source of disturbance, the problem settings change, in terms of data variations and time constraints, introducing different rescheduling classes. We give a brief description of main classes of rescheduling considered by the current literature in the following paragraphs.

Rescheduling for new orders. It is very common in production planning, that new unforeseen requests are received during time. They may be for instance rush orders from customers, or job rework requests arising from the system itself. At any scheduling stage, the scheduler considers the current information on the set of jobs available for processing and makes a plan to optimize system performances. Since scheduling performances rarely depend on the time a new request is received, the scheduler may decide to revise the schedule of jobs that have not been processed yet to integrate the new incoming orders. At any rescheduling point, a new schedule is created that includes all jobs known at that time.

Rescheduling for machine unavailability. In many industrial settings, core operations are processed by machines and they guarantee efficiency and a proper execution of the work plan. Well-defined plans for preventive maintenance may reduce the number of unpredicted interruptions, however it is impossible to completely eliminate temporary breakdowns as well as unpredicted maintenance periods imposed by higher levels. As a result, the scheduler has to revise the initial schedule in order to take in account the period in which the machine will be unavailable for continuing processing the jobs. Hence, the new scheduling decision takes in account the same set of jobs but different available time intervals to allocate them.

Rescheduling for job unavailability. When the scheduler creates a plan for processing jobs, he or she bases the decision on the availability of parts or components used for processing jobs. The unavailability of some of them makes impossible to process some jobs, that must be delayed, at their scheduled time, hence causing a disruption in the schedule. As a result, additional constraints are added in the form of updated release dates or new precedence relations between jobs. The new revised schedule has to satisfy the newly added constraints.

1.3.2 Rescheduling for new orders

Rescheduling for new orders has been one of the most considered problems among rescheduling problems. In a production setting, schedules for the known jobs must be prepared in advance and must be available for the daily advance of manufacturing operations. On the contrary, orders from customers can be received at any time. A rescheduling strategy allows the advantage of an improved flexibility in operations planning and customer orders acceptance. Its vast practical applicability explains why many researchers have devoted their work in developing models and solution methods for this problem.

The model that has prevailed over the others is one defined in the first place in Hall and Potts (2004). Rescheduling is performed over an initial *optimal* schedule of old jobs to allow the integration of a new set of jobs when these arrive. The initial schedule is assumed to be known on a regular basis based on global information about customer orders. The objective is to minimize a scheduling function over both sets of jobs, while a constraint, the so-called *disruption constraint* limits the disruption of the initial schedule. The disruption constraint is modelled in the form of a function of the absolute time deviations of old jobs. In these terms, each time shift of one of the original jobs causes a disruption cost, which represents the cost of reorganizing the associated resources over time. Notice that both an anticipation or a delay of a job enforce changes in the production line. We assume that all jobs incur the same disruption cost per unit shift.

This dissertation discusses several aspects that should be taken in account for solving rescheduling problems for new orders. The formal problem description as well as the notation used will be given in the next section. Section 1.4 summarizes the related literature.

1.3.3 Notation, definitions and problem description

We introduce in this section the formal definition of the rescheduling problem for new orders and give the notation that will be used throughout the thesis together with classical scheduling notation (see Section 1.1).

We are given a set of jobs, called *old* jobs, that is scheduled to be processed on a single machine, when a new set of jobs, called *new* jobs, arrives and create a disruption. Let O be the set of n_O old jobs. At the first stage, each job $j \in O$ is assumed to be optimally scheduled in a sequence π^*, with completion time $C_j(\pi^*)$, to minimize a given objective function f. Let N be the set of n_N new jobs, that appear in the second stage. After the arrival of jobs in N, the goal is to generate a new schedule σ^* of all jobs to minimize f while not disrupting too much the initial schedule π^*. The disruption Δ_j of any job $j \in O$ is measured as the absolute time deviation from the initial completion time, i.e. $\Delta_j = |C_j(\sigma^*) - C_j(\pi^*)|$. The disruption of the schedule is modelled as the maximum or the sum of the absolute time deviations of old jobs. We call $\Delta_{max} = max_{j\in O}(\Delta_j)$ the maximum disruption and $\sum\Delta = \sum_{j\in O}\Delta_j$ the total disruption. To limit the amount of schedule disruption, we introduce a threshold Y and define two constraints $\Delta_{max} \leq Y$ and $\sum\Delta \leq Y$.

In this thesis, we consider minimizing a regular objective function over both set of jobs. Among bottleneck objective functions f_{max}, we consider:

- maximum lateness $L_{max} = \max_{j\in O\cup N}(C_j - d_j)$.

Among sum objective functions $\sum f$, we consider:

- total weighted completion time $\sum w_jC_j = \sum_{j\in O\cup N} w_jC_j$,
- total number of late jobs $\sum U_j = \sum_{j\in O\cup N} U_j$, where $U_j = 1$ if $C_j > d_j$ and $U_j = 0$ otherwise,
- total tardiness $\sum T_j = \sum_{j\in O\cup N}(0, C_j - d_j)$.

Using the $\alpha|\beta|\gamma$ classification scheme, we denote the problems that we study as $1|\Delta_{max} \leq Y|f$ and $1|\sum\Delta \leq Y|f$ for problems that allow the presence of unforced idleness on the machine, and $1|no-idle, \Delta_{max} \leq Y|f$ and $1|no-idle, \sum\Delta \leq Y|f$ for problems with non-delay schedules.

Table 1.1 General Notation.

Notation	Description
$O = \{1, \ldots, n_O\}$	set of old jobs
$N = \{n_O + 1, \ldots, n_O + n_N\}$	set of new jobs
$n = n_O + n_N$	total number of jobs
p_j	processing time of job j
w_j	weight of job j
d_j	due date of job j
π^*	optimal original schedule of old jobs
$C_j(\sigma)$	completion time of job j in schedule σ, denoted as C_j when there is no ambiguity
$f(\sigma)$	objective function evaluated for schedule σ
Δ_j	disruption of old job j, as $\lvert C_j - C_j(\pi^*)\rvert$
Y	threshold on the maximum allowable disruption
P_S	total processing time of jobs in set S
W_S	total weight of jobs in set S

In Table 1.1 it is given a summary of the notation used throughout the dissertation.

1.4 Rescheduling for new orders: a literature review

Rescheduling is understood as a set of strategies to update schedules to face some kind of variation in the workload that has to be scheduled. So, it refers to a set of methods and models designed to be implemented in different contexts. The two surveys by Vieira et al. (2003) and Aytug et al. (2005) give a general introduction to rescheduling problems. The work by Vieira et al. (2003) describes a framework for understanding rescheduling strategies, policies and methods, and presents definitions that are appropriate for most applications of rescheduling manufacturing systems. A variety of experimental and practical approaches described in the rescheduling literature form the basis of the provided framework. The paper discusses studies that show how rescheduling affects the performance of a manufacturing system, and concludes with a discussion on how understanding rescheduling can bring some aspects of scheduling theory and practice closer together. The second survey by Aytug et al. (2005) provides a framework for understanding the different issues involved in developing effective scheduling and rescheduling methods for environments, where

there is some execution uncertainty. A first relevant parameter is the source of uncertainty. For this purpose, the paper introduces and discusses a taxonomy for viewing and classifying production uncertainties. A number of different problem formulations in the scheduling literature are presented and discussed to consider the different factors. A focus is put on the case, where schedules are executed in automated settings when uncertainty is present.

The two surveys show what are the general ideas and directions for models that are commonly adopted in the manufacturing industry. Rescheduling is based on the idea that disruptions create opportunities to improve shop performance based on what happens after a disruption occurs. These opportunities can be exploited at a cost, where the two main types of costs are those related to the performance of the system in terms of conventional scheduling criteria, and those related to the cost of reconfiguration, i.e. the cost of having schedule *instability*, or schedule *disruption*. We will see in the remaining part of this section that disruption costs have been modeled in several ways. The two most common models consider position and time deviations as a cost. The first case is relevant, for example, for components processed in sequence at later stages. Any change in the position of an order in the sequence causes additional costs (e.g., increased buffer capacity). The time deviation model is relevant for systems that organize resources and production flows based on time, where any time shift of jobs causes a reorganization of the system. Regardless of the type of cost, some consider only delays, while others consider both delay and anticipation as disruptions. Another disruption cost, relevant in the case of multi-machine scheduling, is given by the change in machine assignment. Again, a change in machine assignment causes a reorganization in the system (e.g., jobs that were ready to be processed on one machine must be transported to another). Given the general purpose of rescheduling in the industry, we now turn to rescheduling for new orders in the way it has been understood as an optimization model.

Rescheduling with a specific focus on addressing the unpredicted arrival of new orders has received much attention and we show the related literature in the following.

This family of rescheduling problems was first formalized by Hall and Potts (2004) that introduce a setting with a single processing machine, on which it is planned to process a set of so-called old jobs. These jobs are scheduled in order to minimize one of the two scheduling functions among total completion time and

maximum lateness. The processing of this optimal schedule is interrupted by the arrival of a new set of jobs, that also need to be scheduled on the same machine. To minimize the objective function, the old jobs are allowed to be re-arranged, causing a disruption, that is measured as the, maximum or total, sequence or time disruption. In this setting, authors study the different problems arising from the combination of an objective function, among total completion time and maximum lateness, and the, maximum or total, sequence or maximum disruption. The eight resulting problems are examined for the cases, where the disruption is modelled as a constraint or in a weighted sum with the scheduling function in the objective. They provide polynomial-time algorithms or show *NP*-hardness for most problems. The problem formalization and definition has been used in the following works.

Yuan and Mu (2007) studied four single machine rescheduling problems with minimization of the makespan in the presence of job release dates and different disruption criteria. They consider both maximum and total sequence or time disruption under the assumption that the initial schedule is optimal with respect to the objective function. Complexity results are studied for the different scenarios. As a result, the problems with the time disruption criteria are shown to be strongly *NP*-hard. Whereas, an algorithm running in polynomial time $O(n_O^2 n)$ is given for the problem with the maximum sequence disruption. The latter is solved using the fact that a so-called weak *Earliest Release Date* ordering property applies and allows to solve the problem in polynomial time. The remaining problem is left open.

The work was later extended in Yuan et al. (2007), where authors consider the rescheduling problem for jobs on a single machine with release dates to minimize total sequence disruption subject to a constraint on the maximum makespan. They prove the considered problem can be solved in polynomial time and so does the complementary problem to minimize makespan under a limit on the total sequence disruption.

Zhao and Yuan (2013) show that finding all Pareto optimal solutions of the bi-objective problem of minimizing the makespan and the total sequence disruption can be done in polynomial time.

Yang (2007) consider rescheduling problems, where to reduce the negative impacts of disruptions to the original schedule, the processing times of the new jobs can be reduced at a time compression cost. The objective of the problem is to minimize total cost after rescheduling, which includes schedule disruption costs,

time compression costs, and a cost that depends on the schedule efficiency. The two scheduling objective functions to measure schedule efficiency are the total completion time and total weighted tardiness. First, they give a polynomial time algorithm for the total completion time case. For the weighted tardiness, known to be strongly *NP*-hard, they provide a very large scale neighborhood search heuristic.

Hall et al. (2007) consider rescheduling problems with multiple disruptions. The setting here changes in the sense that there are multiple sets of new jobs arriving at different times thus, the initial state does not necessarily imply optimal schedules. They show that solving the problem to minimize the maximum lateness under a constraint on the maximum time disruption is *NP*-hard even if there is no new job that arrives. They provide approximation algorithms and a branch and bound algorithm to solve large-size instances.

Zhao and Tang (2010) and Liu and Zhou (2015) consider the single machine rescheduling problem in which jobs have time-dependent processing times. In particular, Zhao and Tang (2010) study the problem to minimize total completion time under a constraint on the maximum sequence or time disruption and provide polynomial-time algorithms. In Liu and Zhou (2015), two problems are studied, the first to minimize the makespan subject to a limit on the maximum sequence disruption and the second to minimize a linear combination of the makespan and the maximum sequence disruption. For each problem, they provide polynomial-time algorithms.

Problems with family setup times are considered in Hoogeveen et al. (2012), where the authors study rescheduling problems with job families. Each job belongs to a family, and changing production from one family to another family f induces a setup time s_f. It is assumed that the total setup time is minimized, which equals minimizing the makespan, by the initial schedule of the old jobs. The new jobs can either be scheduled after the old jobs or inserted into the existing schedule, resulting in a disruption cost among the maximum and total time deviation, and the difference of positions before and after rescheduling, denoted as P_{max}. The authors propose optimal polynomial-time algorithms for enumerating the set of strict Pareto optima for minimizing disruption cost and makespan, or provide NP-hardness proofs.

Teghem and Tuyttens (2014) study the same combinations of scheduling functions and disruption criteria considered in Hall and Potts (2004) and provide algorithms to solve the related bi-objective problems.

In Pai et al. (2014), the problem investigated considers both learning and deterioration effects. The disruption constraints are in the form of the maximum sequence disruption and the maximum time disruption. The objectives are to minimize the total completion time. The authors prove that the two problems can be solved in polynomial time.

Gao et al. (2015) deal with the scheduling and rescheduling problem in remanufacturing. The problem is modelled as a flexible job shop scheduling problem and is divided into two stages: scheduling and rescheduling when a new job arrives. The authors consider both cases where the original jobs are not allowed to have any time disruption or that any operation that has not started yet can be freely rescheduled. The objective is to minimize makespan. A two-stage artificial bee colony heuristic algorithm is proposed for this hard scheduling problem. Extensive computational experiments are carried out and the results show that the heuristic is effective in both scheduling and rescheduling.

Zhao et al. (2016) consider similar problems as in Hall and Potts (2004) but introduce a generalization of the disruption criterion by imposing on each old job j a maximum amount of allowed disruption k_j. In this case, both the problems of minimizing the sum of the completion times and the maximum lateness become NP-hard.

Zhao and Yuan (2017) study the rescheduling on a single machine to minimize the maximum lateness under a job-dependent sequence disruptions. Each old job has a constraint disruption on how much its sequence position change with respect to an original optimal schedule. They consider the three different cases of penalizing disruptions: the case where only an increase in the sequence position causes a disruption, the opposite case where a decrease in the sequence position causes a disruption, and the case where both of them are considered as relevant disruptions. The authors show that the three problems are equivalent and can be solved in polynomial time.

Rahmani (2016) addresses a dynamic flexible flow shop environment considering unexpected arrival of new jobs into the process as disruptions. A novel reactive model is proposed to minimize a linear combination of total weighted tardiness and total absolute time deviation of jobs. The proposed model is presented to generate a stable schedule against any possible occurrences of mentioned disruption. Due to the computational complexity, a variable neighborhood search algorithm is implemented

to solve the problem. To show the performance of the reactive approach, a case study in petrochemical industry is studied. Computational experiments and comparisons of the proposed algorithm with three dispatching rule and an efficient rescheduling approach show the efficiency of the presented reactive approach to reschedule the jobs.

In Liu (2019), it is addressed the two-machine flow shop outsourcing and rescheduling problem. Each job can be processed in the in-house shop or outsourced to a single subcontractor. An efficient schedule is constructed for the in-house jobs, and its performance is measured by the makespan. Each of the outsourced jobs requires paying an outsourcing cost. The objective is to minimize the sum of the in-house makespan and total outsourcing cost, subject to a limit on the number of original jobs processed by the subcontractor after disruption. Initial optimal schedule are obtained with Johnson's rule. To solve this NP-hard problem, a mixed integer programming formulation and helpful optimization properties are first established, and then a hybrid variable neighborhood search algorithm is developed.

Guo et al. (2021) study a rescheduling problem motivated by energy savings in the quartz manufacturing industry. Here, new jobs are rework jobs that need to be added to the original sequence to minimize the total waiting time of the jobs, that we denote as $\sum(C_j - r_j)$. The relative sequence of the original jobs must be preserved and, in addition, each original job is not allowed to deviate from its release time for more than a given threshold. This constraint implicitly models a job-dependent time disruption constraint.

Zhang et al. (2022) took into consideration the minimization of the maximum weighted tardiness for rescheduling problems subject to job-specific and total time disruption constraints. The authors provide complexity results for the problems and several simulated annealing algorithms.

Finally, Fang et al. (2023) studied the rescheduling problem with rejection for minimizing a linear combination of the total weighted completion time, the maximum time disruption and the rejection cost. They first proved the *NP*-hardness of the problem, and then solved it with an exact dynamic programming algorithm and a fully polynomial time approximation algorithm. They showed the efficiency of the proposed algorithms with several computational results.

Table 1.2 summarizes the articles of the literature that focus on rescheduling for new orders, for single machine first, and multiple machines next, and the con-

tributions of this dissertation. Each row first contains the information about the authors and the reference to the articles. Then, the measure of disruption and the objective functions considered are specified. We use Δ_{max}, $\sum\Delta$, D_{max}, $\sum D$ respectively for the maximum and the absolute time disruption, the maximum and the total absolute sequence disruption. We use Δ_j (resp. D_j) for the absolute time (resp. sequence) disruption constraint in the form of $\Delta_j \leq Y_j$ (resp. $D_j \leq Y_j$), where Y_j is a job-dependent threshold, and P_{max} for the difference of positions before and after rescheduling (where a negative difference is not considered as a disruption). Then, the table gives information about whether the initial scheduled is assumed to be optimal (π^*) or not (π), followed by the type of the main contributions given. Among them, we use *cmpx* to denote complexity results, *poly alg* for polynomial algorithms, *heuristic* for heuristic algorihtms, *exact* for exact algorithms, *approx* for approximation algorithms. Finally, the last column shows if there are additional features or characteristics considered.

Most of the above mentioned papers consider mainly a single machine environment and focus on providing theoretical insights on the problems. Among the few researches that consider algorithms for real-world applications, we mention the works by Katragjini et al. (2013) and Valledor et al. (2018). While most of the existing work addresses one type of disrupting event independently, their work considers managing different types of disruptions at the same time.

Katragjini et al. (2013) adopts a managerial point of view, to address real-world manufacturing scheduling. They consider flow shops and three types of disruptions that simultaneously interrupt the original schedules, i.e. machine breakdowns, new job arrivals, and job ready time variations. The authors propose rescheduling methods that seek a good trade-off between the makespan and the schedule instability. The latter is calculated as the sum of operations whose starting times have been anticipated or delayed in the new schedule. The weighted sum method is used to recast the bi-objective problem into a single objective optimization problem.

Valledor et al. (2018) consider rescheduling for minimizing three objectives: makespan, total weighted tardiness and stability. The measure of stability is a function depending on the average absolute time deviation of jobs and of the current scheduling time. such that rescheduling a job that is to be scheduled soon is more penalizing than one that will be scheduled far from the current scheduling time. Three types of disruptions are considered: the arrival of new jobs, machine breakdowns

Table 1.2 Summary of literature articles on rescheduling for new orders

Reference	*Disruption constraint*	*Obj function*	*Initial schedule*	*Contributions*	*Additional features*
Hall and Potts (2004)	Δ_{max}, $\sum\Delta$, D_{max}, $\sum D$	$\sum C_j$, L_{max}	π^*	cmpx	
Yuan and Mu (2007) Yuan et al. (2007)	Δ_{max}, $\sum\Delta$, D_{max}, $\sum D$	C_{max}	π^*	cmpx	r_j
Yang (2007)	$\sum\Delta$	$\sum C_j$, $\sum w_j T_j$	π^*	poly alg, heuristic	
Hall et al. (2007)	Δ_{max}	L_{max}	π	cmpx, approx, exact	
Zhao and Tang (2010)	Δ_{max}, D_{max}	$\sum C_j$	π^*	poly alg	time-dependent p_j
Zhao and Yuan (2013)	$\sum D$	C_{max}	π^*	poly alg	
Liu and Zhou (2015)	D_{max}	C_{max}	π^*	poly alg	time-dependent p_j
Hoogeveen et al. (2012)	Δ_{max}, $\sum\Delta$, P_{max}	C_{max}	π^*	poly alg, cmpx	s_f
Teghem and Tuyttens (2014)	Δ_{max}, $\sum\Delta$, D_{max}, $\sum D$	$\sum C_j$, L_{max}	π^*	exact	
Pai et al. (2014)	Δ_{max}, D_{max}	$\sum C_j$	π^*	poly alg	time-dependent p_j
Zhao et al. (2016)	Δ_j	$\sum C_j$, L_{max}	π^*	cmpx	
Zhao and Yuan (2017)	D_j	L_{max}	π^*	poly alg	
Guo et al. (2021)	Δ_j	$\sum(C_j - r_j)$	π	cmpx, exact, heuristic	r_j
Zhang et al. (2022)	$\sum\Delta$, Δ_j	$\max w_j T_j$	π^*	cmpx, heuristic	
Fang et al. (2023)	Δ_{max}	$\sum w_j C_j$	π^*	cmpx, exact, approx	outsourcing
Gao et al. (2015)		C_{max}	π	heuristic	flexible job shop
Rahmani (2016)	$\sum\Delta$	$\sum w_j T_j$	π^*	heuristic	flexible flow shop
Liu (2019)	outsourced old jobs	C_{max}	π^*	heuristic	two-machine flow shop
This Dissertation	Δ_{max}, $\sum\Delta$	$\sum w_j C_j$, $\sum U_j$ $\sum T_j$, L_{max}	π^*	cmpx,approx,exact, poly alg,heuristic	fixed sequence, non-delay schedules

and variations in job processing times. The adopted strategy is a predictive-reactive strategy, and uses clustering algorithms on the Pareto frontier of the solution space to find a representative solution at each point of rescheduling.

Peng (2019) considers a hybrid flowshop subject to three types of dynamic events (i.e. machine breakdown, new job arrival and job release variation). The mathematical model to minimize the weighted sum of makespan and system instability, where the latter is measured as the sum of jobs that, at any stage, change starting time. Lower and upper bounds of the two optimization objectives are developed. A Multi-Start Variable Neighbourhood Descent algorithm is proposed for the hybrid flowshop rescheduling. Extensive experimental results verify the effectiveness of the algorithms.

If we turn our attention to the literature on rescheduling in general, the choice is vast. Rescheduling for machine unavailability is considered for instance in Liu and Ro (2014), and Luo et al. (2018) for single machine, and in Özlen and Azizoğlu (2011) and Yin et al. (2016) for parallel machines.

Rescheduling for job unavailability, i.e. when a subset of jobs become unavailable for scheduling at their scheduled time, is considered for instance in Hall and Potts (2010) and Luo et al. (2020) for single machine, and in Wang et al. (2018) for parallel machines.

Unal et al. (1997) study a rescheduling problem on a serial batching machine, where new jobs must be included in the original schedule without leading to additional setups and without making any of the existing jobs late. Azizoğlu and Alagöz (2005) study rescheduling on parallel machines to minimize the total completion time and the number of jobs rescheduled. They consider the number of jobs scheduled on different machines with respect to the original plan as a disruption measure. Liu et al. (2018) investigate the rescheduling on a two-machine flow shop with outsourcing solved by means of a hybrid variable neighbourhood search. They formulate the problem as a scheduling game. Nicosia et al. (2021) consider rescheduling of a given schedule to minimize classical objective functions in an automated system where the re-sequencing of jobs is allowed by the usage of LIFO buffers. They provide several algorithms and complexity results for a whole set of problems. A follow-up paper considering exact algorithms for the NP-hard case of minimizing the weighted number of late jobs can be found in Pferschy et al. (2022).

1.5 Contributions and structure of the dissertation

We consider single machine rescheduling problems for new orders to minimize one of several scheduling objective functions subject to a disruption constraint on the maximum or total absolute time deviation. Rescheduling for new orders with a constraint on the absolute time deviation has been the model that has attracted the most research in this area. After the seminal work of Hall and Potts (2004), many other papers have studied various special cases. However, the research has been scattered and there are still a lot of open questions. The goal of the dissertation is to provide an insight into the entire class of these problems in order to understand the issues involved in solving them. Since solving a scheduling problem is a typical combinatorial optimization problem, we use methods and techniques that come from this field. The contributions are presented in three different chapters, each focusing on one aspect.

In Chapter 2, we provide a systematic analysis of rescheduling problems, both in terms of structure and complexity. The focus is on providing general properties for rescheduling for new orders to give an overall understanding of this set of problems with respect to structural properties and computational complexity. The first aspect considers the ordering of the set of old jobs. There is a subset of problems in which preserving a known valid ordering is enough to reach optimal solutions. Several examples support the different behaviours of this set and the remaining problems. The second aspect considers the need of idle time insertion in optimal solutions. Based on this, we propose a classification of rescheduling problems according to whether machine idle times have to be inserted in optimal solutions or not. The study of the computational complexity is given in the third part of the chapter. The chapter is based on the content of two submitted works, Rener et al. (2023) and Lendl et al. (2023).

Chapter 3 is devoted to the study of the timing problem inherent in some rescheduling problems. Structural analysis reveals the existence of some problems in which the inclusion of idle time allows the decrement of the objective function. In the chapter, the goal is to study the problem to understand the degree of complexity it adds to rescheduling problems with idle time. Several algorithms are proposed, one for each of the timing problems that arise in rescheduling, which show that timing in rescheduling is a polynomial problem. The chapter is based on Rener et al. (2023).

In Chapter 4, we study the special case of rescheduling to minimize the total weighted completion time subject to a maximum disruption constraint. The problem lies at the intersection between easy polynomially solvable problems and strongly *NP*-hard problems, when the maximum disruption is considered, which makes it of a particular interest. It obeys to the ordering property of old jobs presented in Chapter 2, but it is already *NP*-hard. Therefore, we devote the chapter to this problem and we provide three algorithms: an exact algorithm running in pseudo-polynomial time, that uses dynamic programming, a fully polynomial approximation scheme and a bounded-ratio approximation algorithm. The chapter is based on Lendl et al. (2023).

In Chapter 5, we study the special case of rescheduling with a non-delay schedule to minimize the maximum lateness under a total disruption constraint. The problem is strongly *NP*-hard and is solved using techniques from combinatorial optimization. In particular, a branch and memorize algorithm is developed that takes advantage of a generic structure and specific properties. Using efficient exploration of the solution space, the algorithm is the first exact search tree algorithm for a hard rescheduling problem. Computational experiments in which the algorithm is compared with a commercial solver for optimization problems show that the algorithm is indeed able to exploit the specific features of the problem and to integrate them into a more efficient solution of the problem. The chapter is based on the work found in Rener et al. (2022).

The dissertation gives an introduction to some relevant topics to be considered in rescheduling problems and leaves the door open for further developments. Thus, chapter 6 summarizes the conclusions of the work presented and discusses several future work directions that should be considered.

Chapter 2

Structural properties and complexity of rescheduling problems

The focus in this chapter is on providing general properties to give an overall understanding of rescheduling problems for new orders with time disruption constraints. The first section provides some considerations on the cases where there are ordering rules that are valid for the set of old jobs in optimal solutions. Several examples are provided for supporting the considerations done. The second aspect considers the need of idle time insertion in optimal solutions, i.e. when the problem formulation leads to unforced idleness on the machine in optimal solutions. Often in single machine scheduling is enough to consider the job permutation problem to find optimal solutions. As opposed to this, it is shown that for several rescheduling problems idle time insertion is necessary to obtain optimal solutions. We propose a classification of rescheduling problems according to whether machine idle times are to be inserted in optimal solutions or not. Again, several examples support the results obtained. Finally, in the last part of the chapter the problems are studied in terms of computational complexity. The results show that most problems are intractable, with few exceptions, and provide for these special cases polynomial-time algorithms.

2.1 Ordering properties

Old jobs are initially sequenced in a schedule π^* to minimize an objective function f. This order might be preserved in an optimal schedule of the rescheduling problem. In

this section, we focus on establishing some properties of when this order is preserved, as opposed to when a *re-sequencing* of old jobs is required.
The first two properties illustrated below serve a dual purpose. The first is to show the source of the need for re-sequencing. The second, in a bi-objective perspective, is to provide actual structural properties for problems $1|f \leq Y|\Delta_{max}$ and $1|f \leq Y|\sum\Delta$. The two disruption criteria lead to different behaviours. In fact, we first show that problem $1|f \leq Y|\Delta_{max}$ never requires a re-sequencing to obtain an optimal solution, if any feasible schedule exists with old jobs ordered as in π^*. On the contrary, we show that problem $1|f \leq Y|\sum\Delta$ may require a re-sequencing of old jobs, even if there exists a feasible schedule with old jobs ordered as in π^*.

Property 2.1.1. *If there exists a non-empty set of feasible schedules for problem $1|f \leq Y|\Delta_{max}$ with old jobs ordered as in π^*, there exists an optimal solution that belongs to this set of schedules.*

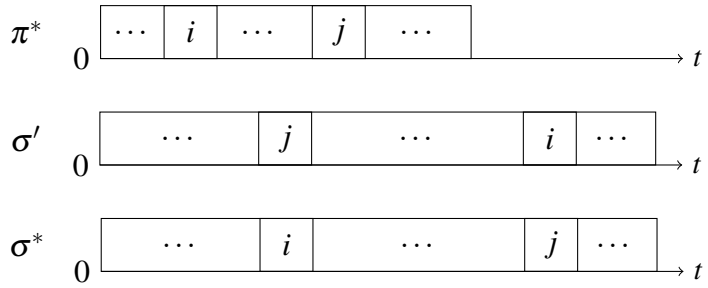


Figure 2.1 Jobs sequencing for the $1|f \leq Y|\Delta_{max}$ problem.

Proof. Consider Figure 2.1. Schedule π^* shows the original optimal schedule with old jobs only. Schedule σ' is a schedule with old and new jobs, where at least two old jobs i and j are swapped with respect to their ordering in π^*. Suppose that σ' is an optimal solution for the problem $1|f \leq Y|\Delta_{max}$. Let i be the first old job that has been moved after other old jobs that followed i in π^*, and let j be the old job that precedes i in σ' with respect to old jobs only. Notice that in σ', before job j there can be both old and new jobs, while between j and i there can be only new jobs because j is the closest old job that precedes i in σ'. Let σ^* be the same schedule than σ' but with i and j swapped, such that the starting time of j in σ' is the same of i in σ^* and $C_i(\sigma') = C_j(\sigma^*)$. In σ^*, before job i there can be both old and new jobs, while between i and j there can be new jobs only. Since σ' is assumed to be optimal, $\Delta_{max}(\sigma') \leq \Delta_{max}(\sigma^*)$.

The proof is done by contradiction showing that, if there exists such a schedule σ^*, with $f \leq Y$, then $\Delta_{max}(\sigma^*) < \Delta_{max}(\sigma')$.

First, notice that $\Delta_i \geq 0$ in both schedules by definition of job i and that $\Delta_i(\sigma^*) < \Delta_i(\sigma')$ since $C_i(\sigma') > C_i(\sigma^*)$. Given that $C_j(\pi^*) > C_i(\pi^*)$ and $C_i(\sigma') = C_j(\sigma^*)$, we have that $\Delta_j(\sigma^*) < \Delta_i(\sigma')$. Next, we show that $\Delta_j(\sigma') \leq \Delta_i(\sigma')$. If $C_j(\sigma') \geq C_j(\pi^*)$, then this holds, since i is right-shifted in time more than j. If $C_j(\sigma') < C_j(\pi^*)$, we have $\Delta_j(\sigma') \leq C_j(\sigma') - C_j(\sigma^*)$. For the disruption of job i it holds $\Delta_i(\sigma') \geq C_i(\sigma') - C_i(\sigma^*)$ and since $C_j(\sigma') - C_j(\sigma^*) = C_i(\sigma') - C_i(\sigma^*)$, we have $\Delta_j(\sigma') \leq C_j(\sigma') - C_j(\sigma^*) = C_i(\sigma') - C_i(\sigma^*) \leq \Delta_i(\sigma')$.

Putting all together, we obtain

$$\max(\Delta_i(\sigma'), \Delta_j(\sigma')) = \Delta_i(\sigma') > \max(\Delta_i(\sigma^*), \Delta_j(\sigma^*)),$$

and since all other old jobs do not change their starting and completion times,

$$\Delta_{max}(\sigma') > \Delta_{max}(\sigma^*).$$

Repeating the argument for any pair of old jobs i and j defined as above proves the statement. □

Going back to the rescheduling problems with the Δ_{max} constraint, we use Property 2.1.1 to derive valid ordering properties. It was shown that for objective functions of total completion times and maximum lateness no re-sequencing occurs (Hall and Potts (2004)). We extend this result to the case of total weighted completion times.

Property 2.1.2. *For problem* $1|\Delta_{max} \leq Y|\sum w_j C_j$ *there exists an optimal schedule where old jobs are sequenced by WSPT, as in the original schedule* π^*.

Proof. Assume that there is an optimal schedule σ' where the old jobs violate the WSPT property. Let $\Omega = \sum_{j \in O \cup N} w_j C_j(\sigma')$ be the objective function value of schedule σ'. Restoring the WSPT order among old jobs never increases the objective function value thus leading to an objective function value $\sum_{j \in O \cup N} w_j C_j(\sigma^*) \leq \Omega$. It follows that if σ' is a feasible solution to the complementary problem $1|\sum w_j C_j \leq Y|\Delta_{max}$, then schedule σ^* is it too. Then, by Property 2.1.1 $\Delta_{max}(\sigma') > \Delta_{max}(\sigma^*)$. Hence, restoring the WSPT order is feasible and optimal. □

Property 2.1.2 states that beside the problems studied by Hall and Potts (2004), also problem $1|\Delta_{max} \leq Y|\sum w_j C_j$ answers to a known ordering, found in polynomial time, of old jobs in optimal solutions. This makes the problem easier to solve and, in particular, allows to identify a very specific structure of optimal solutions that will be shown in Chapter 4 and exploited to show that the problem can be solved in pseudo-polynomial time.

For the remaining objective function $\sum U_j$ and $\sum T_j$, there is no valid ordering that applies to the old jobs. We show in Section 2.3 that this will make the problem with $f = \sum U_j$ harder to solve. For $f = \sum T_j$ there is no such result but despite there is no stated proof for its higher complexity, the absence of such property makes hard to believe that there exists a pseudo-polynomial algorithm for this problem.

Let us turn to the $\sum \Delta$ criterion and consider the complementary problem $1|f \leq Y|\sum \Delta$. The following property holds.

Property 2.1.3. *For problems $1|f \leq Y|\sum \Delta$, an optimal schedule may have old jobs scheduled in a different order with respect to π^* even if swapping them keeps the schedule feasible with respect to the constraint $f \leq Y$.*

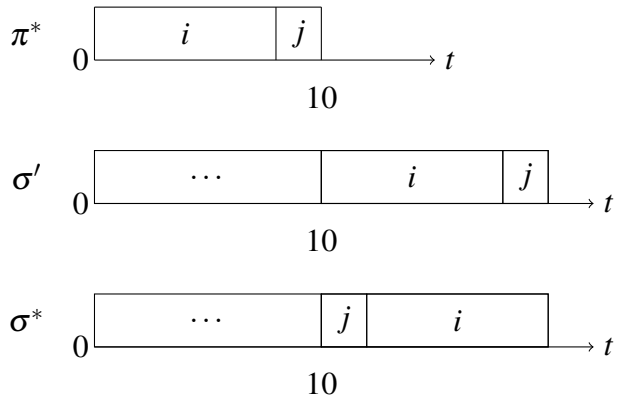


Figure 2.2 Jobs sequencing in the $1|f \leq Y|\sum \Delta$ problem.

Proof. Consider the following example. Two old jobs i and j with $p_i = 8$ and $p_j = 2$ are scheduled consecutively in π^* so that $C_i(\pi^*) = 8$ and $C_j(\pi^*) = 10$ (Figure 2.2). Let be $d_i = C_i(\pi^*)$ and $d_j = C_j(\pi^*)$. Schedule σ' is a feasible schedule with old and new jobs, where job i precedes job j, as in the original schedule π^*. Let σ^* be another feasible schedule, with all jobs scheduled as in σ', except for jobs i and j,

which are swapped. Assume that the starting time of job i in σ' as well as the starting time of job j in σ^* is $t = 10$. Computing the total disruption in the two schedules, we obtain

$$\sum \Delta(\sigma') - \sum \Delta(\sigma^*) = 20 - 14 > 0$$

Since any other new job scheduled does not have an effect on the total disruption, σ^* is an optimal schedule. □

2.2 Compact and non-compact schedules

Usually, minimizing a regular objective function allows to consider only the sequencing or permutation problem in scheduling. However, the constraint on the time disruption in rescheduling problems implicitly models a second objective, which is non regular. Therefore, unforced idleness on the machine may appear in optimal solutions. Several considerations and examples are given in this section to provide insights on when the insertion of machine idle times is required.

Consider the following example. Three old jobs are originally scheduled in $\pi^* = (i, j, k)$ to minimize the total weighted tardiness $\sum w_j T_j$. Let be $p_i = 6, p_j = p_k = 1$, $d_i = 6, d_j = 7, d_k = 8$ and $w_i = 1, w_j = w_k = 2$. Then, the minimum cost schedule with no inserted idle times, that includes a new job h, with $p_h = 1$, $d_h = 0$ and $w_h = 1000$, is $\sigma^*_{no-idle} = (h, j, k, i)$ as in Figure 2.3.In schedule $\sigma^*_{no-idle}$, we have $C_j(\pi^*) - C_j(\sigma^*_{no-idle}) > C_i(\sigma^*_{no-idle}) - C_i(\pi^*)$ and $C_k(\pi^*) - C_k(\sigma^*_{no-idle}) > C_i(\sigma^*_{no-idle}) - C_i(\pi^*)$. In this case, constructing a schedule σ^*_{idle} with one unit of idle time before job j reduces both Δ_{max} and $\sum \Delta$, while keeping the increase of $\sum w_j T_j$ lower with respect to other job permutations. So, in this example, for the Δ_{max} criterion and $Y = 5$, as well as for the $\sum \Delta$ criterion and $Y = 14$, schedule σ^*_{idle} makes this sequence of jobs feasible and optimal.

The example gives a first feeling of the correlation between the re-sequencing of old jobs and the insertion of idle times to obtain optimal solutions. Notice that the decrease of the disruption obtained by right-shifting jobs is due by the fact that there are two jobs, j and k, that are scheduled before their original completion time and for which a right-shift is beneficial in terms of disruption, and only one job, i, that is scheduled after and for which a right-shift is disadvantageous. So, clearly, no job

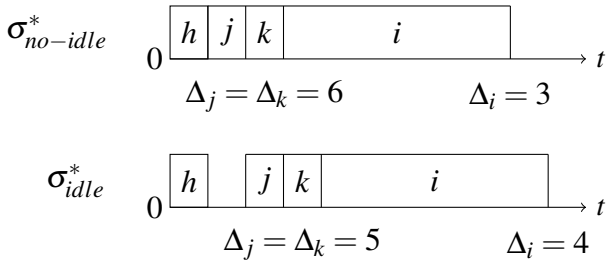


Figure 2.3 Decrease of schedule disruption via idle time insertion.

can be scheduled earlier with respect to its original completion time, if all jobs keep their initial ordering.

In fact, when the old jobs are sequenced as in the original schedule, the following property applies to an optimal schedule.

Property 2.2.1. *If there exists an optimal solution in which old jobs are ordered as in the original schedule π^*, then this solution is without inserted idle times.*

Proof. Since jobs in O are in the same order as in schedule π^*, the disruption of any job j in an optimal schedule σ^* is given by $\Delta_j = \sum_{i \in N: i \to j} p_i$, where $i \in N : i \to j$ denote the new jobs i that precede j in the schedule. By inserting an idle time $\delta > 0$ before job j, the disruption changes to $\Delta'_j = \sum_{i \in N: i \to j} p_i + \delta$. Then, in this new schedule σ', $\Delta_{max}(\sigma') \geq \Delta_{max}(\sigma^*)$ as well as $\sum \Delta(\sigma') > \sum \Delta(\sigma^*)$. Also, $f(C_j + \delta) > f(C_j),\ \forall j \in O \cup N$ since the f_j's are regular functions. Then, $f_{max}(\sigma') \geq f_{max}(\sigma^*)$ as well as $\sum f(\sigma') > \sum f(\sigma^*)$. Therefore, neither the regular objective functions nor the disruption criteria are reduced by inserting any idle time. □

We saw in Section 2.1 that there are some problems for which the ordering of old jobs is known and that is the same as in the original optimal schedule. We derive the following property.

Property 2.2.2. *There exists an optimal solution to problems $1|\Delta_{max} \leq Y|f$, $f \in \{L_{max}, \sum C_j, \sum w_j C_j\}$, and $1|\sum \Delta \leq Y|\sum C_j$, where the jobs are processed consecutively without inserted idle times.*

Proof. It is shown that in optimal solutions for the cases with objective function L_{max} and $\sum C_j$ (Hall and Potts (2004)) and $\sum w_j C_j$ (Property 2.1.2) no re-sequencing is applied to the old jobs. Therefore, in all four cases, Property 2.2.1 applies and it is not necessary to consider idle time insertion in optimal solutions. □

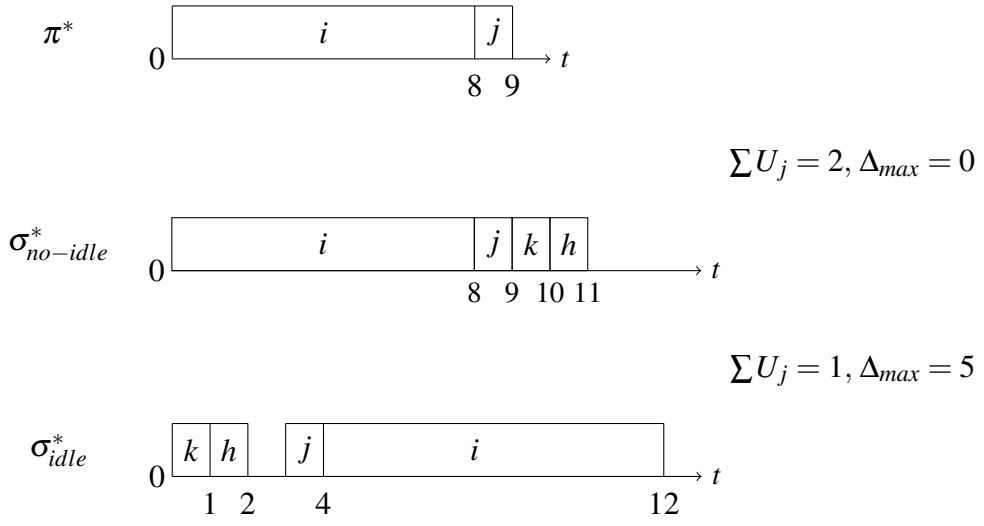


Figure 2.4 Idle time insertion for the $1|\Delta_{max} \leq Y|\sum U_j$ problem

Conversely, for remaining problems where the order of old jobs is unknown, we state the following property.

Property 2.2.3. *An optimal solution to problems* $1|\Delta_{max} \leq Y|f_1$, $f_1 \in \{\sum U_j, \sum T_j\}$, *and* $1|\sum \Delta \leq Y|f_2$, $f_2 \in \{L_{max}, \sum w_j C_j, \sum U_j, \sum T_j\}$ *may contain inserted idle times.*

Proof. To prove each of the stated results we provide, for each problem, an example that shows the optimality of a schedule with inserted idle time.

We start with problem $1|\Delta_{max} \leq Y|\sum U_j$. Consider the schedule π^* in Figure 2.4, with two old jobs i and j, with $p_i = 8$, $p_j = 1$ and $d_i = 8$, $d_j = 9$ and two new jobs k and h with $p_k = p_h = 1$ and due dates $d_k = d_h = 8$. Let be $Y = 5$. An optimal solution for this instance is given by schedule σ_{idle} with $\sum U_j = 1$, and where the idle time enables to meet the disruption constraint. In contrast, restricting to the set of schedules without inserted idle times leads to the optimal solution $\sigma_{no-idle}$ with $\sum U_j = 2$.

Next, we consider the problem $1|\Delta_{max} \leq Y|\sum T_j$. Consider jobs i, j, k, h from the above example and $Y = 5$. Consider an additional old job ℓ with $p_\ell = 1$ and $d_\ell = 10$. Schedule π^* is shown in Figure 2.5. The optimal schedule reaches $\sum T_j = 5$. In contrast, restricting to the set of schedules without inserted idle times leads to the optimal solution $\sigma_{no-idle}$ with $\sum T_j = 7$.

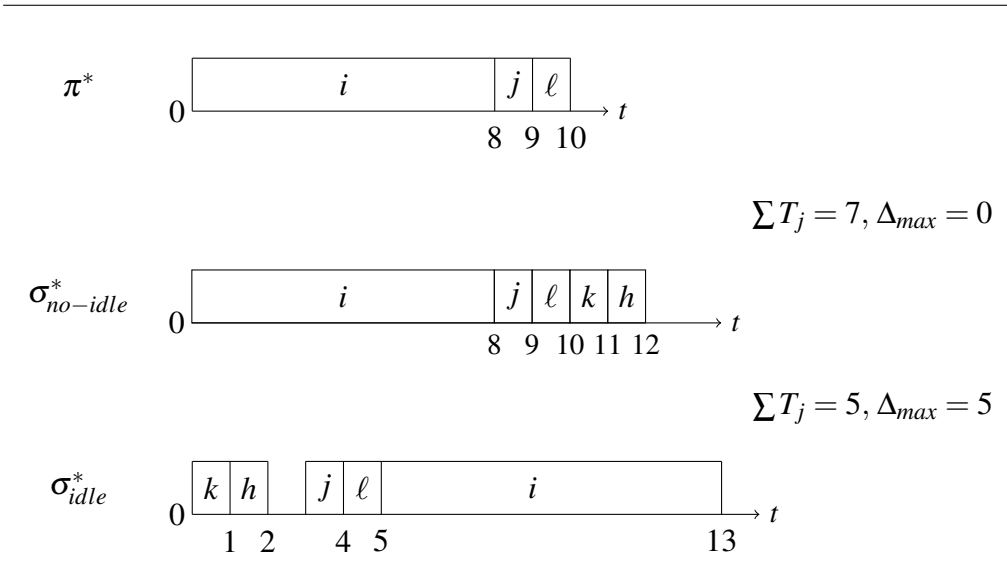


Figure 2.5 Idle time insertion for the $1|\Delta_{max} \leq Y|\sum T_j$ problem

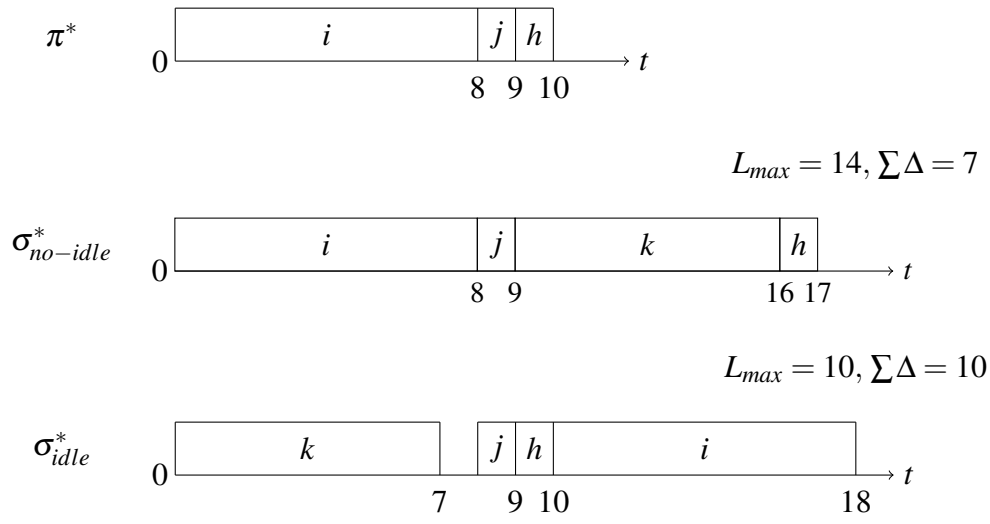


Figure 2.6 Idle time insertion for the $1|\sum \Delta \leq Y|L_{max}$ problem

Now, we turn to the problem $1|\sum\Delta \leq Y|L_{max}$. Let be i, j, h the jobs in O with $p_i = 8, p_j = p_h = 1$ and $d_i = 12, d_j = 13, d_h = 14$ and k a new job with $p_k = 7$ and $d_k = 1$. Let us have $Y = 10$. As illustrated in Figure 2.6, we assume that $\pi^* = (i, j, h)$. the optimal schedule is $\sigma^*_{idle} = (k, j, h, i)$ with $L_{max} = 10$ with an idle time of 1 unit before job j. In contrast, restricting to the set of schedules without inserted idle times leads to the optimal solution $\sigma_{no-idle}$ with $L_{max} = 14$.

Next, we consider the problem $1|\sum\Delta \leq Y|\sum w_jC_j$. Consider jobs $i, j, h \in O$ and $k \in N$ with $p_i = 6, p_j = p_h = 1, p_k = 4$ and $w_i = 7, w_j = w_h = 1, w_h = 12$. For $Y = 7$, the optimal schedule is $\sigma^*_{idle} = (k, j, h, i)$ with $\sum w_jC_j = 139$ with an idle time of 1 unit before job j (Figure 2.7). However, by imposing that no inserted idle time is allowed, the optimal solution becomes $\sigma^*_{no-idle}$ with $\sum w_jC_j = 193$.

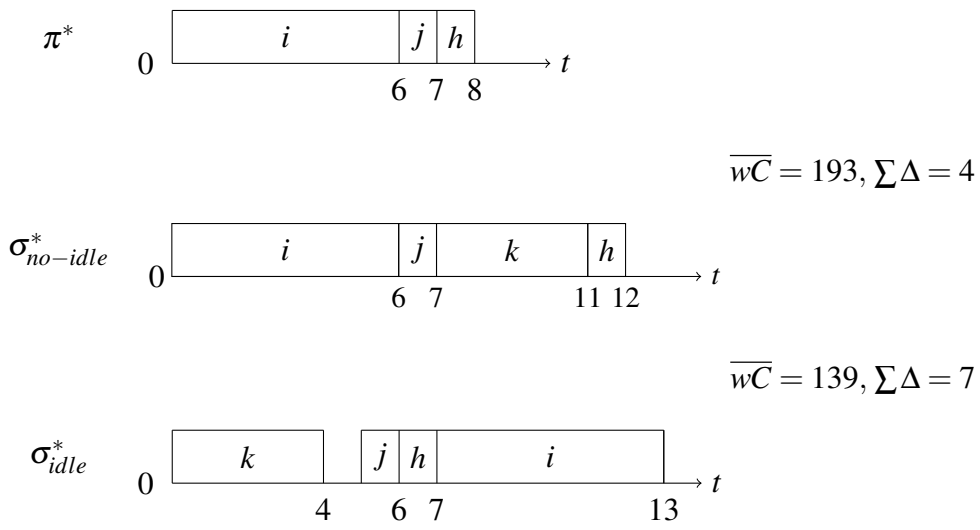


Figure 2.7 Idle time insertion for the $1|\sum\Delta \leq Y|\sum w_jC_j$ problem

Finally, we consider problems $1|\sum\Delta \leq Y|\{\sum U_j, \sum T_j\}$. Set O is made up of the three jobs i, j, ℓ with $p_i = 8$, $p_j = p_\ell = 1$ and $d_i = 8$, $d_j = 9$, $d_\ell = 10$ and two new jobs k and h with $p_k = p_h = 1$ and $d_k = d_h = 8$. Let be $Y = 5$ and the optimal solution with $\sum U_j = 1$ is given by schedule $\{k, h, j, \ell, i\}$ with one unit of idle time before j (Figure 2.8). However, by imposing that no inserted idle time is allowed, the optimal solution becomes $\sigma_{no-idle}$ with $\sum U_j = 2$.

The same counterexample holds for $\sum f_j = \sum T_j$, with a resulting objective $\sum T_j = 5$ of an optimal schedule with idle time and $\sum T_j = 7$ when imposing a constraint of no inserted idle time. □ □

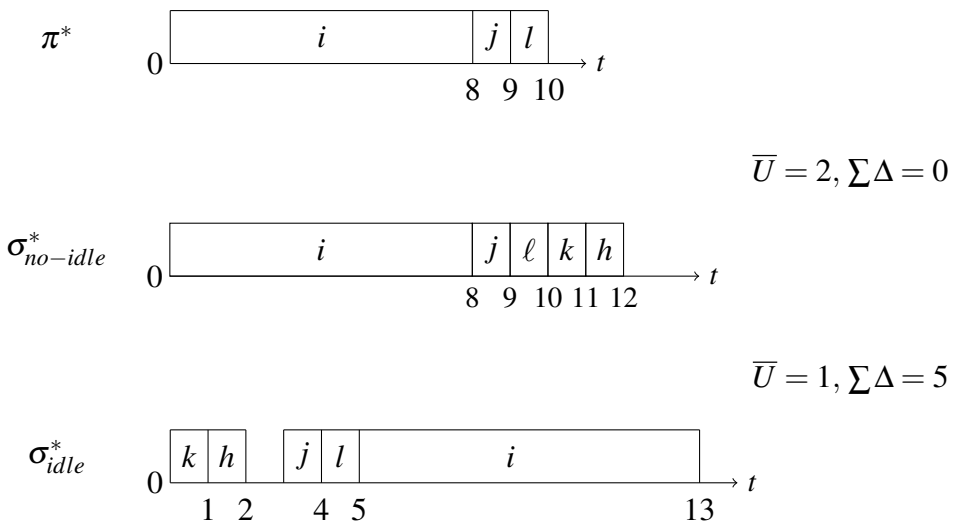


Figure 2.8 Idle time insertion for the $1|\sum \Delta \leq Y|\sum U_j$ and $1|\sum \Delta \leq Y|\sum T_j$ problems

Table 2.1 summarizes the above results: for each pair of objective function and disruption constraint, we indicate *idle times* when inserting machine idle times may be necessary to compute optimal solutions. We indicate *no idle times* for problems whose optimal solutions are without inserted idle times.

Table 2.1 Problem classification w.r.t. the insertion of machine idle times.

f	Δ_{max}	$\sum \Delta$
L_{max}	no idle times	idle times
$\sum C_j$	no idle times	no idle times
$\sum wC$	no idle times	idle times
$\sum U_j$	idle times	idle times
$\sum T_j$	idle times	idle times

2.3 Complexity results

In this section we set the computational complexity of several open problems and analyze some special polynomially-solvable problems.

We first introduce the definition of the partition problem (PARTITION) and the three partition problem (3-PARTITION), two decision problems which we use to state the complexity of rescheduling problems.

PARTITION is a decision problem that answers the following question: given a set $A = a_1, a_2, \ldots, a_m$ of positive integer numbers, is there a partition of the elements into two subsets S_1 and S_2, such that $\sum_{i \in S_1} a_i = \sum_{i \in S_2} a_i = B$? PARTITION is weakly *NP*-complete (Garey and Johnson (1979)).

3-PARTITION is another decision problem that answers to a different question: given $3t$ elements $a_1, \ldots, a_{3t}$ with $\sum_{i=1}^{3t} a_i = ty$ and $y/4 < a_i < y/2$ for $i = 1, \ldots, 3t$, is there a partition of the elements into t sets $S_1, \ldots, S_t$ such that $\sum_{i \in S_k} a_i = y$, $k = 1, \ldots, t$? 3-PARTITION is strongly *NP*-complete (Garey and Johnson (1979)).

The reductions from 3-PARTITION that will follow in the chapter will set the complexity of some problems to strong *NP*-hardness. The reductions given from PARTITION will instead prove some problems to be *NP*-hard *at least* in the weak sense, leaving them open for setting weak or strong *NP*-hardness (when possible, the matter will be investigated in the following chapters).

2.3.1 Minimizing the total weighted completion time subject to a maximum time disruption constraint

The rescheduling problem $1|\Delta_{max} \leq Y|\sum w_j C_j$ is next shown to be *NP*-hard in the weak sense.

Theorem 2.3.1. *Problem* $1|\Delta_{max} \leq Y|\sum w_j C_j$ *is NP-hard.*

Proof. The proof is given by reduction from PARTITION. Consider an instance of the $1|\Delta_{max} \leq Y|\sum w_j C_j$ problem with one old job indexed with 1 and $n_N = m$ new jobs indexed with $2, \ldots, m+1$ and the following weights and processing times.
$w_{1+i} = p_{1+i} = a_i$, $\forall i = 1, \ldots, m$,

$w_1 = 1,\ p_1 = 3B^2 - B - 1,$
$Y = B, z = 3B^2 + B + Bp_1 + p_1$

We show that there is a solution of $1|\Delta_{max} \leq Y, \sum w_j C_j \leq z|-$, i.e. with objective value of at most z, with if and only if there exists a solution to the PARTITION problem.

If solution sets S_1 and S_2 to PARTITION exists, then we claim that schedule σ in Fig. 2.9 gives a positive answer to the decision problem $1|\Delta_{max} \leq Y, \sum w_j C_j \leq z|-$. The rescheduling constraint is fulfilled with equality. We compute an upper bound

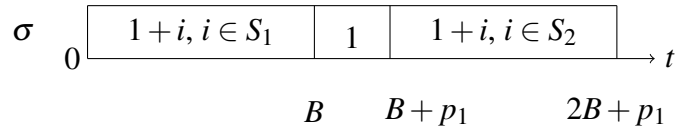


Figure 2.9 Schedule solving $1|\Delta_{max} \leq Y, \sum w_j C_j \leq z|-$ when a solution to PARTITION exists.

on $f(\sigma)$ as follows:

$$\begin{aligned} f(\sigma) &= \sum_{j \in S_1} w_j C_j + B + p_1 + \sum_{j \in S_2} w_j C_j \\ &\leq (\sum_{j \in S_1} w_j) B + B + p_1 + (\sum_{j \in S_2} w_j)(2B + p_1) \\ &= B^2 + B + p_1 + 2B^2 + Bp_1 \\ &= 3B^2 + B + Bp_1 + p_1 = z \end{aligned}$$

On the other hand, we show that if there is no solution to PARTITION, a positive answer to the decision problem $1|\Delta_{max} \leq Y, \sum w_j C_j \leq z|-$ does not exist.

Consider an arbitrary feasible solution σ' of the rescheduling problem. Let set L (resp. R) be the non empty set of new jobs scheduled before (resp. after) job 1 (the trivial case $L = \emptyset$ can be ruled out by a simple calculation). In order to fulfill the disruption constraint there must be $W_L = P_L \leq Y = B$. Since there is no solution to PARTITION, i.e. $\sum_{j \in L} w_j \neq \sum_{j \in R} w_j \neq B$, this implies $\sum_{j \in R} w_j \geq B + 1$. Let us

compute a bound on $f(\sigma')$ based on $C_j \geq 1$ for $j \in L$ and $C_j \geq 2 + p_1$ for $j \in R$:

$$\begin{aligned}
f(\sigma') &= \sum_{j\in L} w_j C_j + (\sum_{j\in L} p_j + p_1) + \sum_{j\in R} w_j C_j \\
&\geq (\sum_{j\in L} w_j) + 1 + p_1 + (\sum_{j\in R} w_j)(2 + p_1) \\
&= 2B + 1 + p_1 + (\sum_{j\in R} w_j)(1 + p_1) \\
&\geq 2B + 1 + p_1 + (B+1)(1 + p_1) \\
&= 3B + Bp_1 + 2p_1 + 2 \\
&= Bp_1 + 3B + p_1 + 2 + (3B^2 - B - 1) \\
&= 3B^2 + Bp_1 + 2B + p_1 + 1 > z
\end{aligned}$$

This shows that if there exist no sets S_1 and S_2 of elements a_j solving PARTITION then any feasible schedule σ' for $1|\Delta_{max} \leq Y|\sum w_j C_j$ has an objective function value greater than z. □

Complexity of Special Cases

We can directly extend the general *NP*-hardness result of theorem 2.3.1 to the case with one old job and constant weight/profit ratios, since the proof uses an instance of $1|\Delta_{max} \leq Y|\sum w_j C_j$ where there is one old job and all new jobs have the same $\frac{w_j}{p_j} = k$ ratio.

Corollary 2.3.1. *The problem* $1|\Delta_{max} \leq Y, |\sum w_j C_j$ *is NP-hard even when all the new jobs have the same ratio* $\frac{w_j}{p_j}$ *and there is one old job.*

Considering the *NP*-hardness of $1|\Delta_{max} \leq Y|\sum w_j C_j$, even for the special case given above, it makes sense to identify polynomially solvable special cases. Recall that the case with unit weights, i.e. $1|\Delta_{max} \leq Y|\sum C_j$, has been shown by Hall and Potts (2004) to be optimally solvable in polynomial time. We will extend this result and consider the case of equal processing times and the case with a "strong ordering" property. For the latter, we call two vectors w and p agreeable, denoted as $agr(w, p)$, if $w_i \geq w_j$ always implies $p_i \leq p_j$ for all pairs of indices i, j.

In the following we will introduce a straightforward Greedy algorithm, which considers all jobs in WSPT order and produces a Greedy schedule σ_G as follows: old jobs are immediately added to the sequence, new jobs only if their addition does not

violate the rescheduling constraint. Otherwise, they are put aside and finally added at the end of the sequence in WSPT order (see Algorithm 1).

Algorithm 1 Algorithm GREEDY for $1|\Delta_{max} \leq Y|\sum w_j C_j$

σ_G is the empty sequence
$L := \emptyset$, $R := \emptyset$
Order and index all jobs by WSPT.
for $j = 1, \ldots, n$ **do**
  if $j \in O$ **then**
    append job j to sequence σ_G
  else if $j \in N$ and $P_L + p_j \leq Y$ **then**
    $L := L \cup \{j\}$
    append job j to sequence σ_G
  else
    $R := R \cup \{j\}$
  end if
end for
append all jobs of R to sequence σ_G

Clearly, this Greedy algorithm can be performed in $O(n \log n)$ time. We can show the following statement.

Theorem 2.3.2. *The greedy schedule σ_G is an optimal solution for problem $1|\Delta_{max} \leq Y, agr(w,p)|\sum w_j C_j$.*

Proof. The Greedy schedule σ_G produces a set L of new jobs, scheduled before the last old job, having largest weights and smallest processing times, and a set R of new jobs scheduled in WSPT after old jobs. For agreeable weights and processing times, jobs ordered by WSPT are also ordered by non-increasing weights as well as by non-decreasing processing times. Notice that since old jobs are ordered in WSPT, $\Delta_j \leq P_L$ for any old job.

Assume that such a solution is not optimal, i.e. there exists a schedule σ' with a lower objective function value.

If two jobs from O and L are swapped with each other, then the objective function value does never decrease because of the violated WSPT order. If the swap is between two old jobs, then the schedule is suboptimal with respect to the disruption constraint by Property 2.1.2. If the swap is between an old job and a job in L, or between two jobs in L, the disruption still satisfies $\Delta_j \leq P_L$ for any old job.

If two new jobs from R are swapped with each other, then the objective function value does never decrease because of the violated WSPT order and the disruption stays unchanged therefore, the swap is again never optimal.

Otherwise, assume that there exists an optimal solution with a different set of jobs L'. If L' is a strict subset of L, this can not give a better solution, since the first job in R must be a job in $L \setminus L'$ and could be moved from R to L, thus improving the objective function due to the WSPT ordering. Therefore, there must exist a new job $j \in L' \setminus L$, and thus also a new job $i \in L \setminus L'$, since $p_j \geq p_i$ for all $i \in L$ and given the maximality of L. We can write the schedule as $\sigma' = \alpha\, j\, \beta\, i\, \gamma$ with $w_i > w_j$ and $p_i \leq p_j$, where α and β are sub-sequences made by old and new jobs and γ is a sub-sequence made by new jobs only. Construct schedule σ^* by swapping i and j, so that $\sigma^* = \alpha\, i\, \beta\, j\, \gamma$. Then $f(\sigma') - f(\sigma^*) = w_i(P_\beta + p_j) - w_j(P_\beta + p_i) + W_\beta(p_j - p_i) = (w_i - w_j)P_\beta + (w_i p_j - w_j p_i) + W_\beta(p_j - p_i)$, which is positive, since all terms in parentheses are positive by definition of i and j. Also, since $p_i \leq p_j$ there is $\Delta_{\max}(\sigma^*) \leq \Delta_{\max}(\sigma') \leq Y$. By repeating the same argument for a finite number of times we obtain schedule σ_G from σ' and the statement is proved. □

If the processing times of the new jobs are all equal, then the WSPT order implies an ordering by non-increasing weights. Trivially, the weights and processing times are agreeable and therefore Theorem 2.3.2 implies the following corollary.

Corollary 2.3.2. *The problem* $1|\Delta_{max} \leq Y, p_j^N = p|\sum w_j C_j$ *is optimally solved in polynomial time by schedule* σ_G.

For general instances, one can show that the output of Greedy can be arbitrarily bad with respect to the optimal solution.

Property 2.3.1. *Algorithm Greedy does not have a bounded performance ratio for* $1|\Delta_{max} \leq Y|\sum w_j C_j$.

Proof. We define an instance related to the worst-case instance for the Greedy algorithm for the well-known 0-1 knapsack problem (refer to Kellerer et al. (2004)). The instance consists of one old job 1 and two new jobs 2 and 3 with $p_1 = M^2, w_1 = 1$, $p_2 = 1, w_2 = 2$, $p_3 = M, w_3 = 2M - 1$ and $Y = M$. We have $\frac{w_2}{p_2} > \frac{w_3}{p_3} > \frac{w_1}{p_1}$. Then, the schedule returned by Algorithm Greedy is $\sigma_G = (2,1,3)$. The total weighted

completion time of such schedule is

$$\begin{aligned} f(\sigma_G) &= 2+M^2+1+(2M-1)(M^2+M+1) \\ &= 2+M^2+1+2M^3+2M^2+2M-M^2-M-1 \\ &= 2M^3+2M^2+M+2. \end{aligned}$$

An alternative solution is given by schedule $\sigma_{opt} = (3,1,2)$ with a total weighted completion time of

$$\begin{aligned} f(\sigma_{opt}) &= M(2M-1)+M^2+M+2(M^2+M+1) \\ &= 2M^2-M+M^2+M+2M^2+2M+2 \\ &= 5M^2+2M+2. \end{aligned}$$

Clearly, the ratio $\frac{f(\sigma_G)}{f(\sigma_{opt})}$ tends to infinity for increasing M. □

For the 0-1 knapsack problem it is known that the unbounded approximation ratio of Greedy can be easily limited by a factor of 2 if the singleton solution consisting only of the single item with the highest profit is taken into account as a possible alternative solution. A similar remedy could work for $1|\Delta_{max} \leq Y|\sum w_j C_j$, by considering an alternative solution where L contains only the single job with largest weight. This would yield the optimal solution for the above worst-case instance. However, it is easy to see, with simple calculations, that the following extension of that instance from the above proof gives again an unbounded approximation ratio also for this upgraded version of Greedy, while the analogous extension of the worst-case instance for the knapsack problem gives a 2-approximate solution. The extended instance follows:
$p_1 = M^2, w_1 = 1$, $p_2 = 1, w_2 = 2$ and $p_3 = p_4 = M, w_3 = w_4 = 2M-1$ and $Y = 2M-2$.

2.3.2 Minimizing the number of tardy jobs subject to a maximum time disruption constraint

In this section, we study the complexity of problem $1|\Delta_{max} \leq Y|\sum U_j$ and show that is *NP*-hard in the strong sense, closing a gap in the literature. When considering the Δ_{max} criterion, the problem differs from the two problems of minimizing the

maximum lateness and the total completion time and the problem of minimizing the total weighted completion time for the presence of idle times in optimal solutions. This new complexity result helps to emphasize the different nature of these rescheduling problems. In addition to this result, we show that the problem cannot be approximated in polynomial time by a factor less than 2, unless $\mathscr{P} = NP$.

Theorem 2.3.3. *Problem* $1|\Delta_{max} \leq Y|\sum U_j$ *is strongly NP-hard.*

Proof. The proof is given by reduction from 3-PARTITION. Consider the following instance of the rescheduling problem. Let be $Y = Kt + t + y + 1$ with a positive value K such that $K \geq y$. We will introduce $6t+1$ jobs in total. First, there are $2t$ old jobs $1,\ldots,2t$ arranged in pairs. For any $i = 0,\ldots,t-1$ we set:

- $p_{2i+1} = Kt + 2t + 2 + (i+2)y,\ d_{2i+1} = p_{2i+1} + 2Yi$
- $p_{2i+2} = Kt - iy,\ d_{2i+2} = 2Y(i+1)$

The construction implies that the sum of the processing times of each pair of jobs is equal to $2Y$, i.e., $p_{2i+1} + p_{2i+2} = 2Kt + 2t + 2y + 2 = 2Y$ for $i = 0,\ldots,t-1$. Moreover, the odd-indexed longer jobs increase by y whereas the shorter even-indexed jobs decrease by y, formally $p_{2i+1} = p_{2i-1} + y$ and $p_{2i+2} = p_{2i} - y$ for $i = 1,\ldots,t-1$. Jobs are initially ordered in schedule π^* according to their indices. In this schedule each job is exactly on time as its due date coincides with its completion time.

Secondly, there are $3t$ new jobs with indices $2t+1,\ldots,5t$ corresponding to the elements of 3-PARTITION.

- $p_{2t+i} = a_i,\ d_{2t+i} = 2Yt$ for $i = 1,\ldots,3t$.

Finally, there are additional $t+1$ new jobs $5t+1,\ldots,6t+1$ with

- $p_{5t+i} = 1,\ d_{5t+i} = i$ for $i = 1,\ldots,t+1$.

We show that there is a solution to the decision version of the rescheduling problem $1|\Delta_{max} \leq Y, \sum U_j \leq t|-$, i.e. with objective function at most t, if and only if there exists a solution to 3-PARTITION.

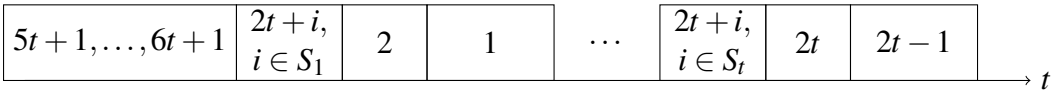


Figure 2.10 Schedule solving $1|\Delta_{max} \leq Y,\ \sum U_j \leq t|-$ when a solution to 3-PARTITION exists.

If 3-PARTITION is a "yes"-instance and allows a partition into sets S_i, the following schedule (Fig. 2.10) solves the decision problem $1|\Delta_{max} \leq Y, \sum U_j \leq t|-$. The new jobs $5t+1,\ldots,6t+1$ are scheduled in increasing order of due date, so that each completes exactly at its due date.

All new jobs $2t+1,\ldots,5t$ complete before the last two old jobs $2t-1$ and $2t$. In fact, for any $i = 1,\ldots,3t$, there is:

$$\begin{aligned} C_{2t+i} \leq t+1+ty+\sum_{\ell=0}^{t-2}(p_{2\ell+1}+p_{2\ell+2}) &= \\ t+1+ty+2Y(t-1) &= \\ t+1+ty-2(Kt+t+y+1)+2Yt &= \\ -t-1-2y-t(2K-y)+2Yt &< \quad 2Yt = d_{2t+i} \end{aligned}$$

which is true, since $K \geq y$. So, also the new jobs $2t+1,\ldots,5t$ complete on time.

The even-indexed old jobs $2i+2$ complete on time for any $i = 0,\ldots,t-1$:

$$\begin{aligned} C_{2i+2} = t+1+(i+1)y+p_{2i+2}+2Yi &= \\ t+1+(i+1)y+Kt-iy+2Yi &= \\ t+1+y+Kt+2Yi &= \\ Y+2Yi &< \quad 2Y(i+1) = d_{2i+2} \end{aligned}$$

However, the odd-indexed old jobs $2i+1$ are late for any $i = 0,\ldots,t-1$:

$$\begin{aligned} C_{2i+1} = t+1+(i+1)y+2Y(i+1) &= \\ t+1+(i+1)y+p_{2t-1}+p_{2t}+2Yi &> \quad p_{2i+1}+2Yi = d_{2i+1}, \end{aligned}$$

since $p_{2t-1} \geq p_{2i+1}$ for $i = 0,\ldots,t-1$.

So, all together there are t late jobs, i.e., $\sum U_j \leq t$. In addition, the schedule is feasible since the disruption of every old job is exactly Y. In fact, for any $i =$

$0,\ldots,t-1$, the odd-indexed old jobs $2i+1$ have a disruption

$$\begin{aligned}\Delta_{2i+1} &= t+1+(i+1)y+p_{2i+2}\\ &= t+1+(i+1)y+Kt-iy\\ &= Kt+t+y+1=Y\end{aligned}$$

and the even-indexed old jobs $2i+2$ have a disruption

$$\begin{aligned}\Delta_{2i+2} &= p_{2i+1}-(t+1+(i+1)y)\\ &= Kt+2t+2+(i+2)y-(t+1+(i+1)y)\\ &= Kt+t+y+1=Y\end{aligned}$$

Next, we show that if there is a "yes" answer to the decision problem $1|\Delta_{max}\leq Y,\sum U_j\leq t|-$, then there is also a partition giving a "yes" answer to 3-PARTITION.

If there exists a schedule for $1|\Delta_{max}\leq Y,\sum U_j\leq t|-$, at least one of the $t+1$ jobs $5t+1,\ldots,6t+1$ must be on time and therefore scheduled in the interval $[0,t+1]$. Now consider job 1: Since $p_1>t+1$, the on time dummy job must precede job 1. Therefore, job 1 must be late, because $p_1=d_1$. If the given sequence of old jobs π^* is kept, then all old jobs will be late since their due dates exactly match their completion times in π^*. So, we consider moving forward one of the old jobs. At first we try to find an old job j that can complete before job 1. The disruption constraint bounds the new completion time C_j of job j by $C_j\geq C_j(\pi^*)-Y$ and C_1 of job 1 by $C_1\leq C_1(\pi^*)+Y=p_1+Y$ and $C_j\leq C_1$. This implies that job j can precede job 1 if the following condition holds:

$$C_j(\pi^*)-Y\leq p_1+Y \tag{2.1}$$

For any odd-indexed job $j=2i+1$, $i=0,\ldots,t-1$, we can use $p_{2i+1}=p_1+iy$ and get for (2.1):

$$\begin{aligned}2Yi+(p_1+iy)-Y &\leq p_1+Y\\ (2Y+y)i &\leq 2Y\end{aligned}$$

This holds only if $i=0$, i.e., for none of the jobs $j=2i+1$, $j>1$.

For any even-indexed job $j = 2i+2$, $i = 0,\ldots,t-1$, we obtain for (2.1):

$$\begin{aligned} 2Y(i+1)-Y &\leq p_1+Y \\ 2Yi &\leq p_1 = 2Y-p_2 \end{aligned}$$

which holds only if $i = 0$, i.e., for job $j = 2$.

By repeating the same argument for trying to forward any higher indexed job before a job $2,3,\ldots,2t$, we show that we can only exchange every even-indexed old job with its direct predecessor. This creates a new sequence $2,1,4,3,\ldots,2t,2t-1$, where all even-indexed jobs $2i+2$, for $i = 0,\ldots,t-1$ are on time (the reduction of their completion times offsets the inserted dummy jobs) and the others are late.

After this pairwise interchange of old jobs, we can describe the time interval in which each such pair $(2i+2,2i+1)$ must be scheduled because of the disruption bound. The earliest starting time for an even-indexed job $2i+2$, $i = 0,\ldots,t-1$, can not be before

$$\begin{aligned} & C_{2i+2}(\pi^*)-Y-p_{2i+2} \\ = \;& 2Y(i+1)-Y-(Kt-iy) \\ = \;& 2Yi+Kt+t+y+1-Kt+iy \\ = \;& t+1+(i+1)y+2Yi. \end{aligned} \tag{2.2}$$

For an odd-indexed job $2i+1$, $i = 0,\ldots,t-1$, the completion time is bounded by

$$\begin{aligned} & C_{2i+1}(\pi^*)+Y \\ = \;& 2Y(i+1)-p_{2i+2}+Y \\ = \;& 2Y(i+1)-(Kt-iy)+Kt+t+y+1 \\ = \;& t+1+(i+1)y+2Y(i+1). \end{aligned} \tag{2.3}$$

Combining (2.2) and (2.3) we obtain a time interval

$$[t+1+(i+1)y+2Yi,\, t+1+(i+1)y+2Y(i+1)] \tag{2.4}$$

for processing jobs $2i+2$ and $2i+1$. Observing that the sum of processing times of each such pair of jobs matches exactly the width $2Y$ of its interval, it follows that the jobs must be scheduled exactly in this interval.

Summarizing, we have t pairs of old jobs, constrained to the intervals given by (2.4), and among these $2t$ jobs t are late. Thus, for a feasible schedule with $\sum U_j \leq t$, all dummy jobs $5t+1,\ldots,6t+1$ must be scheduled on time, i.e., at the beginning of the schedule in interval $[0,t+1]$ in increasing order of their due dates.

Also, all new jobs $2t+1,\ldots,5t$ corresponding to the elements of 3-PARTITION must be scheduled before the last old job to be on time, since their due date is $2Yt$. Considering any two successive intervals defined by (2.4), it is easy to see that an idle time of length y remains available between them. These gaps of the form $[t+1+(i+1)y+2Y(i+1), t+1+(i+2)y+2Y(i+1)]$ are the only possibilities for scheduling all new jobs $2t+1,\ldots,5t$ on time. Allocating all these $3t$ jobs with processing times a_i into the t gap intervals of length y gives a feasible solution and thus a "yes" answer for 3-PARTITION.

In order to conclude the proof, we show that the problem is not a number problem. The condition to show this result requires the existence of a polynomial λ of the size of a vector containing all input values of the rescheduling problem that bounds the largest number of the input, as follows:

$$\begin{aligned} \max(\max_{j\in O\cup N}(p_j,d_j),Y) &\leq \lambda(2(|O|+|N|)+1) \\ 2t(Kt+t+y+1) &\leq \lambda(2(6t+1)+1) \end{aligned}$$

Consider setting $K=y$, the condition then requires $2t(yt+t+y+1) \leq \lambda(2(6t+1)+1,3t+1)$. Since 3-PARTITION is not a number problem, there exists a polynomial λ' such that $y \leq \lambda'(t)$ which implies the existence of λ. Therefore, the statement holds and shows that the rescheduling problem is not a number problem. □

Corollary 2.3.3. *Problem* $1|no-idle,\Delta_{max} \leq Y|\sum U_j$ *is strongly NP-hard.*

Proof. Since problem $1|\Delta_{max} \leq Y|\sum U_j$ has been proven to be strongly *NP*-hard by reduction from 3-PARTITION on a set of instances with no idle time in the optimal schedule, the result follows. □

Next, we derive a non-approximability result.

Theorem 2.3.4. *Approximating problem* $1|\Delta_{max} \leq Y|\sum U_j$ *to a factor* $c<2$ *is NP-hard.*

Proof. To prove the result, we show that the c-gap problem, with $c < 2$, i.e. that to distinguish between the two cases $\sum U_j \leq 1$ and $\sum U_j \geq 2$ is a *NP*-complete problem.

We show that by reduction from PARTITION to the decision version of the rescheduling problem denoted by $1|\Delta_{max} \leq Y, \sum U_j \leq 1|-$. Consider the following instance of the decision version of the rescheduling problem:
There are two old jobs $1,2$, with

- $p_1 = 2B+5,\ d_1 = 2B+5$ and
- $p_2 = 1,\ d_2 = 2B+6,$

two new jobs $3,4$, with

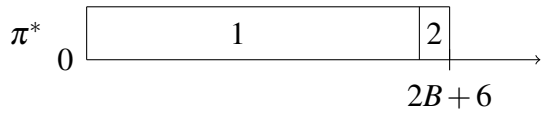
- $p_3 = p_4 = 1,\ d_3 = d_4 = 2,$

and n additional new jobs $5,6\ldots,4+n$ corresponding to the elements of PARTITION, with

- $p_{4+i} = a_i,\ d_{4+i} = 4B+8$ for $i = 1,\ldots,n.$

Finally, we set $Y = B+3$.

Schedule π^* is constructed as follows to have both old jobs on time.



If there exists a solution to PARTITION, there exist two sets S_1 and S_2 and schedule

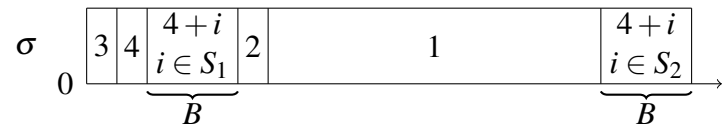


Figure 2.11 Schedule solving $1|\Delta_{max} \leq Y,\ \sum U_j \leq 1|-$ when a solution to PARTITION exists.

σ in Fig. 2.11 solves the rescheduling problem $1|\Delta_{max} \leq Y,\ \sum U_j \leq 1|-$, where job 1 is the only tardy job. Also,

$$\Delta_1 = (2B+5)+2+1+B-(2B+5) = B+3 \leq Y$$

and

$$\Delta_2 = (2B+5)+1-2-1-B = B+3 \leq Y$$

On the other side, if the rescheduling problem has a solution with $\sum U_j \leq 1$, the two jobs 3 and 4 must be scheduled at the beginning positions of the schedule, otherwise at least two jobs will become tardy. But then, 1 and 2 must be swapped and, to meet the constraint on the disruption, 2 must complete not earlier than $B+3$ and 1 not later than $3B+8$. Given that, since the due date of jobs $4+i$ for any $i = 1,\ldots,n$, equals the sum of all jobs, the solution cannot contain idle times and the only way to get that is to have a partition of jobs $4+i$, $i = 1,\ldots,n$, in two sets such that the sum of the processing times of each set is equal to B, i.e. if a solution to PARTITION exists.

This concludes the proof since, for any factor $c = 2-\varepsilon$, $0 < \varepsilon < 1$, answering to the c-gap problem, is *NP*-hard and so is approximating the problem to factor c. □

2.3.3 Minimizing the number of tardy jobs or the total tardiness subject to a total time disruption constraint

Theorem 2.3.5. *Problems* $1|\sum\Delta \leq Y|\sum U_j$ *and* $1|\sum\Delta \leq Y|\sum T_j$ *are strongly NP-hard.*

Proof. We first show that the rescheduling problem $1|\sum\Delta \leq Y|\sum U_j$ is strongly *NP*-hard, following the line of the proof of Hall and Potts (2004) for problem $1|\sum\Delta \leq Y|L_{max}$. The proof is given by reduction from 3-PARTITION. Given any instance of 3-PARTITION, we show that the problem reduces to the decision version of $1|\sum\Delta \leq Y|\sum U_j$ denoted by $1|\sum\Delta \leq Y, \sum U_j \leq 0|-$.

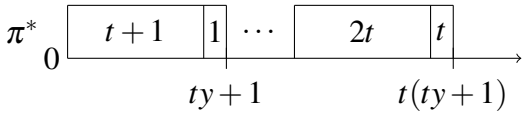
Consider the following instance of the rescheduling problems with $2t$ old jobs and $3t$ new jobs.

- $O = \{1,\ldots,2t\}$
- $N = \{2t+1,\ldots,5t\}$
- $p_j = 1,\ j = 1,\ldots,t$
- $p_j = ty,\ j = t+1,\ldots,2t$

- $d_j = t(2ty+1),\ j = 1,\ldots,2t$
- $p_j = ta_{j-2t},\ j = 2t+1,\ldots,5t$
- $d_j = t^2y+t-1,\ j = 2t+1,\ldots,5t$
- $Y = t^3y+t(t+1)/2$

It is shown below, that a solution to $1|\sum\Delta \le Y, \sum U_j \le 0|-$ exists if and only if a solution of 3-PARTITION exists.

Schedule π^* is constructed as follows.



If a solution to 3-PARTITION exists, there exist index sets $S_1,\ldots,S_t$. Then, schedule σ of Fig. 2.12 solves $1|\sum\Delta \le Y, \sum U_j \le 0|-$.

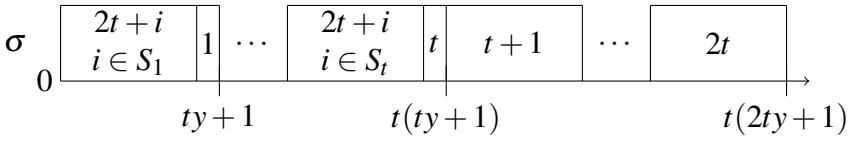


Figure 2.12 Schedule solving $1|\sum\Delta \le Y,\ \sum U_j \le 0|-$ when a solution to 3-PARTITION exists.

On the other hand, in order to have a schedule with $\sum U_j \le 0$, all jobs $t+1,\ldots,2t$ have to be scheduled after the new jobs, otherwise one of them would be late. Suppose that exactly h jobs among jobs $1,\ldots,t$ are scheduled after the first job among $t+1,\ldots,2t$ in σ^*. Each such job completes in σ^* no earlier than time $t^2y+ty+(t-h)+1$, whereas it completes in π^* no later than time $t(ty+1)$. Moreover, jobs $t+1,\ldots,2t$ are completed at times $ty,(2ty+1),\ldots,(t^2y+t-1)$ in π^* and are preceded by all new jobs and $t-h$ jobs of $\{1,\ldots,t\}$ and therefore are completed no earlier than times $(t^2y+(t-h)+ty),(t^2y+(t-h)+2ty),\ldots,(t^2y+(t-h)+t^2y)$ in σ^*. Computing the total disruption, we obtain:

$$\begin{aligned}\sum\Delta &\ge t^3y+t(t-h)-t(t-1)/2+h(ty-h+1)\\ &= Y+h(ty-h-t+1)\end{aligned}$$

Since $y \geq 3$, by definition of 3-PARTITION, we conclude that $h = 0$. Also, in order to have the disruption constraint satisfied, jobs $1,\ldots,t$ must be scheduled on time with respect to their initial completion times and the new jobs in sets $S_1,\ldots,S_t$ between them as shown in the figure above. The partition of the new jobs solves 3-PARTITION.

The same reduction also works for problem $1|\sum\Delta \leq Y|\sum T_j$ and shows that 3-PARTITION reduces to the decision problem $1|\sum\Delta \leq Y, \sum T_j \leq 0|-$.

In order to conclude the proof, we show that the two problems are not a number problem. The condition to prove this result requires the existence of a polynomial λ of the size of the rescheduling problem that bounds the largest number of the input, as follows:

$$\begin{aligned} \max(\max_{j\in O\cup N}(p_j,d_j),Y) &\leq \lambda(2(|O|+|N|)+1) \\ 2t^3y+t^2+t &\leq \lambda(10t+1) \end{aligned}$$

Since 3-PARTITION is not a number problem, there exists a polynomial λ' such that $y \leq \lambda'(3t)$ which implies the existence of λ. Therefore, the statement holds and shows that the rescheduling problem is not a number problem. □

Corollary 2.3.4. *Problems* $1|no-idle,\sum\Delta \leq Y|\sum U_j$ *and* $1|no-idle,\sum\Delta \leq Y|\sum T_j$ *are strongly NP-hard.*

Proof. Since both problems $1|\sum\Delta \leq Y|\sum U_j$ and $1|\sum\Delta \leq Y|\sum T_j$ were proved to be strongly *NP*-hard by reduction from 3-PARTITION, even if there is no idle time in the optimal schedule, problems $1|no-idle,\sum\Delta \leq Y|\sum U_j$ and $1|no-idle,\sum\Delta \leq Y|\sum T_j$ are strongly *NP*-hard too. □

2.3.4 Other complexity results

First, we derive a corollary to Theorem 3 in Hall and Potts (2004), which derives from the fact that the reduction from 3-PARTITION used in the proof does not use idle times.

Corollary 2.3.5. *Problem* $1|no-idle,\sum\Delta \leq Y|\sum L_{max}$ *is strongly NP-hard.*

Next, we turn to the problem of minimizing total tardiness subject to a maximum disruption constraint. The result is a corollary to the weak *NP*-hardness proof given by Du and Leung (1990).

Corollary 2.3.6. *Problem* $1|\Delta_{max} \leq Y|\sum T_j$ *is NP-hard.*

Proof. Consider the well-known generalization of the problem, i.e. the scheduling problem $1||\sum T_j$, which is known to be weakly *NP*-hard. It is enough to take $Y = +\infty$ to show that $1||\sum T_j$ is a particular case of $1|\Delta_{max} \leq Y|\sum T_j$. Thus, the rescheduling problem is *NP*-hard. □

Next, we derive a corollary to Theorem 2.3.1.

Corollary 2.3.7. *Problem* $1|\sum \Delta \leq Y|\sum w_j C_j$ *is NP-hard.*

Proof. Consider the special case with one old job j. Clearly, with one old job we have $\sum \Delta = \Delta_j = \Delta_{max}$. Since problem $1|\Delta_{max} \leq Y|\sum w_j C_j$ has been proven to be *NP*-hard in the weak sense in Section 2.3.1, even when there is a single old job and when there are no inserted idle times, problems $1|\sum \Delta \leq Y|\sum w_j C_j$ is *NP*-hard too. □

Again, this last complexity result originates from a polynomial reduction that uses a schedule with no idle times. Hence, the following corollary holds.

Corollary 2.3.8. *Problem* $1|no-idle, \sum \Delta \leq Y|\sum w_j C_j$ *is NP-hard.*

2.3.5 Overview on complexity results of rescheduling problems

Given the new results obtained that were presented in this section, we give here an overview of the main known computational complexity results of rescheduling problems for new orders.

Table 2.2 summarizes the current state of known complexity results. Each row contains the results for each different objective function. For each row, the results for the Δ_{max} criterion are shown first, and for the $\sum \Delta$ criterion next. All results hold for both versions with and without the non-delay schedule constraint.

The first two rows contain the results from the literature (Hall and Potts (2004)), while remaining results highlighted in bold are given in this thesis.

Table 2.2 Computational complexity of rescheduling problems.

f	Δ_{max}	$\sum\Delta$
L_{max}	$O(n+n_N log(n_N))$	strongly NP-hard
$\sum C_j$	$O(n+n_N log(n_N))$	NP-hard
$\sum w_j C_j$	**NP-hard**	**NP-hard**
$\sum U_j$	**strongly NP-hard**	**strongly NP-hard**
$\sum T_j$	**NP-hard**	**strongly NP-hard**

The new complexity results obtained for several problems that were left open in the previous literature give a better understanding of the intrinsic complexity of most of rescheduling problems. The table also emphasizes the higher complexity of rescheduling problems with the $\sum\Delta$ criterion, which is not surprising since the Δ_{max} constraint is a kind of local constraint, i.e. it constrains the completion time of each old job independently, while $\sum\Delta$ constrains the time deviation computed over all jobs.

Despite the current characterization of rescheduling problems in terms of their complexity, it is important to emphasize that there are two open problems. The two problems are $1|\Delta_{max} \leq Y|\sum T_j$ and $1|\sum\Delta \leq Y|\sum w_j C_j$, for which we set weak *NP*-hardness but for which no pseudo-polynomial algorithm is known. The structure of problem $1|\Delta_{max} \leq Y|\sum T_j$ has many similarities with problem $1|\Delta_{max} \leq Y|\sum U_j$ for minimizing the total number of late jobs subject to a maximum disruption constraint. These similarities should be investigated to see if the problem is also strongly *NP*-hard. On the contrary, there is no similar result that suggests whether problem $1|\sum\Delta \leq Y|\sum w_j C_j$ can be solved in pseudo-polynomial time or not. However, the problem does not answer to ordering properties for old jobs and it may lead to unforced idleness on the machine, and these two facts make hard to identify valid structures to help solve the problem. Thus, further analysis should prove whether there is such a pseudo-polynomial algorithm or whether the problem is *NP*-hard in the strong sense.

Chapter 3

Timing problems given a fixed sequence of jobs

Timing problems require the determination of suitable task execution dates within a given processing sequence, such that constraints and objectives are met. Their efficient solution is critical in branch-and-bound and neighbourhood search methods for vehicle routing, project and machine scheduling, and various other applications. An overview of the applications, properties and methods of timing problems, derived from various combinatorial optimization problems, is given in Vidal et al. (2015). In scheduling, the best-known timing problem is the problem of minimizing total earliness and tardiness of a given sequence (see for instance Garey et al. (1988), Davis and Kanet (1993) and Croce and Trubian (2002)). In scheduling, especially in manufacturing scheduling, this is one of the few scheduling problems, apart from its variants, that requires solving a timing problem. In fact, scheduling often considers regular objective functions that are non-decreasing in terms of job completion times, because as completion times increase, costs and delivery times usually increase as well, having a negative impact on the overall system. However, with the growing interest in just-in-time policies, there has been an increasing interest in objective functions in the form of (weighted) *absolute* deviations from given dates. In this context, orders are penalized if they are both early and late. Several papers introduced and revised existing work on single machine scheduling problems with earliness and tardiness penalties (Sidney (1977), Bagchi et al. (1986), Garey et al. (1988), Baker and Scudder (1990)).

The timing problem in rescheduling arises because of the disruption constraint, which introduces a measure of the absolute time deviations of old jobs, where the target dates from which the deviation is computed are the original completion times of the jobs. This chapter is devoted to understanding the complexity of the timing problems in rescheduling problems for new orders studied in this dissertation, when the job sequence is fixed, and to providing efficient algorithms.

3.1 Rescheduling with a fixed job sequence

In Section 2.2, two classes of problems were introduced, one where there are always optimal solutions without idle times and the other where optimal solutions may require the insertion of idle times. In this section, we study the timing problem, that is underlying the second class of problems, i.e. the problem of determining the completion times of jobs when the sequence is given and fixed (see Figure 3.1). Recall that so far we have considered several rescheduling problems with a scheduling objective, which is given by a regular objective function, and a constraint on the maximum or total absolute time deviation of the old jobs.

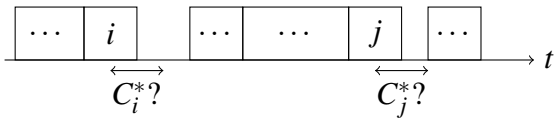


Figure 3.1 The timing problem

Since the input of the timing problem is a sequence of jobs, we assume to start with a compact solution, i.e. one without idle times between jobs. Since any regular objective function is minimized in a schedule without inserted idle times, in this chapter we assume that this schedule is infeasible with respect to the disruption constraint. Otherwise, the timing in this schedule is already solved to optimality. Since the starting solution is a compact one, we consider only *right*-shifting, i.e. delaying in time, jobs or blocks of jobs. Since we have to evaluate a regular objective function f and a non-regular disruption function among Δ_{max} and $\sum\Delta$, this clearly leads to evaluating the increase of f and the change in the disruption, which may be both an increase or a decrease.

In this chapter, we first show that the timing problems can be formulated as Linear Programs, and then develop dedicated algorithms, that set their polynomial

Table 3.1 Notation for timing problems.

Notation	Description
α	schedule of jobs scheduled with no idle times
ω	value of the objective function in α
d_j^C	original completion time $C_j(\pi^*)$ of old job j
B	any block of jobs scheduled consecutively without interruptions
A_j	a tail block starting with job j
C_B	completion time of block B
t_B	starting time of block B
E_B	set of jobs in B such that $C_j < d_j^C$
T_B	set of jobs in B such that $C_j > d_j^C$
O_B	set of jobs in B such that $C_j = d_j^C$
δ_j^Δ	decrease of $\sum\Delta$ if tail block A_j is right shifted by one time unit

complexity. The algorithms will use the trade-off of limiting the increment of the objective function and decreasing the disruption, which is necessary to obtain a feasible solution.

In the chapter, we introduce and explain some additional notation. For easier reference, we give a summary of main notation that will be introduced in Table 3.1.

3.2 Linear programming model

We present a linear program (LP) for the set of problems tackled in this chapter to show that they fall into the category of problems that can be formulated as Linear Programs.

We are given a fixed sequence of old and new jobs $j = 1, \ldots, n$ and, for each of them, we introduce the continuous variables $C_j \geq 0$ that represent the completion times of jobs. Then, for any job $j \in O$, we introduce variables $\Delta_j \geq 0$ as the disruption of each old job.

We express the *LP* formulation as follows:

Minimize $f(C_j)$

subject to

$$C_1 \geq p_1 \qquad (1)$$

$$C_j \geq C_{j-1} + p_j \quad j = 2,\ldots,n \qquad (2)$$

$$\Delta_j \geq C_j - d_j^C \qquad \forall j \in O \qquad (3)$$

$$\Delta_j \geq d_j^C - C_j \qquad \forall j \in O \qquad (4)$$

In addition, depending on whether we consider the Δ_{max} or the $\sum\Delta$ criterion, we have, respectively:

$$\Delta_j \leq Y \qquad \forall j \in O \quad (5a)$$

or

$$\sum_{j\in O}\Delta_j \leq Y \qquad (5b)$$

The objective function $f(C_j)$ can be any regular objective function considered so far. Constraints (1) and (2) define the completion times of jobs, such that there is no overlap between jobs. Constraints (3) and (4) define the disruption of each old job as the absolute time deviation from the initial completion times. Finally, constraints (5a) and (5b) define the disruption constraint.

3.3 Timing algorithms

From Table 2.1 and Table 2.2 we can see that rescheduling problems that belong to the family of problems that need idle time insertion are all *NP*-hard. In this section, we prove that the special case with fixed jobs sequence is polynomially solvable. The problem with fixed sequence becomes a timing problem, i.e. the problem of determining the exact starting times of the jobs. So, as soon as a sequence has to be evaluated, the timing problem must be solved. Beside the interest in the field of timing problems, these algorithms may be used for solution procedures such as in branch-and-bound algorithms, where to each node is associated a specific (partial) sequence of jobs.

Assume we are given a fixed sequence of old and new jobs $\alpha = (1,2,\ldots,n)$ that are initially scheduled with no inserted idle times. We also assume continuous completion times since it is a standard assumption when considering timing problems (see Chrétienne and Sourd (2003), Vidal et al. (2015)). In the following, we consider that jobs are indexed following their order in the sequence.

In the remaining part of the section, we first address timing problems with the maximum disruption constraint Δ_{max} before turning to the case of the total disruption $\sum\Delta_j$. Since timing problems usually determine optimal starting times of jobs in terms of time deviation from due dates, from now on we will call the original completion time of old jobs as d_j^C. So, we have $d_j^C = C_j(\pi^*)$.

3.3.1 Timing with the Δ_{max} criterion

This set of problems is identical to the set of problems of minimizing a regular objective function subject to time windows constraints. Given the value of the maximum disruption Y, for each old job we can define a release dates and deadlines as follows:

$$r_j = \begin{cases} \max(0, d_j^C - Y), & j \in O \\ 0, & j \in N \end{cases} \tag{3.1}$$

$$\tilde{d}_j = \begin{cases} d_j^C + Y, & j \in O \\ +\infty, & j \in N \end{cases} \tag{3.2}$$

Clearly, processing a job outside the interval $[r_j, \tilde{d}_j]$ makes the schedule infeasible with respect to the disruption constraint. The problem at hand is solved by the *minimum idle time policy* (Vidal et al. (2015)) as follows. Each job of the sequence is scheduled at its earliest feasible execution date, i.e. when the previous job in the sequence completes or at its release date. If all jobs can be successfully scheduled in this way, from the first to the last, the schedule is feasible and the increase in the objective function is minimum. The timing algorithm for scheduling with time windows (Algorithm TIM_TW) is shown in Algorithm 2 and runs in $O(n)$ time.

Algorithm 2 Algorithm TIM_TW for $1|seq,\Delta_{max} \leq Y|f$

```
Compute r_j (resp. d̃_j) according to equations 3.1 (resp. 3.2)
C_1 = r_j + p_1
for i = 1,...,n-1 do
    C_{i+1} = min(r_j, C_i) + p_{i+1}
    if C_{i+1} > d̃_j then
        return infeasibility.
    end if
end for
```

3.3.2 Timing with the $\sum\Delta$ criterion

The proposed timing algorithms are developed in two steps. First, a general pre-processing step provides an optimal timing for the schedule subject to the constraint that the initial objective function value must not increase with respect to the initial sequence α. After the pre-processing, the algorithms are developed separately and one is proposed for objective functions $\sum w_jC_j, \sum U_j, \sum T_j$ and one for L_{max}.

Before introducing the pre-processing algorithm, let us introduce some notations, that will be useful to present the two algorithms. At any point of the schedule, let a block be a group of jobs that are scheduled consecutively without interruptions and preceded and followed by idle times, with the exception of a block starting at time 0: in this case, it is only followed by an idle time. A subblock is any job sequence of consecutive jobs which is part of a block. A subblock is a *head* if it is preceded by an idle time or it is starting at time 0. A subblock is a *tail* if it is followed by idle times. A schedule σ is constituted by a set $\mathscr{B}$ of blocks scheduled consecutively and separated by idle times. Each block $B \in \mathscr{B}$ has a starting time t_B and a completion time C_B. Figure 3.2 shows an example of a block of jobs constituted by a head subblock H with starting time t_H and completion time C_H and a tail subblock T with starting time t_T and completion time C_T.

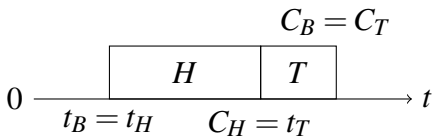


Figure 3.2 A block B of jobs, made up of a head H and a tail T.

The pre-processing phase

The algorithm takes as input a compact sequence of jobs scheduled without idle times. If the schedule satisfies the constraint on the disruption, then the timing of jobs is already optimal since any regular objective function is also minimized, thus we consider only the case where $\sum \Delta_j > Y$.

The pre-processing answers to the question if it is possible to get a feasible schedule without any increase of the current objective function. Given the initial value $\omega = f(\alpha)$ of the objective function in the compact schedule α, the new schedule σ needs to satisfy the constraint $f(\sigma) \leq \omega$. For this purpose, we assign deadlines to jobs, such that a violation of any of them implies an increase of f.

First, we focus on the case of function L_{max}. Given its value ω in the compact schedule, imposing $f_{max} \leq \omega \iff f_j \leq \omega, \forall j \in O \cup N$. To have this constraint guaranteed, we set $\tilde{d}_j$ such that $\tilde{d}_j \leq d_j + \omega$. There are cases, where some job i has a more restrictive deadline than some job j that precedes i in the schedule. In this case, i will first meet its deadline if both are right shifted of the same amount, and j will never meet its own deadline if i is scheduled at a feasible time. Consequently, we set:

$$\tilde{d}_j = \begin{cases} d_j + \omega & j = n \\ \min(\tilde{d}_{j+1} - p_{j+1}, d_j + \omega) & j = n-1, \ldots, 1 \end{cases}$$

Now, let us consider the case of sum objective functions. The initial compact schedule α has an objective of ω, which is given by $\omega = \sum_{j \in O \cup N} f_j(\alpha)$. Set $\omega_j = f_j(\alpha), \forall j \in O \cup N$ as the value of f_j in α. Since the sequence is fixed and the f_j's are regular functions, in any schedule σ with jobs in the same sequence as in α it holds $f_j(\sigma) \geq \omega_j$. But then, imposing $\sum f_j = \sum_{j \in O \cup N} f_j \leq \omega \implies f_j \leq \omega_j, \forall j \in O \cup N$.

We illustrate the above definitions on the objective functions we are considering in this paper. When $f = \sum U_j$,

$$\tilde{d}_j = \begin{cases} d_j & \omega_j = 0, j = n \\ +\infty & \omega_j = 1, j = n \\ \min(\tilde{d}_{j+1} - p_{j+1}, d_j) & \omega_j = 0, j = n-1, \ldots, 1 \\ \min(\tilde{d}_{j+1} - p_{j+1}, +\infty) & \omega_j = 1, j = n-1, \ldots, 1 \end{cases}$$

When $f = \sum T_j$,

$$\tilde{d}_j = \begin{cases} d_j + \omega_j & j = n \\ \min(\tilde{d}_{j+1} - p_{j+1}, d_j + \omega_j) & j = n-1, \ldots, 1 \end{cases}$$

Finally, notice that when we are considering $\sum f_j = \sum w_j C_j$, $\tilde{d}_j = \omega_j$, i.e. there is no way to shift jobs without increasing the objective function and the pre-processing will not modify the initial compact schedule.

The pre-processing algorithm minimizes $\sum \Delta_j$, given the limit on f and checks if $\sum \Delta_j \leq Y$. The idea follows the one of the algorithm for the earliness/tardiness scheduling problem with a fixed sequence by Garey et al. (1988) that runs in $O(n \log n)$ time. Minimizing the sum of the earliness and tardiness over a job set J w.r.t due dates d_j means, by definition, minimizing $\sum_{j \in J} |C_j - d_j|$, which is equal to minimizing the total disruption w.r.t. the original completion times. So, we modify the algorithm of Garey et al. (1988) and use the modified version to minimize the total disruption subject to job deadlines constraints. We illustrate the algorithm in the following paragraphs.

First, let us give some definitions. We say that a job $j \in O$ in the rescheduling problem is *early*, *on time* or *late* if it completes before, exactly at or after d_j^C. Notice that the new jobs have no due date d_j^C as the total disruption is only computed on O. For a given block B in a schedule σ, let us denote by $E_B(\sigma)$ the set of early old jobs, by $O_B(\sigma)$ the set of on time old jobs and by $T_B(\sigma)$ the set of tardy old jobs (E_B, O_B, T_B when there is no ambiguity).

The algorithm for the earliness/tardiness problem, iteratively adds a job following the order of the fixed sequence, appending it to the last block or creating a new block. This job is either inserted at its due date, if this does not cause an overlapping of jobs, or exactly next to the previously added job. Let B be the block to which job j belongs to after the insertion: if $|E_B| + |O_B| - |T_B| = 0$, then the block is shifted to the left until a job of the block becomes on time or the previous block is met or $t_B = 0$.

To adapt this algorithm to our problem, we distinguish the steps done for the set of new jobs and those for the set of old jobs. Considering new jobs, since they do not have any impact on the total disruption, we can schedule them immediately after

Algorithm 3 Algorithm TIM_C for $1|seq, f \leq \Omega|\sum\Delta_j$

```
Compute jobs deadlines following equations (1), (2), (3), (4)
C_1 = min(max(p_1, d_1^C), d̃_1)
B := 1
for i = 1, ..., n-1 do
    if i+1 ∈ O then
        if C_i + p_{i+1} ≤ min(d_{i+1}^C, d̃_{i+1}) then
            C_{i+1} = min(d_{i+1}^C, d̃_{i+1})
            B := B + 1
        else
            C_{i+1} = C_i + p_{i+1}
            if |E_B| + |O_B| - |T_B| = 0 then
                left-shift B by min(min_{j∈O∩B, d_j^C < C_j}(C_j - d_j^C), t_B - C_{B-1}, t_B)
            end if
        end if
    else
        C_{i+1} = C_i + p_{i+1}
    end if
end for
```

their predecessors or at $t = 0$ if there is no job scheduled before, i.e. for any job $i \in N$ we define its completion time as $C_i = C_{i-1} + p_i$ with $C_0 = 0$. If we consider old jobs, we schedule them so that they complete by d_j^C, or at $\tilde{d}_j$ if $\tilde{d}_j < d_j^C$. If scheduling a job by d_j^C this causes an overlapping of jobs, then the current job is scheduled exactly next to the previously added job. Notice that for any job that is appended to its predecessor, the deadlines are satisfied by definition. The modified timing algorithm for minimizing the total disruption subject to a constraint on the objective function (Algorithm TIM_C) is shown in Algorithm 3.

Remark 3.3.1. *Notice that in Algorithm 3, by construction, for any head subblock H that is not starting at time 0 (see Figure 3.2), we have $|E_H| + |O_H| - |T_H| > 0$. Also, for any tail subblock T (see Figure 3.2), where none of the jobs has $C_j = \bar{d}_j$, we have $|E_T| - |O_T| - |T_T| \leq 0$, since otherwise, subblock T would not have been shifted so far to the left.*

Theorem 3.3.1. *Algorithm 3 returns an optimal solution for the problem $1|seq, f \leq \Omega|\sum\Delta_j$.*

Consider schedule σ returned by Algorithm 3. Let us introduce, for each tail subblock A_j where j is the starting job of the subblock, a value

$$\delta_j^{\Delta} = \begin{cases} |E_{A_j}| - |O_{A_j}| - |T_{A_j}| & \forall j \in O \\ 0 & \forall j \in N \end{cases}$$

The value δ_j^{Δ} is the change in the total disruption if A_j is right shifted by one time unit and since only the old jobs contribute to to it, we assign a value $\delta_j^{\Delta} = 0$ to each A_j made up of only new jobs. If $\delta_j^{\Delta} > 0$, there is a gain in the total disruption, i.e. the total disruption decreases, and vice versa.

Recall that the jobs are indexed by position in the sequence. For any block B, we call j_B^* the job in B, such that $\delta_{j_B^*}^{\Delta} > 0$, $\delta_{j_B^*}^{\Delta} \geq \delta_i^{\Delta}$ and $j_B^* > i \; \forall i \in B$. Notice, that by definition, j_B^* is always an old job or does not exist.

Lemma 3.3.1. *The head subblocks of any block B, that do not contain j_B^*, are optimally scheduled.*

Lemma 3.3.2. *Shifting any of the tail subblocks with $\delta_j^{\Delta} > 0$ always increases f.*

In the next two sections, we establish how to iteratively identify the optimal set of subblocks to be shifted to determine the optimal timing, first for problem $1|seq, \sum \Delta_j \leq Y|L_{max}$, next for problem $1|seq, \sum \Delta_j \leq Y|\{\sum w_j C_j, \sum U_j, \sum T_j\}$.

Timing for problem $1|seq, \sum \Delta_j \leq Y|L_{max}$

We saw, that right shifting any of the tail subblocks with corresponding $\delta_j^{\Delta} > 0$, always leads to an increase of any objective function f (Lemma 3.3.2) meaning that L_{max} exactly increases by the amount of the shift. Considering this, it is more convenient to right shift simultaneously the subblocks $A_{j_B^*}$ that corresponds to $\delta_{j_B^*}^{\Delta}$. By definition, j_B^* has the maximum δ_j^{Δ} value in block B, i.e. the unitary gain in the total disruption is maximum. Therefore, for the same increase of the L_{max}, shifting all tail subblocks with the maximum δ_j^{Δ} value inside each block indeed leads to the maximum decrease of the total disruption. For any block B, let us denote by $\mathscr{B}^E$ the set of all these tail subblocks of the schedule that start with a job j_B^*. Consider only tail subblocks A_j with $\delta_j^{\Delta} > 0$.

Now, consider shifting simultaneously all the tail subblocks in $\mathscr{B}^E$. The situation is depicted in Figure 3.3. The white blocks are the tail subblocks that start with the respective j_B^* and therefore belong to set $\mathscr{B}^E$. The dotted blocks are the head subblocks that do not move.

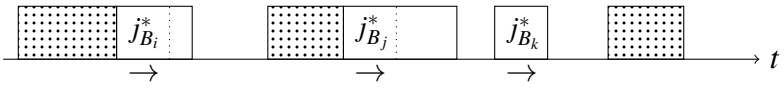


Figure 3.3 Set $\mathscr{B}^E$ of tail subblocks shifted at each iteration of Algorithm 4.

The total unit gain $\bar{\delta}^{\Delta}$ in the total disruption per unitary time shift is given by $\bar{\delta}^{\Delta} = \sum_{B \in \mathscr{B}^E} \delta^{\Delta}_{j^*_B}$. We can derive the following corollary of Lemma 3.3.1 and 3.3.2 for the $1|seq, \sum \Delta_j \leq Y|L_{max}$ problem.

Corollary 3.3.1. *In any schedule σ, it is necessary and sufficient to shift all the blocks in set $\mathscr{B}^E$ of schedule σ to find the schedule with the minimum increase of L_{max} and the greatest decrease of $\sum \Delta_j$ for each unit of inserted idle time.*

Algorithm 4 Algorithm TIM_M for $1|seq, \sum \Delta_j \leq Y|L_{max}$

```
1: Run Algorithm 3 on the given job sequence and store solution in σ
2: τ = ∑Δ(σ) − Y
3: if τ ≤ 0 then
4:     return σ
5: end if
6: Compute 𝔅^E, δ̄^Δ, q_B, q_j
7: while δ̄^Δ > 0 do
8:     δ = min(τ/δ̄^Δ, q_B, q_j)
9:     Shift all blocks in σ belonging to 𝔅^E by δ
10:    τ := τ − δ · δ̄^Δ
11:    if τ ≤ 0 then
12:        return σ
13:    end if
14:    Compute 𝔅^E, δ̄^Δ, q_B, q_j
15: end while
16: return The given job sequence is infeasible
```

The algorithm iteratively shifts all the tail subblocks in $\mathscr{B}^E$. When moving these subblocks several events may happen:

(E_1). One block meets another block.

(E_2). At least one old job reaches its original completion time d^C_j.

Consider event E_1. The event occurs when at least a subblock $B \in \mathscr{B}^E$ is followed by a block $B+1 \in \mathscr{B} \setminus \mathscr{B}^E$. Then, E_1 occurs after $q_B = \min_{B \in \mathscr{B}^E, (B+1) \in \mathscr{B} \setminus \mathscr{B}^E} (t_{B+1} - C_B)$ units of time. Next, consider event E_2. Clearly, the first event of this type occurs after $q_j = \min_{j \in O \cap \mathscr{B}^E, d^C_j > C_j} (d^C_j - C_j)$ units of time.

Whenever one of the mentioned events occur, to restore the optimality condition of Corollary 3.3.1, the δ_j^Δ values must be recomputed to find the new optimal set of subblocks $\mathscr{B}^E$ to be shifted.

There are two stopping conditions for the algorithm. First, the algorithm stops as soon as feasibility is reached, i.e. if at any iteration we can shift by $\frac{\tau}{\bar{\delta}^\Delta}$ units of time before any of the events occur. Secondly, the algorithm stops as soon as there are no strictly early tail subblocks. In this case, all $\delta_j^\Delta \leq 0$ and there is no way to reach feasibility with the given sequence of jobs.

Putting all together, we obtain the timing algorithm for the problem with $f = L_{max}$ (Algorithm TIM_M) depicted in Algorithm 4. The algorithm takes as input the schedule computed by Algorithm 3. Then, at each iteration the algorithm computes set $\mathscr{B}^E$, the corresponding $\bar{\delta}^\Delta$ and when the next event occurs (quantities q_j, q_B). Next, all the subblocks in $\mathscr{B}^E$ are right shifted. If at any time, feasibility is reached or there are no subblocks in $\mathscr{B}^E$, i.e. $\bar{\delta}^\Delta \leq 0$, then the algorithm stops.

Lemma 3.3.3. *Algorithm* TIM_M *finds an optimal solution for* $1|seq, \sum \Delta_j \leq Y|f_{max}$, *with* $f_{max} = L_{max}$, *in* $O(n^2)$ *time.*

Proof. Computing $\mathscr{B}^E$, $\bar{\delta}^\Delta$, q_B and $q_j, \forall j \in O \cup N$ can be done in $O(n)$ time by computing the δ_j^Δ's backward and meanwhile keeping track of $\bar{\delta}^\Delta$, q_B and q_j and the subblocks in $\mathscr{B}^E$. Starting from the last job of each block, we assign $\delta_j^\Delta = 1$ if $q_j > 0$, $\delta_j^\Delta = -1$ if $q_j \leq 0$ or $\delta_j^\Delta = 0$ if j is a new job. Then, for each further job in the same block, $\delta_j^\Delta = \delta_{j+1}^\Delta + 1$ if $q_j > 0$, $\delta_j^\Delta = \delta_{j+1}^\Delta - 1$ if $q_j \leq 0$ or $\delta_j^\Delta = \delta_{j+1}^\Delta$ if j is a new job.

Next, we bound the number of events, i.e. the number of iterations of the algorithm by an additional factor $O(n)$. Clearly, the number of E_2 events is bounded by n, since once an old job goes on time, it cannot become early again. To bound the number of E_1 events, consider instead what happens after one occurs. Two subblocks $B \in \mathscr{B}^{\mathscr{E}}$ and $(B+1) \in \mathscr{B} \setminus \mathscr{B}^{\mathscr{E}}$ merge and possibly continue to shift if there is some tail subblocks $B_j \subset B$ with $\delta_j^\Delta > 0$. However, $(B+1)$ will not be split again, since, by definition, $E_{B+1} - O_{B+1} + T_{B+1} \leq 0$ (from Proof of Lemma 3.3.1) and right-shifting $(B+1)$ alone does never decrease the total disruption. So, the number of merging operations, i.e. the number of E_1 events, is bounded by the number of blocks, which is at most n.

The time complexity of Algorithm 4 is then bounded by $O(n^2)$ time. □

Timing for problems $1|seq, \sum \Delta \leq Y|\{\sum w_j C_j, \sum U_j, \sum T_j\}$

We now set the procedure to identify and shift subblocks for finding an optimal timing for objectives $\sum f_j \in \{\sum w_j C_j, \sum U_j, \sum T_j\}$. Let us introduce, for any tail subblock A_j, a value δ_j^f as the increase of $\sum f_j$ when right shifting A_j by one time unit. Let us define the tail subblock A_j^*, with maximum $\frac{\delta_j^\Delta}{\delta_j^f}$. Let $\delta_j^{\Delta *}$ be the unitary decrease of the total disruption when right-shifting A_j^*. We derive the following corollary of Lemma 3.3.1 and 3.3.2 for any of the $1|seq, \sum \Delta_j \leq Y|\sum f_j$ problems.

Corollary 3.3.2. *In any schedule σ, it is necessary and sufficient to right shift the tail subblock A_j^* to find the schedule with the minimum increase of $\sum f_j = \{\sum w_j C_j, \sum U_j, \sum T_j\}$ and the greatest decrease of $\sum \Delta_j$ for one unit of inserted idle time.*

It appears that the main difference with problem $1|seq, \sum \Delta_j \leq Y|L_{max}$ is given by the fact that one tail subblock at a time is shifted, instead of moving multiple tail subblocks simultaneously. The timing algorithm for sum objective functions $1|seq, \sum \Delta_j \leq Y|\sum f_j$, referred to as Algorithm TIM_S, shifts the tail subblock A_j^* until there is a change in the δ_j^Δ's or the δ_j^f's. In addition to events (E_1) and (E_2), the following event may happen:

(E_3). At least one job reaches its due date d_j.

The additional event E_3 may occur, when function $\sum f_j$ depends on due dates, i.e. for $\sum U_j$ and $\sum T_j$, and causes a change in the contribution δ_j^f to the objective function for a unitary shift, when moving a subblock. The first event of this type occurs after $q_D = \min_{j \in O \cup N, d_j > C_j}(d_j - C_j)$ units of time.

Beside this, the algorithm works as the previous algorithm for the L_{max} case. TIM_S is sketched in Algorithm 5.

Lemma 3.3.4. *Algorithm* TIM_S *finds an optimal solution for any problem* $1|seq, \sum \Delta_j \leq Y|\sum f_j$, *with* $\sum f_j \in \{\sum w_j C_j, \sum U_j, \sum T_j\}$, *in* $O(n^2)$ *time.*

Proof. In each iteration, the algorithm identifies A_j^* and computes $\delta_j^{\Delta *}$, q_B, q_j, q_D. All can be computed in $O(n)$ time.

Next, we bound the number of events. We have already shown that the number of E_2 events is bounded by n (proof of Lemma 3.3.3). In the same way, the number of E_3 events is bounded by n, since this type of event is generated by a job meeting

Algorithm 5 Algorithm TIM_S for $1|seq, \sum \Delta_j \leq Y| \sum f_j$

```
Run Algorithm 3 on the given job sequence and store solution in σ
τ = ∑Δ(σ) − Y
if τ ≤ 0 then
    return σ
end if
Compute A*_j, δ^{Δ*}_j, q_B, q_j, q_D
while δ^{Δ*}_j > 0 do
    δ = min(τ/δ^{Δ*}_j, q_B, q_j, q_D)
    shift A*_j by δ
    τ := τ − δ · δ^{Δ*}_j
    if τ ≤ 0 then
        return σ
    end if
    Compute A*_j, δ^{Δ*}_j, q_B, q_j, q_D
end while
return The given job sequence is infeasible
```

its due date and there are n jobs. To bound the number of E_1 events, consider instead what happens after one occurs. We consider an E_1 event, where no event E_2 or E_3 occur. Two subblocks A_i and A_j merge and since there are no events E_2 or E_3 happening at the same time, the value $\frac{\delta_i^\Delta}{\delta_i^f}$ in subblock A_i does not change if we neglect the merged block A_j. Also, if A_i is selected as a candidate to be right shifted, $\frac{\delta_i^\Delta}{\delta_i^f} \geq \frac{\delta_j^\Delta}{\delta_j^f}$ value. Then, when we recompute this value for the merged block $A_i \cup A_j$, we have $\frac{\delta_i^\Delta + \delta_j^\Delta}{\delta_i^f + \delta_j^f}$. The merged block is the new optimal block to be right shifted, since

$$\begin{aligned}
\frac{\delta_i^\Delta + \delta_j^\Delta}{\delta_i^f + \delta_j^f} - \frac{\delta_i^\Delta}{\delta_i^f} &= \\
\frac{\delta_i^f \delta_i^\Delta + \delta_i^f \delta_j^\Delta - \delta_i^f \delta_i^\Delta - \delta_j^f \delta_i^\Delta}{\delta_i^f (\delta_i^f + \delta_j^f))} &= \\
\frac{\delta_i^f \delta_j^\Delta - \delta_j^f \delta_i^\Delta}{\delta_i^f (\delta_i^f + \delta_j^f))} &\geq 0
\end{aligned}$$

and $\frac{\delta_i^\Delta}{\delta_i^f} \geq \frac{\delta_j^\Delta}{\delta_j^f}$ by definition and the denominator is always positive since we consider regular objective functions. Then, once two blocks merge, they do not split again, unless another type of event is involved. The total number of E_1, E_2 and E_3 events is hence bounded by $3n$.

Given the maximum total number of iterations, i.e. the total number of events, and the number of operations for updating the values to identify the optimal idle time insertion, the overall time complexity of the algorithm is in $O(n^2)$ time. □

Chapter 4

Exact and approximation algorithms for $1|\Delta_{max} \leq Y|\sum w_j C_j$

In this chapter we consider the sum of weighted completion times subject to a maximum time disruption constraint. This is a boundary problem, when dealing with the Δ_{max} criterion, since we showed that the structural property of no idle time in the optimal rescheduling solution still holds, but the problem is already weakly NP-hard. Total weighted completion time is a step further from the easier, polynomial cases of minimizing maximum lateness and total completion time, but potentially more tractable than the cases with idle times in the solution as total number of late jobs, which is strongly NP-hard. The chapter will show that the problem is solvable in pseudo-polynomial time and this highlights the special intermediate status of total weighted completion time.

The chapter is structured as follows. In the first section, we introduce structural properties that will be exploited in the following algorithms. In the second section, we present a dynamic programming (DP) algorithm to solve the problem to optimality. In the third section, we present two approximation algorithms, a fully polynomial approximation scheme (FPTAS) and a polynomial-time approximation algorithm with a performance ratio bounded with respect to the number of jobs.

Table 4.1 Notation for problem $1|\Delta_{max} \leq Y|\sum w_j C_j$.

Notation	Description
n_O	old job scheduled last
L	set of new jobs scheduled before n_O
R	set of new jobs scheduled after n_O
$O_{<j}\,(N_{<j})$	subset of old (new) jobs i, with i scheduled before j in π^* (with i such that $\frac{w_i}{p_i} \geq \frac{w_j}{p_j}$)
$O_{>j}\,(N_{>j})$	subset of old (new) jobs $i \neq j \in O \backslash O_{<j}$ ($i \neq j \in N \backslash N_{<j}$)
$O_{<j}(\sigma)\,(N_{<j}(\sigma))$	subset of old (new) jobs scheduled before job j in schedule σ
$T = P_O + Y$	latest feasible completion time of job n_O

4.1 Structural properties

In the following, we list several properties that hold for the optimal solution of the problem, which allow to impose certain assumptions without loss of generality or to exclude trivial cases from further consideration.

We will introduce some additional notation specific to this problem and, for ease of reference, give a summary of the main notation introduced in Table 4.1. The WSPT ordering rule does indeed play an important role. Whenever relevant, we will break ties by scheduling old jobs before new jobs, then shorter jobs before longer jobs, and in the case of identical data, jobs will be sorted lexicographically by index number.

First, consider the WSPT schedule of both old and new jobs which minimizes the total weighted completion time over all jobs. If such a schedule satisfies $\Delta_{max} \leq Y$, then this schedule automatically solves $1|\Delta_{max} \leq Y|\sum w_j C_j$.

In Chapter 2, the structural analysis showed that there is a valid ordering property that applies to the old jobs when solving this problem, i.e. old jobs are ordered in WSPT order in an optimal solution (Property 2.1.2) and there is no unforced idle time in an optimal solution (Property 2.2.2).

Thus, in the structure of an optimal schedule there are new jobs that are inserted in between the old jobs $1,\ldots,n_O$ ordered in WSPT order. All old jobs sequenced later than a newly inserted job are delayed, which means that delays are propagated

throughout the sequence of old jobs and that the largest disruption is always attained for the last old job. We introduce the following property.

Property 4.1.1. *In an optimal schedule for the* $1|\Delta_{max} \leq Y|\sum w_j C_j$ *problem,* $\Delta_{max} = \Delta_{n_O}$ *and* $C_{n_O} \leq P_O + Y$.

Let us refine the above description by characterizing the structure of the optimal solution as follows.

Property 4.1.2. *An optimal schedule of* $1|\Delta_{max} \leq Y|\sum w_j C_j$ *consists of two disjoint subsets: the subset* $O \cup L$ *containing all old jobs* O *and those new jobs* L *scheduled before the last old job* n_O*, and the subset* R *of new jobs scheduled after the last old job. In both* $O \cup L$ *and* R*, all jobs are sorted in WSPT order.*

Proof. By Property 4.1.1, any job $1, \ldots, n_O$ satisfies the disruption constraint for any ordering of the new jobs in L. Hence, the WSPT order of the job set $O \cup L$ minimizes the objective function without affecting the feasibility of the schedule. The same holds for jobs in R, that can be ordered in WSPT order to minimize the objective function. □

The solution structure is visualized in Figure 4.1.

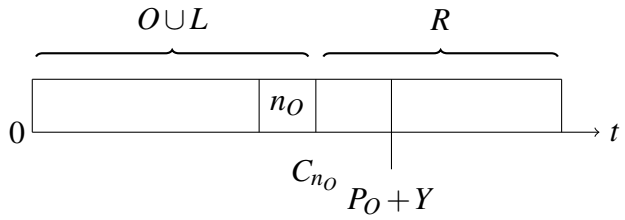


Figure 4.1 Structure of an optimal schedule of problem $1|\Delta_{max} \leq Y|\sum w_j C_j$.

Property 4.1.2 implies the following corollary.

Corollary 4.1.1. *Each new job* j *with* $\frac{w_j}{p_j} \leq \frac{w_{n_O}}{p_{n_O}}$*, is never scheduled before job* n_O *in an optimal schedule.*

From Corollary 4.1.1, we can derive the straightforward assumption that all new jobs with $\frac{w_j}{p_j} \leq \frac{w_{n_O}}{p_{n_O}}$ will be scheduled after the last job n_O in WSPT order and never contribute to the disruption of the old jobs. Thus, we can set aside these jobs, solve the remaining instance to optimality, and then append these jobs in WSPT order at

the end of the computed sequence. Formally, we can assume the following property to hold w.l.o.g. in the remainder of the paper:

Property 4.1.3. *All new jobs $j \in N$ fulfill $\frac{w_j}{p_j} > \frac{w_{n_O}}{p_{n_O}}$.*

In analogy to the previous observation, we can conclude that new jobs j with processing time $p_j > Y$ can never be inserted to the left of any old job.

Thus, the following property will be assumed to hold w.l.o.g.

Property 4.1.4. *All new jobs $j \in N$ fulfill $p_j \leq Y$.*

For an optimal schedule, consider the completion time $C^*_{n_O}$ of job n_O and let $T = P_O + Y$ be the maximum feasible completion time of n_O in this schedule. For the first job $j \in N$ scheduled after n_O it must hold that $C^*_{n_O} + p_j > T$, because otherwise it would be feasible, and by Property 4.1.3 optimal, to swap j and n_O. Bounding p_j by $p_{\max} = \max_{j \in N} p_j$ and invoking Property 4.1.1, it follows:

Corollary 4.1.2. *In an optimal schedule, the completion time C_{n_O} of the last old job is included in the interval $(T - p_{\max}, T]$, and for the first new job j scheduled after n_O there is $C_j > T$.*

4.2 A dynamic programming algorithm

In the following, we use the structure of an optimal schedule in order to derive a pseudo-polynomial time dynamic programming (DP) algorithm solving $1|\Delta_{max} \leq Y|\sum w_j C_j$ exactly.

Note that the rescheduling problem shows similarities with the *interfering jobs* problem in the multi-agent scheduling setting (see for instance Agnetis et al. (2014)). In problem $1|IN, L^2_{max} \leq Q|\sum w_j C_j$, jobs of agent 2 are included in those of agent 1, and while total weighted completion time is minimized over all jobs, only agent 2 has to keep the maximum lateness of its own jobs under a threshold Q. The structural analysis of the rescheduling problem showed that old jobs are ordered in WSPT order in an optimal solution (Property 2.1.2). This means that under specific due dates, the bound on L^2_{max} corresponds to the bound on the disruption. Whereas this problem is generally strongly NP-hard, we will show in the following that this special case is solvable in pseudo-polynomial time.

The dynamic programming algorithm starts with the following assumptions. The old jobs $1,\ldots,n_O$ are scheduled and indexed by the WSPT rule (Property 2.1.2) and the maximum disruption equals the disruption of the last old job n_O (Property 4.1.1). Index the new jobs $n_O+1,\ldots,n_O+n_N$ by the WSPT rule too. By Property 4.1.2 there are two sets L and R of new jobs scheduled by WSPT, where L contains all new jobs scheduled before n_O in an optimal schedule and R all those scheduled later. Moreover, the completion time of n_O will never exceed the time $T = P_O + Y$ (Corollary 4.1.2).

Let us call C the starting time of the first job in R. It holds that $C \geq T - p_{\max}$ (Corollary 4.1.2). Also, since it is never optimal to insert idle times, it follows that $C \leq T$ and that the total length of the jobs in L must be exactly $C - P_O$.

Summarizing the above, we can reduce the problem to finding the optimal subset L of new jobs that will be scheduled before n_O for the given optimal starting time C of the new jobs in R.

The optimal solution can be found via dynamic programming using state generation, and/or using recursive functions. We use the first formulation, where we generate only feasible states and keep only the dominating ones. Although this results in a longer presentation, it allows us to show step by step how the optimal solution is generated while keeping the overall time complexity the same.

The dynamic programming algorithm will consider a collection of states $(C,P,f)_j$ for $j = 1,\ldots,n_N$ defined as follows. Each state contains the total weighted completion time $f(\sigma_j)$, which we will denote as f, of an optimal solution σ_j to the subproblem that considers all old jobs and new jobs $n_O+1,\ldots,n_O+j$, with a starting time C of the jobs scheduled in R and with an exact total length P of the new jobs scheduled in subset L (see Fig. 4.2).

We seek the optimal solution σ^* to the problem with all jobs scheduled, i.e. a state with minimum objective function value $f^* = f(\sigma^*)$ and the corresponding optimal starting time C^* of job set R, with no idle time, i.e. the state $(C^*, C^* - P_O, f^*)_{n_N}$. The set of states is generated via dynamic programming by the following recursive algorithm. The corresponding schedule can be retrieved by backtracking.

For every pair of parameters (C,P) only the states with minimal objective function value are relevant for reaching the optimal solution. Thus, we say that state $(C,P,f')_j$ *dominates* state $(C,P,f'')_j$ if $f' < f''$.

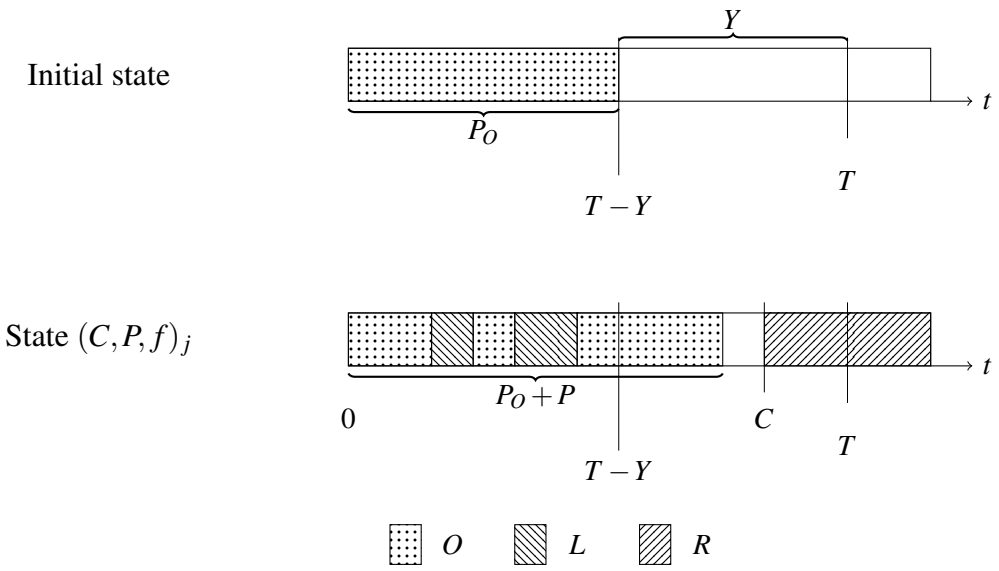


Figure 4.2 Structure of a DP state $(C,P,f)_j$.

Algorithm 6 shows the steps to compute an optimal solution. For any possible value of $C = T - p_{\max} + 1, \ldots, T$, states $(C,0,f(\pi^*))_0$ are defined, where $f(\pi^*)$ is the total weighted completion time of all the old jobs ordered by the WSPT rule, assuming they are starting at time 0, i.e. $f(\pi^*) = \sum_{j \in O} w_j C_j(\pi^*)$.

At every stage of the recursion, the algorithm considers two possible solutions for a new job $n_O + j$, namely putting job $n_O + j$ in R (Eq. 4.1 in Algorithm 6) or in L (Eq. 4.2 in Algorithm 6). The entry of the objective function in (4.1) is given by the increment caused by the contribution of $n_O + j$ scheduled at the end of set R. In (4.2) it is given by the old jobs scheduled after $n_O + j$ that are shifted to the right by p_{n_O+j} time units and by the contribution of $n_O + j$ which will be scheduled after the old jobs with a better WSPT ratio and the available "slot" of P time units.

Note that a new state generated in (4.2) is only feasible if the following two conditions are met. On one hand, the total length P of new jobs in L must be bounded by Y, i.e. the new state in (4.2) can only be generated if $P + p_{n_O+j} \leq Y$. On the other hand, the starting time C of jobs in R should not overlap with any job in $O \cup L$, i.e. $P_O + P + p_{n_O+j} \leq C$. Since $C - P_O \leq T - P_O = Y$, the latter condition dominates the former.

We establish the following:

Algorithm 6 Algorithm DP for $1|\Delta_{max} \leq Y|\sum w_j C_j$

Order and index new jobs by WSPT.
Compute $p_{\max} := \max\{p_j \mid j \in N\}$
Initialize states $(C, 0, f(\pi^*))_0 \quad \forall C = T - p_{\max} + 1, \ldots, T$
for $j = 1, \ldots, n_N$ **do**
  for $(C, P, f)_{j-1}$ **do**
    Add job $n_O + j$ to every state either to R or to L generating new states:

$$(C, P, f + w_{n_O+j}(C + P_{N<n_O+j} + p_{n_O+j} - P))_j \text{ and} \tag{4.1}$$

$$(C, P + p_{n_O+j}, f + w_{n_O+j}(P_{O<n_O+j} + P + p_{n_O+j}) + W_{O>n_O+j}\, p_{n_O+j})_j$$
$$\text{if } P + p_{n_O+j} \leq C - P_O \tag{4.2}$$

  end for
  eliminate all *dominated states*
end for
Find the state $(C, C - P_O, f)_{n_N}$ with minimal f among all $C = T - p_{\max} + 1, \ldots, T$.

Theorem 4.2.1. *Algorithm DP solves* $1|\Delta_{max} \leq Y|\sum w_j C_j$ *to optimality in* $O(n_N P_N\, p_{\max})$ *time.*

Proof. It remains to evaluate the running time of algorithm DP. The recursion is computed by iterating over all n_N new jobs and all states. The number of states is bounded by the number of possible values of C, which is $p_{\max}$, and by the number time units P available for scheduling new jobs left of n_O, which is bounded by $Y \leq P_N$. By domination, only one value of f remains for every combination of C and P. This yields an overall running time of $O(n_N P_N\, p_{\max})$. □

4.3 Approximation algorithms

4.3.1 A fully polynomial approximation scheme

In this section, we will derive a *fully polynomial approximation scheme* (FPTAS) based on Algorithm DP of Section 4.2. The FPTAS employs two scaling and one reduction step to the dynamic program. Scaling is applied to the starting times C of the jobs in R and to the objective function values. Therefore, we cannot apply the dominance criterion. Reduction is performed on the length P of jobs in L. Since P_L is strictly constrained by Y, we cannot round this value, but we partition the range for P into intervals and restrict our attention to a limited number of representatives.

A similar strategy was applied, e.g., for an FPTAS for the Subset Sum Problem in Kellerer et al. (2003).

For the scaling of the objective function we give two alternative approaches: The first employs a partitioning of the objective values into equal-sized intervals which requires upper and lower bounds linked by a bounded performance guarantee. The second is a geometric rounding approach, which does not rely on lower bounds but is not strongly polynomials since its running time depends on the logarithm of the largest input values.

For the first approach, let $f^{LB} \leq f^* \leq f^{UB}$ be lower and upper bounds on the optimal objective function value of $1|\Delta_{max} \leq Y|\sum w_j C_j$. These bounds can be derived from any approximate solution σ^A with performance guarantee ρ by setting $f^{UB} = f(\sigma^A)$ and $f^{LB} = f(\sigma^A)/\rho$. For the definition of the lower bound f^{LB}, we will use a $(n+4)$-approximation that will be showed in the next section (Theorem 4.3.3, Section 4.3.2).

Define an accuracy parameter $\eta := \frac{\varepsilon}{n_N} f^{LB}$. We partition the range of objective function values into a sequence of N_f intervals of identical width η such that $N_f \eta \geq f^{UB}$:

$$(0,\eta], (\eta, 2\eta], \ldots, ((N_f - 1)\eta, N_f \eta] \tag{4.3}$$

For the second approach we partition the range of objective function values into a sequence of $N_g := \lfloor n_N \cdot \log_{1+\varepsilon}(f^{UB}) \rfloor + 2$ intervals of geometrically increasing width, as it is often done for deriving approximation schemes, see e.g. Boeckmann et al. (2023):

$$(0,1], (1,(1+\varepsilon)^{\frac{1}{n_N}}], ((1+\varepsilon)^{\frac{1}{n_N}}, (1+\varepsilon)^{\frac{2}{n_N}}], \ldots, ((1+\varepsilon)^{\frac{(N_g-1)}{n_N}}, (1+\varepsilon)^{\frac{N_g}{n_N}}] \tag{4.4}$$

The number of these intervals is polynomial in the input size and $\frac{1}{\varepsilon}$ since $N_g \in O\left(n_N \cdot \frac{1}{\varepsilon} \cdot \log\left(f^{UB}\right)\right)$. Clearly, the intervals cover the whole range $\{0, \ldots, f^{UB}\}$ of possible objective values.

For the starting time C of jobs in R a second accuracy parameter $\eta_C := \lfloor \varepsilon p_{\max} \rfloor$ is introduced. Note that w.l.o.g. we can assume $\eta_C \geq 1$ since $\varepsilon p_{\max} < 1 \Longleftrightarrow p_{\max} < 1/\varepsilon$ would imply that the running time of the exact DP-algorithm of Section 4.2 is polynomial in n_N and $1/\varepsilon$. Recall that $T - p_{\max} < C \leq T$. The range for C is

partitioned into $N_C + 1$ intervals of identical integer width η_C:

$$\begin{aligned}(T - p_{\max}, T - p_{\max} + \eta_C], (T - p_{\max} + \eta_C, T - p_{\max} + 2\eta_C], \ldots, \\ (T - p_{\max} + N_C\eta_C, T - p_{\max} + (N_C + 1)\eta_C]\end{aligned} \tag{4.5}$$

N_C is chosen as the smallest integer such that $T - p_{\max} + (N_C + 1)\eta_C > T$. Thus, we get $N_C\eta_C \leq p_{\max}$ and trivially $N_C \leq p_{\max}$. The former condition can be relaxed to

$$N_C \varepsilon p_{\max} \leq N_C(\lfloor \varepsilon p_{\max} \rfloor + 1) \leq p_{\max} + N_C \leq 2p_{\max}. \tag{4.6}$$

This yields $N_C \leq \lceil \frac{2p_{\max}}{\varepsilon p_{\max}} \rceil = \lceil \frac{2}{\varepsilon} \rceil$. It will be convenient to denote the set of all interval endpoints in (4.5) as $\tilde{C} := \{T - p_{\max} + j\,\eta_C \mid j = 0, 1, \ldots, N_C, N_C + 1\}$.

As a third scaling step, the range of total processing times P is partitioned into N_P intervals of identical width $\eta_P := \frac{\varepsilon}{n_N} Y$:

$$(0, \eta_P], (\eta_P, 2\eta_P], \ldots, ((N_P - 1)\eta_P, N_P\eta_P] \tag{4.7}$$

Since $P \leq Y$, we have $N_P = \lceil \frac{n_N Y}{\varepsilon Y} \rceil = \lceil \frac{n_N}{\varepsilon} \rceil$.

As in Section 4.2, we will keep states $(C, P, f)_j$ with the same meaning as before. However, we do not eliminate dominated states but keep states for *all reachable* objective function values. On the other hand, the set of states will be reduced by the following three steps:

1. Scaling of R-starting times: We do not consider *all* values of C, but only right interval endpoints from (4.5), i.e. the elements of $\tilde{C}$ (including the last endpoint which is $> T$).

2. Reducing values of total processing times P: For every C and f we keep only two states $(\tilde{P}_i, f)$, $i = 1, 2$, among all states (C, P, f) with P in the same interval w.r.t. (4.7), namely the one with minimal and with maximal value P. Throughout the algorithm the current reduced set of total processing times of jobs in subset L will be denoted by $\tilde{P}$.

3. Scaling of the objective function: Every generated value f is *rounded up* to the nearest interval bound in (4.3) or in (4.4), depending on the chosen approach.

In addition, the overall DP algorithm with the scaled parameter space is iteratively run n_N times. By iterating, we *guess* the new job $n_O + j^*$ with the *largest weight* among all jobs in R, i.e. placed right of n_O. It follows, that all new jobs $n_O + j$ with $w_{n_O+j} > w_{n_O+j^*}$ must be placed in L, on the left side of n_O. Then, the following pre-processing step is done in each such run.

4. Given a *guess* of j^*, add all new jobs $n_O + j$ with $w_{n_O+j} > w_{n_O+j^*}$ to the set of old jobs and insert them in WSPT order left of n_O. Recall that by Property 4.1.3 no new job with smaller WSPT ratio would be scheduled right of n_O. Then run the FPTAS with Y reduced by their total processing time. If this reduced Y is negative, the guess for j^* is infeasible. In the resulting execution of the FPTAS we know that there are no new jobs with weight greater than $w_{n_O+j^*}$.

The initialization of the states is done as follows:

5. For any *guess* of j^*, the algorithm is initialized with states $(\tilde{C}, 0, f_0)_0$, where f_0 is computed as the total weighted completion time of all the old jobs and the new jobs $n_O + j$ with $w_{n_O+j} > w_{n_O+j^*}$ ordered with the WSPT rule, assuming they are starting at time 0.

Algorithm 7 shows the pseudo-code to compute the approximate solution.

To prove that the above algorithm gives actually an ε-approximation we will start from the exact DP algorithm (Algorithm 6) and iteratively add the Steps 1 to 4 listed above. At first we consider the scaling of R-starting times from C to $\tilde{C}$:

Property 4.3.1. *For every state $(C, P, f)_j$ in the exact dynamic program, there exists a state $(\tilde{C}, P, f_1)_j$ in the dynamic program with scaled R-starting times, such that*

$$f_1 \leq f + \varepsilon f.$$

Proof. Consider a state $(C, P, f)_j$ generated in the optimal dynamic program and the associated schedule. Call R_j the set of jobs that are scheduled after the last old job n_O in this schedule. Round up C to $\tilde{C}$, i.e. the next right-endpoint of the intervals defined in (4.5). In the dynamic program the same operations are executed for every value of C. Therefore, there exists a state $(\tilde{C}, P, f_1)_j$, which corresponds to the same schedule of jobs and with the starting time of the jobs in R shifted by $\tilde{C} - C \leq \eta_C$. Given the

Algorithm 7 Algorithm FPTAS for $1|\Delta_{max} \leq Y|\sum w_j C_j$

for $j^* = 1,\ldots,n_N$ **do**
  Preprocess schedule according to Step 4.
  If the guess of j^* is infeasible, go to next j^*.
  Initialize states $(\tilde{C},0,f_0)_0$ for all feasible $\tilde{C}$ by Step 5.
  for $j = 1,\ldots,n_N$, $j \neq j^*$, $w_{n_O+j} \leq w_{n_O+j^*}$ **do**
    for all states $(\tilde{C},\tilde{P},\tilde{f})_{j-1}$ **do**
      Add job $n_O + j$ to $(\tilde{C},\tilde{P},\tilde{f})_{j-1}$ either to R or to L generating new states:

$$(\tilde{C},\tilde{P},f + w_{n_O+j}(\tilde{C} + \sum_{\ell=n_O+1}^{n_O+j} p_\ell - \tilde{P}))_j \text{ and} \tag{4.8}$$

$$(\tilde{C},\tilde{P} + p_{n_O+j}, f + w_{n_O+j}(P_{O<n_O+j} + \tilde{P} + p_{n_O+j}) + W_{O>n_O+j}\, p_{n_O+j})_j$$
$$\text{if } \tilde{P} + p_{n_O+j} \leq \tilde{C} - P_O \tag{4.9}$$

      Round up the new objective functions obtained to the nearest interval bound
      in (4.3) or in (4.4), according to Step 3.
      Reduce states according to Step 2.
    end for
  end for
end for

total weight W_{R_j} of the jobs scheduled in R_j, $f_1 \leq f + W_{R_j}\,\eta_C$. By Property 4.1.4 we have

$$f_1 \leq f + W_{R_j}\,\eta_C \leq f + W_{R_j}\,\varepsilon p_{\max} \leq f + W_{R_j}\,\varepsilon Y \leq f + \varepsilon f. \tag{4.10}$$

The last inequality holds since all jobs in R_j finish after $T \geq Y$ (Corollary 4.1.2). □

Next we consider the reduction of total processing times from P to $\tilde{P}$.

Property 4.3.2. *For every state $(\tilde{C},P,f_1)_j$ in the dynamic program with R-starting times scaled from C to $\tilde{C}$, there exist values $\tilde{P}_1 \leq P \leq \tilde{P}_2$ with $\tilde{P}_2 - \tilde{P}_1 \leq \frac{\varepsilon}{n_N} Y$ with states $(\tilde{C},\tilde{P}_i,f_2)_j$ in the dynamic program with scaled R-starting times and reduced total profit values $\tilde{P}$ such that*

$$f_2 \leq f_1 + j\frac{\varepsilon}{n_N} f^*.$$

Proof. The statement is shown by induction (it is trivially true as long as no *Reduce* was performed, e.g. for $j = 1$, because up to that point $P = \tilde{P}_i$ for some $i \in \{1,2\}$). The situation, where P is still the only value in its interval, is represented by the case $\tilde{P}_1 = P = \tilde{P}_2$.

Now consider a state $(\tilde{C}, P, f_1)_j$ in the dynamic program with scaled R-starting times which was generated by adding $n_O + j$ to some earlier state $(\cdot)_{j-1}$. There are two cases how the new state $(\tilde{C}, P, f_1)_j$ might have been generated.

Case I: job $n_O + j$ was added at the right side. Then there exists a DP-state $(\tilde{C}, P, f_1 - w_{n_O+j}(\tilde{C} + \sum_{\ell=n_O+1}^{n_O+j} p_\ell - P))_{j-1}$ with R-reduced starting times. By induction, there exist two values $\tilde{P'}_i$, $i = 1,2$, with $\tilde{P'}_1 \leq P \leq \tilde{P'}_2$ and $\tilde{P'}_2 - \tilde{P'}_1 \leq \frac{\varepsilon}{n_N} Y$, and with states $(\tilde{C}, \tilde{P'}_i, z)_{j-1}$ such that $z \leq f_1 - w_{n_O+j}(\tilde{C} + \sum_{\ell=n_O+1}^{n_O+j} p_\ell - P) + (j-1)\frac{\varepsilon}{n_N} f^*$. Adding job $n_O + j$ to this state at the right side (as it is done in (4.8)) yields to the new state $(\tilde{C}, \tilde{P'}_i, \tilde{f}_i := z + w_{n_O+j}(\tilde{C} + \sum_{\ell=n_O+1}^{n_O+j} p_\ell - \tilde{P'}_i))_j$. Plugging in the induction hypothesis we get:

$$\begin{aligned}
\tilde{f}_i &\leq f_1 - w_{n_O+j}(\tilde{C} + \sum_{\ell=n_O+1}^{n_O+j} p_\ell - P) + (j-1)\frac{\varepsilon}{n_N} f^* + \\
&\quad + w_{n_O+j}(\tilde{C} + \sum_{\ell=n_O+1}^{n_O+j} p_\ell - \tilde{P'}_i) \\
&= f_1 + w_{n_O+j}(P - \tilde{P'}_i) + (j-1)\frac{\varepsilon}{n_N} f^* \\
&\leq f_1 + w_{n_O+j}\frac{\varepsilon}{n} Y + (j-1)\frac{\varepsilon}{n_N} f^*
\end{aligned}$$

Case II: job $n_O + j$ was added at the left side. Again, there exists an optimal state $(\tilde{C}, P - p_{n_O+j}, f_1 - W_{O>n_O+j} \cdot p_{n_O+j} - w_{n_O+j} \cdot (P + P_{O<n_O+j}))_{j-1}$ and the proposition guarantees the existence of two values $\tilde{P'}_i$, $i = 1,2$, with $\tilde{P'}_1 \leq P - p_{n_O+j} \leq \tilde{P'}_2$ and $\tilde{P'}_2 - \tilde{P'}_1 \leq \frac{\varepsilon}{n_N} Y$, with states $(\tilde{C}, \tilde{P'}_i, z)_{j-1}$ such that $z \leq f_1 - W_{O>n_O+j} \cdot p_{n_O+j} - w_{n_O+j} \cdot (P + P_{O<n_O+j}) + (j-1)\frac{\varepsilon}{n_N} f^*$. Adding job $n_O + j$ to this state at the left side yields the new states $(\tilde{C}, \tilde{P'}_i + p_{n_O+j}, \tilde{f}_i := z + W_{O>n_O+j} \cdot p_{n_O+j} + w_{n_O+j}(\tilde{P'}_i + P_{O<n_O+j} + p_{n_O+j}))_j$. Plugging in the induction hypothesis we get:

$$\begin{aligned}
\tilde{f}_i &\leq f_1 - W_{O>n_O+j} \cdot p_{n_O+j} - w_{n_O+j} \cdot (P + P_{O<n_O+j}) + (j-1)\frac{\varepsilon}{n_N} f^* + \\
&\quad + W_{O>n_O+j} \cdot p_{n_O+j} + \\
&\quad + w_{n_O+j}(\tilde{P'}_i + P_{O<n_O+j} + p_{n_O+j}) \\
&= f_1 + w_{n_O+j}(\tilde{P'}_i - P + p_{n_O+j}) + (j-1)\frac{\varepsilon}{n_N} f^* \\
&\leq f_1 + w_{n_O+j}\frac{\varepsilon}{n_N} Y + (j-1)\frac{\varepsilon}{n_N} f^*
\end{aligned}$$

The last inequality follows from the fact that $\tilde{P'}_1 - (P - p_{n_O+j}) \leq 0$ and $\tilde{P'}_2 - (P - p_{n_O+j}) \leq \frac{\varepsilon}{n_N} Y$.

It still remains to bound the term $w_{n_O+j} \frac{\varepsilon}{n_N} Y$. The following relation holds:

$$w_{n_O+j} \frac{\varepsilon}{n_N} Y \leq w_{n_O+j^*} \frac{\varepsilon}{n_N} C_{n_O+j^*} \leq \frac{\varepsilon}{n_N} f^* \tag{4.11}$$

This is guaranteed by the guess of job $n_O + j^*$ with largest weight among the jobs scheduled right of n_O which leads to the consideration of only new jobs $n_O + j$ with smaller weight than $n_O + j^*$ in the dynamic program; hence, $w_{n_O+j^*} \geq w_{n_O+j}$. Since $n_O + j^*$ is placed right of n_O, in an optimal schedule it holds that the completion time of job $n_O + j^*$ fulfills $C_{n_O+j^*} \geq T \geq Y$ (Corollary 4.1.2). Plugging in the bound given in (4.11) for both cases completes the induction. □

Finally, we consider the rounding up of objective function values.

Property 4.3.3. *For every state $(\tilde{C}, \tilde{P}, f_2)_j$ in the dynamic program with scaled R-starting times and reduced total profit values $\tilde{P}$, there exists a state $(\tilde{C}, \tilde{P}, f_3)_j$ in the final approximate DP such that*

$$f_3 \leq f_2 + j \frac{\varepsilon}{n_N} f^{LB} \text{ for intervals (4.3)},$$

$$\text{or } f_3 \leq (1+\varepsilon)^{\frac{j}{n_N}} f_2 \text{ for intervals (4.4)}.$$

Proof. Again, we can use an inductive argument. The initialization states $(\tilde{C}, 0, f_0)_0$ contain the original values of f.

Consider the dynamic program with scaled R-starting times and reduced total profit values $\tilde{P}$ that produces states $(\tilde{C}, \tilde{P}, f_2)_j$, for some $\tilde{C}, \tilde{P}, j$. In the execution of the algorithm one goes from an initial state $(\cdot)_0$ to a state $(\cdot)_{n_N}$ through a sequence $(\cdot)_0 \to (\cdot)_1 \to \cdots \to (\cdot)_{n_N}$. For the intervals (4.3) an additional rounding error of at most $\eta = \frac{\varepsilon}{n_N} f^{LB}$ is introduced in each iteration for $j = 1, \ldots, n_N$. For intervals (4.4), rounding up can increase the objective value at most by a factor of $(1+\varepsilon)^{\frac{1}{n_N}}$.

Note that the rounding of f-values does not have any influence on the generation of states. Therefore, for any intermediate step j, there exists by induction a state $(\tilde{C}, \tilde{P}, f_3)_j$, with either $f_3 \leq f_2 + j \frac{\varepsilon}{n_N} f^{LB}$ or $f_3 \leq (1+\varepsilon)^{\frac{j}{n_N}} f_2$, which proves the statement. □

Putting things together we get:

Theorem 4.3.1. *For an optimal state $(C,P,f^*)_{n_N}$ computed in the optimal (original) dynamic program, in the FPTAS (i.e. the dynamic program with scaled R-starting times, reduced total profit values $\tilde{P}$, and scaled objective function values) there exists a state $(\tilde{C},\tilde{P},f^A)_{n_N}$ such that*

$$f^A \leq (1+3\varepsilon)f^*.$$

Proof. Concatenating the statements of Propositions 4.3.1 to 4.3.3 for $j = n_N$ we consider every $(C,P,f)_{n_N}$ in the optimal dynamic program. For the scaling by intervals (4.3) there exists

$$f^A = f_3 \leq f_2 + \varepsilon f^{LB} \leq f_1 + \varepsilon f^* + \varepsilon f^{LB} \leq f + \varepsilon f + \varepsilon f^* + \varepsilon f^{LB} \leq (1+3\varepsilon)f^*.$$

For scaling by (4.4) there exists

$$f^A = f_3 \leq (1+\varepsilon)f_2 \leq (1+\varepsilon)(f_1+\varepsilon f^*) \leq (1+\varepsilon)(f^*+2\varepsilon f^*) = f^*+3\varepsilon f^*+2\varepsilon^2 f^*.$$

Plugging in a small enough accuracy parameter, e.g. $\frac{\varepsilon}{4}$, we reach the required approximation ratio in both cases. Removing dominated states can only strengthen the inequalities in Properties 4.3.1 to 4.3.3. □

Theorem 4.3.2. *Algorithm FPTAS runs in $O(n^4(\frac{1}{\varepsilon})^3 \min(n,\ \max(\log n,\ \log(w_{\max}),\ \log(p_{\max})))$ time.*

Proof. Completing the recursion requires a computing time of $O(n_N^2 N_C N_P \min\{N_f, N_g\})$. For the scaling by intervals of identical width with $N_f = \lceil \frac{n_N f^{UB}}{\varepsilon f^{LB}} \rceil$ according to (4.3) the $(n+4)-$approximation algorithm presented in Section 4.3.2 yields an upper bound with $f^{UB} \leq (n+4)f^*$. Choosing the valid lower bound $f^{LB} := \frac{f^{UB}}{n+4}$ we obtain $N_f \in O\left(\frac{n^2}{\varepsilon}\right)$. For geometric scaling according to (4.4) we recall that $N_g \in O\left(n \cdot \frac{1}{\varepsilon} \cdot \log\left(f^{UB}\right)\right)$. Thus, we employ a trivial upper bound $f^{UB} \leq n w_{\max}(n p_{\max})$ and get $N_g \in O\left(\frac{n}{\varepsilon} \cdot (\log n + \log(w_{\max}) + \log(p_{\max})\right)$. Taking the minimum among the two cases yields the stated overall running time of $O(n^4(\frac{1}{\varepsilon})^3 \min(n, \max(\log n,\ \log(w_{\max}),\ \log(p_{\max})))$. □

We can conclude from the proof of Theorem 4.3.2 that geometric rounding yields a better running time if the magnitude of all job data is not too large. However,

Table 4.2 Notation for rescheduling and resumable scheduling.

Notation	Description
σ_{RE}	a feasible schedule for the *Resumable* problem
σ_{RS}	a feasible schedule for the *Rescheduling* problem
σ^*_{RE}	an optimal schedule for the *Resumable* problem
σ^*_{RS}	an optimal schedule for the *Rescheduling* problem
$f^N(\sigma)$	total weighted completion time of new jobs in schedule σ
$f^N(\sigma_{RE})$	total weighted completion time of new jobs in a solution σ_{RE} of the *Resumable* problem
h_1	fixed unavailable time interval on the machine
$N^<$	set of jobs completing before h_1 in a solution σ_{RE} of the *Resumable* problem
$N^>$	set of jobs completing after h_1 in a solution σ_{RE} of the *Resumable* problem

if $n \leq \log(w_{\max})$ or $n \leq \log(p_{\max})$, then the scaling by intervals of identical width gives a shorter running time complexity.

4.3.2 Bounded approximation ratio

In this section we will describe a polynomial time approximation algorithm with an approximation ratio of $n+4$. Although this ratio does not seem too attractive on its own, the achieved bound is a crucial prerequisite for the FPTAS in Section 4.3.1. The algorithm makes use of a known approximation result for a related scheduling problem, namely scheduling with unavailability constraints. One version of this problem, denoted as $1|h_1, Res|\sum w_j C_j$, minimizes the total weighted completion time when a fixed unavailable time h_1 occurs on a machine and where the jobs are allowed to be resumed, i.e. where they can be interrupted before completion and completed when the machine is available again (see Lee (1996), Wang et al. (2005)). For brevity, we call it the *Resumable* problem (*RE*). To avoid confusion between the two scheduling problems we give a list of notations in Table 4.2.

To derive a bounded factor approximation algorithm for $1|\Delta_{max} \leq Y|\sum w_j C_j$, we define a transformation between a solution of the resumable scheduling problem and one of our rescheduling problem. In a first step, the connection to this problem is used to show that a heuristic for the first problem allows to derive an approximation

algorithm of factor 3 for the contribution of new jobs to the weighted completion time in a solution of the rescheduling problem. In a second step, we show that the weighted completion time of the old jobs in any feasible solution, that satisfies all properties from Section 4.1, approximates by a factor of $n+1$ its value in an optimal solution.

First, define the unavailable time of the machine as the interval $h_1 = [T - P_O, T]$. Given a solution σ_{RE} to an instance of *RE* consisting only of the new jobs, we can transform it to a solution σ_{RS} of the *Rescheduling* problem (*RS*) as follows. Let us call the crossover job in *RE*, the one which starts before h_1 and finishes after. If none is split, we call the crossover job the one following the unavailable time. Shift the unavailable time before the crossover job and replace it by the old jobs. Reorder the whole set of new and old jobs before the crossover job in WSPT order. Note that the completion time of the crossover job does not change through this transformation and the completion time of a new job j which is scheduled left of the crossover job is increased by $P_{O<j}$ (recall that set $O_{<j}$ contains the old jobs scheduled before job j in π^*).

In other words, we define a transformation $\sigma_{RE} \rightarrow \sigma_{RS}$ obtained by modifying the completion times of the jobs of the first schedule through the following function α.

$$C_j^{RS} = \alpha(C_j^{RE}) : \begin{cases} C_j^{RE}, & \forall j \in N^{>} \\ C_j^{RE} + P_{O<j}, & \forall j \in N^{<} \\ P_{N<j} + P_{O<j} + p_j, & \forall j \in O \end{cases} \tag{4.12}$$

Transformation $\sigma_{RE} \rightarrow \sigma_{RS}$ builds a feasible schedule for the rescheduling problem (see figure 4.3).

We can also define the inverse function α^{-1}, that given a solution of the rescheduling problem returns a solution of the resumable problem:

$$C_j^{RE} = \alpha^{-1}(C_j^{RS}) : \begin{cases} C_j^{RS}, & \forall j \in R \\ C_j^{RS} - P_{O<j}, & \forall j \in L \end{cases} \tag{4.13}$$

Recall that R and L are the sets of new jobs respectively completing before and after T in solution σ_{RS} of the *Rescheduling* problem. By inserting the interval $h_1 = [T - P_O, T]$ and preemptively scheduling the jobs with the above completion times, the schedule generated is again σ_{RS}.

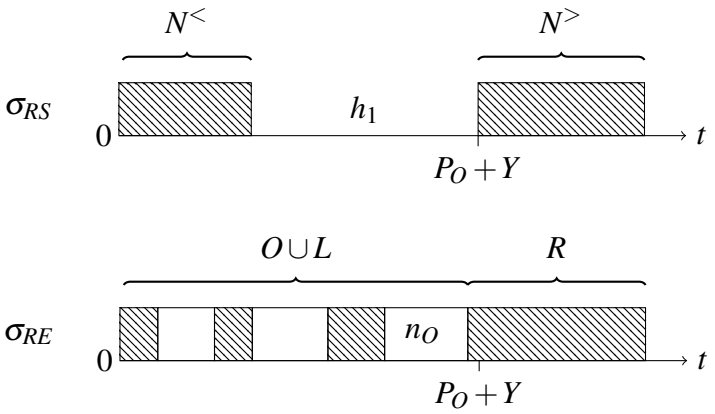


Figure 4.3 Transformation between a schedule with an unavailable time interval and a rescheduling solution.

Given the total weighted completion time $f^N(\sigma_{RE})$ of the jobs in a solution of the resumable problem, the objective function value of the schedule σ_{RS}, obtained through the transformation α, is computed as:

$$f(\sigma_{RS}) = f^N(\sigma_{RE}) + \sum_{j \in N^<} w_j P_{O<j} + \sum_{j \in O} w_j C_j(\sigma_{RS}) \tag{4.14}$$

Considering the reverse direction, one can take any feasible solution σ_{RS} of *RS* and move all old jobs to the right, directly before the last old job. In this way, all new jobs placed left of the last old job in σ_{RS} will receive an earlier or unchanged completion time, while the completion times of new jobs placed right of the last old job in σ_{RS} will remain unchanged. This yields a solution σ_{RE} of *RE* with a consecutive block of length P_O as an unavailable period in *RE*.

Let $\sigma^\star$ be an optimal solution to *RS*. Then the above yields

$$f^N(\sigma^\star_{RE}) \leq f(\sigma^\star). \tag{4.15}$$

Moreover, for any feasible solution σ_{RE} there is

$$\sum_{j \in N^<} w_j P_{O<j} \leq f(\sigma^\star), \tag{4.16}$$

since in an optimal solution of *RS*, the old jobs with WSPT ratio greater or equal than j will precede the new job j, no matter whether j is placed left or right of the last old job (recall Property 4.1.2).

The approximation of the *Rescheduling* problem works as follows. At first, we compute a heuristic solution σ_{RE}^H for *RE* using the algorithm by Wang et al. (2005). Let $\sigma_{RE}^\star$ be the optimal solution to *RE*. The heuristic returns a solution, which is proved by the authors to satisfy $f(\sigma_{RE}^H) \leq 2f(\sigma_{RE}^\star)$. Then we apply the transformation α described above and obtain from σ_{RE}^H a solution σ^H of *RS*. Plugging (4.15) and (4.16) in (4.14), we get:

$$f(\sigma^H) = f^N(\sigma_{RE}^H) + \sum_{j \in N^<} w_j P_{O<j} + \sum_{j \in O} w_j C_j(\sigma^H) \tag{4.17}$$

$$\leq 2f(\sigma_{RE}^\star) + \sum_{j \in N^<} w_j P_{O<j} + \sum_{j \in O} w_j C_j(\sigma^H) \tag{4.18}$$

$$\leq 3f(\sigma^\star) + \sum_{j \in O} w_j C_j(\sigma^H) \tag{4.19}$$

The following Lemma bounds the last term in (4.19).

Lemma 4.3.1. *For the heuristic solution σ^H there is:*

$$\sum_{j \in O} w_j C_j(\sigma^H) \leq (n+1) f(\sigma^*)$$

Proof. Observe that for every old job j

$$C_j(\sigma^H) \leq P_{O<j} + P_{N<j}.$$

For each old job $j = 1, \ldots, n_O$, define a partition of the set $N_{<j}$ in three subsets S_{j1}, S_{j2}, S_{j3}. Recall that set $N_{<j}$ contains new jobs with larger WSPT ratio than j (Table 1.1). Let S_{j1} be the subset containing the new jobs $i \in N_{<j}$ that are scheduled before j in an optimal solution of the Rescheduling problem, S_{j2} the subset containing the new jobs $i \in N_{<j}$ that are scheduled after j in an optimal solution and with $w_i \geq w_j$, S_{j3} the subset containing the new jobs $i \in N_{<j}$ that are scheduled after j in an optimal solution and with $w_i < w_j$. The latter implies $p_i \leq p_j$, given that $\frac{w_i}{p_i} > \frac{w_j}{p_j}$.

It holds that:

$$\begin{aligned}\sum_{j \in O} w_j C_j(\sigma^H) &\leq \sum_{j \in O} w_j \left(\sum_{i \in N_{<j}} p_i + \sum_{i \in N_{<j}} p_i \right) \\ &= \sum_{j \in O} w_j \left(\sum_{i \in N_{<j}} p_i + \sum_{i \in S_{j1}} p_i \right) + \sum_{j \in O} w_j \left(\sum_{i \in S_{j2}} p_i \right) + \sum_{j \in O} w_j \left(\sum_{i \in S_{j3}} p_i \right)\end{aligned}$$

Consider the three terms of the sum separately. We first have

$$\sum_{j\in O} w_j \left(\sum_{i\in N_{<j}} p_i + \sum_{i\in S_{j1}} p_i \right) \leq \sum_{j\in O} w_j C_j(\sigma^*) \leq f(\sigma^*).$$

For every $j \in O$ let $j_{(last)}$ be the new job in S_{j2} scheduled last in the optimal solution. We get:

$$\sum_{j\in O} w_j \left(\sum_{i\in S_{j2}} p_i \right) \leq \sum_{j\in O} w_j C_{j_{(last)}}(\sigma^*) \leq \sum_{j\in O} w_{j_{(last)}} C_{j_{(last)}}(\sigma^*) \leq n_O f(\sigma^*)$$

Notice that if $j_{(last)} \neq i_{(last)}, \forall i,j \in O, i \neq j$, then the n_O factor cancels out, i.e. $\sum_{j\in O} w_{j_{(last)}} C_{j_{(last)}}(\sigma^*) \leq f(\sigma^*)$. Finally, there is

$$\sum_{j\in O} w_j \left(\sum_{i\in S_{j3}} p_i \right) \leq \sum_{j\in O} w_j \left(\sum_{i\in S_{j3}} p_j \right) \leq \sum_{j\in O} w_j n_N C_j(\sigma^*) \leq n_N f(\sigma^*).$$

In summary, we conclude

$$\sum_{j\in O} w_j C_j(\sigma^H) \leq (1 + n_O + n_N) f(\sigma^*) = (n+1) f(\sigma^*).$$

□

Now we immediately obtain our approximation result.

Theorem 4.3.3. *The heuristic solution σ^H gives an $(n+4)$-approximation of problem $1|\Delta_{max} \leq Y|\sum w_j C_j$.*

Proof. It suffices to plug in the statement of Lemma 4.3.1 in (4.19) to show that $f(\sigma^H) \leq (n+4) f(\sigma^\star)$. □

Consider now the special case, where $n_O = 1$ ($\mathscr{NP}$-hard by Theorem 2.3.1). For this case, we show that the heuristic described above yields a solution σ^H which approximates the optimal solution by a factor 3.

Theorem 4.3.4. *If there is only one old job, then the heuristic solution σ^H gives a 3-approximation of problem $1|\Delta_{max} \leq Y|\sum w_j C_j$.*

Proof. Recall that by Property 4.1.3 we neglect all new jobs j with $\frac{w_j}{p_j} < \frac{w_1}{p_1}$. Therefore, for each new job j, $O_{<j} = \emptyset$.

Let us adapt equation (4.18) from above according to the above considerations. We have

$$f(\sigma^H) \leq 2f(\sigma^*_{RE}) + w_1 C_1(\sigma^H). \tag{4.20}$$

Consider the optimal solution $f(\sigma^*) = f^N(\sigma^*) + w_1 C_1(\sigma^*)$. Again, this solution can be constructed starting from a solution to *RE*, by pushing the old job so that it finishes exactly at time T and by allowing preemption for the first new job that follows job 1. From Corollary 4.1.2, we also know that in an optimal schedule it holds

$$C_1(\sigma^*) \geq T - p_{\max}. \tag{4.21}$$

Since $f^N(\sigma^*_{RE}) \leq f^N(\sigma^*)$, it follows

$$f(\sigma^*) = f^N(\sigma^*) + w_1 C_1(\sigma^*) \geq f^N(\sigma^*_{RE}) + w_1(T - p_{\max}). \tag{4.22}$$

Given, that $C_1(\sigma^H) \leq T$ because of the disruption constraint, we obtain from (4.20) and (4.22)

$$\begin{aligned} f(\sigma^H) - f(\sigma^*) &\leq 2f^N(\sigma^*_{RE}) + w_1 T - (f^N(\sigma^*_{RE}) + w_1(T - p_{\max})) && (4.23)\\ &= f^N(\sigma^*_{RE}) + w_1 p_{\max} && (4.24)\\ &\leq f(\sigma^*) + w_1 p_{\max}. && (4.25) \end{aligned}$$

To show that $w_1 p_{\max} \leq f(\sigma^*)$ let us denote as m the largest new job with processing time $p_{\max}$. Job m belongs to one of the three subsets S_{11}, S_{12} or S_{13}, as defined above. In any of the three cases, as it has been shown above, $w_1 p_{\max} \leq f(\sigma^*)$, which gives us

$$f(\sigma^H) \leq 3f(\sigma^*) \tag{4.26}$$

from (4.25). □

Chapter 5

Exact algorithms for $1|no-idle, \sum \Delta \leq Y|L_{max}$

In this chapter we consider the minimization of the maximum lateness subject to a total time disruption constraint with non-delay schedules. This problem is shown to be NP-hard in a strong sense (Corollary 2.3.5, Chapter 2), therefore it is unlikely that there is an algorithm that solves it in polynomial time. The problem under consideration in this chapter does not allow the insertion of idle times in the solutions. This requirement is reasonable in production systems, where introducing idle times on the machines is often detrimental given the high running costs of production machines.

The chapter is structured as follows. In the first section, we introduce two integer programming models (IP). In the second section, we present the branch and memorize (BM) algorithm and several elements that improve its efficiency. To test the algorithm, we provide the results of several computational experiments run on randomly generated instances. The algorithm is compared with standard solvers, run on the best-performing mathematical programming model, and shown to outperform the IP solver.

5.1 Structural properties

In the following, several properties are listed that hold for an optimal solution of the problem. These will allow us to impose certain assumptions without loss of generality or to exclude trivial cases from further consideration.

First, consider the EDD schedule of both old and new jobs which trivially minimizes maximum lateness over all jobs. If such a schedule satisfies $\sum\Delta \leq Y$, then this schedule automatically solves $1|\sum\Delta \leq Y, no-idle|L_{max}$.

From Figure 2.6 (proof of Property 2.2.2), it follows that for this problem there is no ordering property which applies to the set of old jobs. So, both old and new jobs may be in a different order in the optimal schedule and the fact that the problem is strongly *NP*-hard which may be related to the fact that the EDD rule is not necessarily fulfilled in an optimal schedule. In this section, we try to identify situations where the EDD order applies between jobs in an optimal solution. Let us focus to more general considerations.

Let $\sigma = \gamma\ i\ j\ \beta$ be a sequence where γ and β are sub-sequences of old and new jobs, and $i, j \in O$ with $d_i \leq d_j$. Let $\sigma' = \gamma\ j\ i\ \beta$ be the sequence obtained by swapping i and j in σ. For simplification purpose, let us denote by d_i^C (resp. d_j^C) the completion time $C_i(\pi^*)$ (resp. $C_j(\pi^*)$) in the initial schedule π^*. We have $d_i^C \leq d_j^C$ as $d_i \leq d_j$.

Property 5.1.1. *Whenever one of these two conditions holds:*

1. $p_i \geq p_j$ *and* $P_\gamma + p_j + p_i \leq d_i^C$,
2. $p_i \leq p_j$ *and* $P_\gamma + p_i \geq d_j^C$,

we have $L_{max}(s) \leq L_{max}(\sigma')$ *and* $\sum\Delta(\sigma) = \sum\Delta(\sigma')$.

Proof. As in s jobs i and j are in EDD order, the L_{max} value of σ is not increased with respect to σ'. Besides, the two conditions state that the total disruption of σ is not higher than that of σ' which follows directly from the definition of $\sum\Delta$. □

In the remainder we focus on situations where sorting some new jobs in EDD order does not deteriorate the L_{max} and does not increase $\sum\Delta$.
Let $\sigma = \gamma\ i\ \alpha\ j\ \beta$ be a sequence where γ and β are sub-sequences of old and new

jobs, α is a sub-sequence of old jobs only, and $i, j \in N$. Let $\sigma' = \gamma\ j\ \alpha\ i\ \beta$ be the sequence obtained by swapping i and j in σ. We first state the following property which directly follows from the optimality of EDD for the $1||L_{max}$ problem.

Property 5.1.2. *Whenever $\alpha = \emptyset$ and $d_i \leq d_j$, we have $L_{max}(\sigma) \leq L_{max}(\sigma')$ and $\sum\Delta(\sigma) = \sum\Delta(\sigma')$.*

Now, let us assume that α is made up of at least one old job. We first state the following general property.

Proposition 5.1.1. *Whenever the following conditions hold:*

1. $\max(p_i - d_i; p_i + L_{max}(\alpha|0); p_i + P_\alpha + p_j - d_j) \leq \max(p_j - d_j; p_j + L_{max}(\alpha|0); p_j + P_\alpha + p_i - d_i)$, *with $P_\theta = \sum_{\ell\in\theta} p_\ell$ and $L_{max}(\alpha|0)$ the L_{max} value of α when starting at time 0,*
2. $\sum_{k\in\alpha} \Delta_k(\sigma) \leq \sum_{k\in\alpha} \Delta_k(\sigma')$,

schedule σ is always preferable to schedule σ'.

Proof. We first prove the first condition. We have:

$L_{max}(\sigma) \leq L_{max}(\sigma')$

$\Leftrightarrow \quad \max(L_{max}(\gamma|0); L_i(\sigma); L_{max}(\alpha|C_i(\sigma)); L_j(\sigma); L_{max}(\beta|C_j(\sigma)))$
$\leq \max(L_{max}(\gamma|0); L_j(\sigma'); L_{max}(\alpha|C_j(\sigma')); L_i(\sigma'); L_{max}(\beta|C_i(\sigma'))),$

$\Leftarrow \quad \max(L_i(\sigma); L_{max}(\alpha|C_i(\sigma)); L_j(\sigma)) \leq \max(L_j(\sigma'); L_{max}(\alpha|C_j(\sigma')); L_i(\sigma')),$

with $L_{max}(\theta|t)$ the L_{max} value of sub-sequence θ when starting at time t. Besides, we have $C_i(\sigma) = C_j(\sigma')$. By definition of the L_k's and the L_{max} it comes:

$\max(L_i(\sigma); L_{max}(\alpha|C_i(\sigma)); L_j(\sigma)) \leq \max(L_j(\sigma'); L_{max}(\alpha|C_j(\sigma')); L_i(\sigma')),$

$\Leftrightarrow \quad \max(P_\gamma + p_i - d_i; P_\gamma + p_i + L_{max}(\alpha|0); P_\gamma + p_i + P_\alpha + p_j - d_j)$
$\leq \max(P_\gamma + p_j - d_j; P_\gamma + p_j + L_{max}(\alpha|0); P_\gamma + p_j + P_\alpha + p_i - d_i),$

$\Leftrightarrow \quad \max(p_i - d_i; p_i + L_{max}(\alpha|0); p_i + P_\alpha + p_j - d_j)$
$\leq \max(p_j - d_j; p_j + L_{max}(\alpha|0); p_j + P_\alpha + p_i - d_i),$

with $P_\theta = \sum_{\ell\in\theta} p_\ell$.

The second condition directly follows from the definition of $\sum\Delta$ criterion and the fact that $C_j(\sigma) = C_i(\sigma')$. □

The general result stated in Proposition 5.1.1 can be further specified but we need first to focus on the analysis of the $\sum\Delta$ variation when we left- or right-timeshift α from date t (Figure 5.1). This analysis follows the same reasoning used in the timing problems described in Chapter 3.

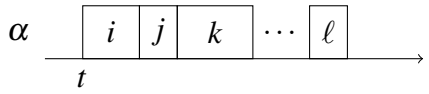


Figure 5.1 Sub-sequence of old jobs

Let us recall some notation used in chapter 3. Let B be the block of old jobs $j\in\alpha$. Let E_B, T_B and O_B be the sets of early, tardy and on-time jobs, with respect to d_j^C, of B when it starts at time t and let be $\delta_B^\Delta(t)=(|E_B|-|T_B|-|O_B|)$. If α is right-timeshifted so that it starts at time $(t+1)$, then the total disruption of jobs in α is decreased, or increased if negative, by $\delta_B^\Delta(t)$. The value of $\delta_B^\Delta(t)$ enables to derive information on how much α can be left/right time shifted. First, assume that $\delta_B^\Delta(t)<0$ which implies that by left-timeshifting α its total time disruption is decreased. Then, to compute a new starting time $t'\leq t$ such that $\delta_B^\Delta(t')\geq 0>\delta_B^\Delta(t'+1)$, we proceed as follows: let q_B^- be the $\delta_B^\Delta(t)-th$ smallest value $|C_j-d_j^C|$, $\forall j\in T_B\cup O_B$. Then, $t'=\max(0;t-q_B^-)$. We can define in a similar way q_B^+ when $\delta_B^\Delta(t)>0$ leading to $t'=t+q_B^+$.
Given this analysis, Proposition 5.1.1 enables to derive the following result.

Property 5.1.3. *Let $\sigma=\gamma\ i\ \alpha\ j\ \beta$ and $\sigma'=\gamma\ j\ \alpha\ i\ \beta$ be two sequences in which i and j are swapped and α is a sub-sequence of old jobs only. Assume that $d_i\leq d_j$. Whenever one of the following conditions holds:*

1. $p_i\leq p_j$, $\delta_B^\Delta(P_\gamma+p_j)<0$ *and* $q_B^-\geq(p_j-p_i)$,
2. $p_i>p_j$, $L_{max}(\alpha|0)>-d_i$, $\delta_B^\Delta(P_\gamma+p_j)>0$, *and* $q_B^+\geq(p_i-p_j)$,
3. $p_i>p_j$, $p_j+P_\alpha-d_j>L_{max}(\alpha|0)$, $\delta_B^\Delta(P_\gamma+p_j)>0$, *and* $q_B^+\geq(p_i-p_j)$,

schedule σ is always preferable to schedule σ'.

Proof. The result is proved by showing that Proposition 5.1.1 applies for each of the three conditions.
First, we focus on the first condition. Assuming that $d_i\leq d_j$ and $p_i\leq p_j$ it follows

that:

$$\begin{cases} p_i - d_i < p_j + P_\alpha + p_i - d_i, \\ p_i + P_\alpha + p_j - d_j \leq p_j + P_\alpha + p_i - d_i, \\ p_i + L_{max}(\alpha|0) \leq p_j + L_{max}(\alpha|0) \end{cases}$$

which implies that condition 1 of Proposition 5.1.1 is answered. Now, assuming that $\delta_B^\Delta(P_\gamma + p_j) < 0$, $q_B^- \geq (p_j - p_i)$ and $p_i \leq p_j$ imply that: in σ', decreasing the starting time α from $P_\gamma + p_j$ to $P_\gamma + p_i$ leads to decrease the total disruption induced by α. So, $\sum\Delta(\sigma) \leq \Delta_j(\sigma')$ and the second condition of Proposition 5.1.1 is answered.

Next, we focus on the second condition. Assuming that $d_i \leq d_j$, $p_i > p_j$ and $L_{max}(\alpha|0) > d_i$, it follows:

$$\begin{cases} p_i - d_i < p_j + P_\alpha + p_i - d_i, \\ p_j + L_{max}(\alpha|0) < p_i + L_{max}(\alpha|0), \\ p_i + L_{max}(\alpha|0) < p_i - d_i, \\ p_i + P_\alpha + p_j - d_j \leq p_j + P_\alpha + p_i - d_i, \end{cases}$$

which implies that condition 1 of Proposition 5.1.1 can be rewritten as:

$$\max(p_i - d_i; p_i + P_\alpha + p_j - d_j) \leq \max(p_j - d_j; p_j + P_\alpha + p_i - d_i),$$

and it necessarily holds. As $p_i > p_j$, from σ' to σ sequence α will be right-timeshifted by $(p_i - p_j)$ which results in a decrease of the total disruption as $\delta_B^\Delta(P_\gamma + p_j) > 0$ and $q_B^+ \geq (p_i - p_j)$, so $\sum\Delta(\sigma) \leq \sum\Delta(\sigma')$ and the second condition of Proposition 5.1.1 holds.

At last, we focus on the third condition. Assuming that $d_i \leq d_j$, $p_i > p_j$ and $p_j + P_\alpha - d_j > L_{max}(\alpha|0)$, it follows:

$$\begin{cases} p_i - d_i < p_j + P_\alpha + p_i - d_i, \\ p_j + L_{max}(\alpha|0) < p_i + L_{max}(\alpha|0), \\ p_i + L_{max}(\alpha|0) < p_i + P_\alpha + p_j - d_j, \\ p_i + P_\alpha + p_j - d_j \leq p_j + P_\alpha + p_i - d_i, \end{cases}$$

which, again, implies that condition 1 of Proposition 5.1.1 can be rewritten as follows, and it necessarily holds:

$$\max(p_i-d_i;p_i+P_\alpha+p_j-d_j)\leq\max(p_j-d_j;p_j+P_\alpha+p_i-d_i).$$

As previously, we can also derive from $\delta_B^\Delta(P_\gamma+p_j)>0$ and $q_B^+\geq(p_i-p_j)$, that the second condition of Proposition 5.1.1 holds. □

The results presented in this section can be used, for instance, to improve the search for an optimal solution in a search tree-based algorithm as it is shown in Section 5.3 for the branch and memorize algorithm.

5.2 Integer programming models

We present two integer programs (IP), a position-based and a time-indexed model, for solving rescheduling problems with an IP solver. Since many rescheduling problems are intractable, it is difficult to obtain optimal solutions in a useful amount of time from a standard IP solver but we use the solutions as a reference to compare the quality of the algorithms developed in the next sections.

A position-based integer program

Position-based integer programs use variables that associate jobs to sequence positions. To this purpose, for each job $j\in O\cup N$, we introduce the following binary variables, for any $1\leq k\leq n$:

$$x_{j,k}=\begin{cases}1, & \text{if job } j \text{ is scheduled in position } k,\\ 0, & \text{otherwise.}\end{cases}$$

Variable L_{max} refers to the maximum lateness of all jobs. We also introduce variables Δ_k for any position k, as the total disruption of completion times of the job j in position k. Notice that, $\forall k=1,\ldots,n$, such that $j\in N$ occupies position k, the solver will set $\Delta_k=0$. All variables are assumed to be integer. The model uses a so-called big-M constraint, which is known to make the model often inefficient the more M is large. Therefore we bound the value M by the total processing time P of all jobs.

We express the IP formulation with positional variables (IP_{pos}) as follows:

Minimize L_{max}

subject to

$$\sum_{j=1}^{n} x_{j,k} = 1 \qquad \forall k = 1,\ldots,n \quad (1)$$

$$\sum_{k=1}^{n} x_{j,k} = 1 \qquad \forall j = 1,\ldots,n \quad (2)$$

$$\sum_{\ell=1}^{k}\sum_{j=1}^{n} p_j x_{j,\ell} - \sum_{j=1}^{n} d_j x_{j,k} \leq L_{max} \qquad \forall k = 1,\ldots,n \quad (3)$$

$$\Delta_k \geq \sum_{j=1}^{n_O} C_j(\pi^*) x_{j,k} - \sum_{\ell=1}^{k}\sum_{j=1}^{n} p_j x_{j,\ell} - M\sum_{j=1}^{n_N} x_{j,k} \qquad \forall k = 1,\ldots,n \quad (4)$$

$$\Delta_k \geq \sum_{\ell=1}^{k}\sum_{j=1}^{n} p_j x_{j,\ell} - \sum_{j=1}^{n_O} C(\pi^*)_j x_{j,k} - M\sum_{j=1}^{n_N} x_{j,k} \qquad \forall k = 1,\ldots,n \quad (5)$$

$$\sum_{k=1}^{n} \Delta_k \leq Y \qquad (6)$$

$$\Delta_k \geq 0 \qquad \forall k = 1,\ldots,n \quad (7)$$

Constraints (1) guarantees that in any $k-th$ position there is exactly one job scheduled. Constraints (2) set that each job j is scheduled in exactly one position. Constraints (3) define the maximum lateness as the maximum difference, among all positions k, of the completion time of job at that position and its due date. Constraints (4) and (5) set the disruption of an old job scheduled in position k as a value greater than the absolute time deviation from initial completion time, or greater than 0 otherwise. Constraints (6) define the disruption constraint and (7) set the range of Δ_k variables. This model involves (n^2+n+1) variables and $(3n+2n_O+1)$ constraints.

A time-indexed integer program

Time-indexed integer programs use variables that associate jobs to starting or completion times. To this purpose, for each job $j \in O \cup N$, we introduce the following binary variables, for any $1 \leq t \leq T$, where $T = \sum_{j=1}^{n} p_j$:

$$x_{j,t} = \begin{cases} 1, & \text{if job } j \text{ starts at time } t, \\ 0, & \text{otherwise.} \end{cases}$$

Variable L_{max} refers to the maximum lateness of all jobs, variables Δ_j are introduced for each old job j. Finally, additional variables C_j are introduced for completion times of jobs.

The *IP* formulation with time-indexed variables (IP_{time}) follows:

Minimize L_{max}

subject to

$$\sum_{t=1}^{T-p_j+1} x_{j,t} = 1 \qquad \forall j = 1,\ldots,n \qquad (1)$$

$$\sum_{j=1}^{n}\sum_{s=t-p_j+1}^{t} x_{j,s} = 1 \qquad \forall t = 1,\ldots,T \qquad (2)$$

$$C_j = \sum_{t=1}^{T-p_j+1} (t-1+p_j)\, x_{j,t} \qquad \forall j = 1,\ldots,n \qquad (3)$$

$$L_{max} \geq C_j - d_j \qquad \forall j = 1,\ldots,n \qquad (4)$$

$$\Delta_j \geq C_j - C_j(\pi^*) \qquad \forall j = 1,\ldots,n_O \qquad (5)$$

$$\Delta_j \geq C_j(\pi^*) - C_j \qquad \forall j = 1,\ldots,n_O \qquad (6)$$

$$\sum_{j=1}^{n_O} \Delta_j \leq Y \qquad (7)$$

$$\Delta_j \geq 0 \qquad \forall j = 1,\ldots,n_O \qquad (8)$$

Constraints (1) and (2) again guarantee that each job is scheduled and that there is no overlap. Constraints (3) define the job completion times. Constraints (4) model the maximum lateness, while constraints (5) and (6) the disruption for the old jobs. Finally, constraint (7) defines the threshold on total disruption and (8) the range of job disruptions. This model involves $(nT+n+n_O+1)$ variables and $(3n+T+2n_O+1)$ constraints.

Alternatively, to reduce the number of variables, an earliest starting time est_j and a latest starting time lst_j can be defined for every old job. Recall the definition of release dates and deadlines introduced, for each job in a given sequence, in Section 3.3.1, which makes a schedule feasible. We will use the same intervals, but adjusted to refer to the starting time of each job. Let the the earliest starting time be defined as $est_j = \max(0, C_j(\pi^*) - Y - p_j)$ and the latest starting time as $lst_j = \min(T - p_j, C_j(\pi^*) + Y - p_j)$. For new jobs, $est_j = 0$ and $lst_j = T - p_j$.

Constraints (1), (2) and (3) can be rewritten into constraints (1b), (2b) and (3b) as follows:

$$\sum_{t=est_j}^{lst_j} x_{j,t} = 1 \qquad \forall j = 1,\ldots,n \quad (1b)$$

$$\sum_{j=1}^{n} \sum_{s=est_j+1}^{lst_j} x_{j,s} = 1 \qquad \forall t = 1,\ldots,T \quad (2b)$$

$$C_j = \sum_{t=est_j}^{lst_j} (t-1+p_j) x_{j,t} \quad \forall j = 1,\ldots,n \quad (3b)$$

5.3 A branch and memorize algorithm

In this section, we present an exact branch and memorize algorithm (BM) for the $1|no-idle, \sum_j \Delta_j \leq Y|L_{max}$ problem. The branch and memorize is a branch and bound-based algorithm that exploits memorization techniques to explore the search space in a more efficient way.

Branch and bound methods solve the problem to optimality, and are based on implicit enumeration. The basic idea is to construct and effectively explore a search tree using three components: a separation principle, a pruning principle and a search principle (see Morrison et al. (2016) for a review on utilization of these components in the literature). The space S of the solutions of the original problem $P(S)$ is divided into the subset S_0 of the solutions already considered and a list of other subsets $S_1, S_2, ..., S_q$, generated with the separation principle, such that $S = S_0 \cup S_1 \cup \cdots \cup S_q$. Each subset S_k identifies a candidate problem $P(S_k)$ which is obtained from $P(S)$ by restricting the space of admissible solutions to S_k. According to the search principle, the algorithm chooses a candidate problem and checks whether its resolution is necessary to find the optimum of $P(S)$. If the answer is negative, $P(S_k)$ can be eliminated, i.e. pruned from the search tree, and the set S_k is added to the set S_0. Otherwise, it is necessary to decompose S_k further and replace the problem $P(S_k)$ with a set of new candidate problems accordingly.

In general, the application of memorization to algorithms has the goal of speeding up their execution, often at the cost of increased memory consumption. In this sense, various algorithms in the literature use embedded memorization to efficiently explore the search space, even dynamic programming algorithms or meta-heuristics like tabu search.

The inclusion of memorization techniques in branch and bound algorithms is basically motivated by the fact that exact search tree algorithms repetitively explore subproblems, which are often similar, if not identical. The underlying idea, to avoid an inefficient exploration of the solution space, is to identify nodes of the tree that are build upon such similar subproblems and to determine in advance if the solution of some of them are dominated by others, and can be neglected for further search. Efficient applications of memorization to search tree algorithms for sequencing problems can be found in Szwarc et al. (2001), T'kindt et al. (2004), Sewell and Jacobson (2012), Shang et al. (2020).

An efficient branch and memorize has been designed for the rescheduling problem under consideration. In the section, we first present a local search with variable neighbourhoods that generates an upper bound solution for the problem. Then, we present the components of the branch and memorize algorithm for exploring and pruning the search tree. The memorization scheme together with dominance conditions to compare stored nodes are finally presented.

In Algorithm 8 we give the general structure of the branch and memorize algorithm, while we give more details about the local search algorithm LS for the upper bound, the search principle (function *search_principle()*), the separation principle (function *separation_principle(node)*) and the pruning principle (function *pruning_principle(node)*) in the next sections.

5.3.1 A local search heuristic for upper bounding

We use a local search-based heuristic (LS) with different neighbourhoods to compute an initial upper bound solution σ^{UB}. The neighbourhoods are defined by swap and insertion operators, which are shown to work well for the problem. The heuristic finds a first feasible solution σ_0 by using an EDD-EDD schedule, i.e. scheduling the set of old jobs first and that of new jobs next, each in EDD. This solution is clearly feasible, with $\sum\Delta = 0$, with respect to the constraints of the problem.

The heuristic makes use of the notion of critical jobs. Given a schedule σ, let $L = \{j \in O \cup N | L_j \geq c \times L_{max}(\sigma)\}$ be the set of critical jobs with $c = 0.9$ a factor experimentally determined. The critical jobs are those that we would like to left-shift in schedule σ to decrease the L_{max} value.

Algorithm 8 Algorithm BM for $1|no-idle,\sum\Delta\leq Y|L_{max}$

1: $\sigma^* \leftarrow$ upper bound solution obtained with LS
2: $UB := L_{max}(\sigma^*)$
3: $node_0 :=$ root node associated to empty schedule
4: add $node_0$ to $node\_list[0][0]$
5: $l = 0$
6: **while** $node\_list$ **not** empty **do**
7: $\quad l = search\_principle()$
8: $\quad node = node\_list[l][0]$
9: $\quad$ **if** $l = n$ **then**
10: $\quad\quad$ **if** $node \rightarrow L_{max} < UB$ **then**
11: $\quad\quad\quad UB := node \rightarrow L_{max}$
12: $\quad\quad\quad \sigma^* := node \rightarrow \sigma$
13: $\quad\quad$ **end if**
14: $\quad\quad$ delete $node$
15: $\quad$ **else**
16: $\quad\quad children\_list = separation(node)$
17: $\quad\quad$ add $children$ to $node\_list[l+1]$
18: $\quad$ **end if**
19: **end while**
20: **Return** σ^*

A local search starts with an initial solution that is iteratively improved by means of neighborhood operators. The local search we propose first computes σ_0. Given an incumbent solution of the current iteration, we apply the following neighborhood operators:

- $Swap(\sigma)$: from σ all possible swaps of two jobs j and k are performed and the best schedule, possibly σ, is returned,
- $Insert(\sigma)$: from σ, the non-critical job j with largest due date that precedes all the critical jobs is tested for insertion in all positions after the first critical job. The best obtained schedule, possibly σ, is returned,
- $Rotate(\sigma)$: from σ, all circular swaps of all triplets of jobs i, ℓ, k are considered. In σ let i, ℓ, k be respectively in positions pos_i, pos_ℓ, pos_k, with $pos_i < pos_\ell < pos_k$. If sequence $\sigma = \beta\ i\ \omega\ \ell\ \gamma\ k\ \alpha$, then the two sequences $\beta\ \ell\ \omega\ k\ \gamma\ i\ \alpha$ and $\beta\ k\ \omega\ i\ \gamma\ \ell\ \alpha$ are generated and evaluated. The best obtained schedule among all possible triplets, possibly σ, is returned.

When applying the above operators, a solution σ' improving σ is a solution such that:

1. $L_{max}(\sigma') < L_{max}(\sigma)$ and $\sum \Delta(\sigma') \leq Y$, or,
2. $L_{max}(\sigma') = L_{max}(\sigma)$ and $\sum \Delta(\sigma') \leq \sum \Delta(\sigma)$.

The local search algorithm is presented in Algorithm 9.

Algorithm 9 Algorithm LS for $1|\sum \Delta \leq Y|L_{max}$

```
Let be σ = σ_0,
while improving solutions are found and time limit is not exceeded do
    Let σ_i be the solution returned by Insert(σ)
    Let σ^s be the solution returned by Swap(σ_i)
    Let σ^r be the solution returned by Rotate(σ^s)
    Let be s = σ^r
end while
Return σ and L_max(σ)
```

5.3.2 Search strategy

The branch and memorize explores the solution space following a search principle, a separation and a pruning principle. We describe them in this section.

The search principle determines the level of the tree (level l in Algorithm 8) that is considered next for the evaluation of active nodes. We use and test the most common search principles, i.e. a *depth-first*, a *breadth-first* and a *best-first* strategy. With *depth-first* is meant the vertical exploration of the search tree in the sense that the first active nodes we evaluate are those corresponding to the subproblems of maximum job cardinality. The level l returned is in this case the deepest level of the tree with at least one active node. For instance, if there are active nodes with 1, 2 or 3 scheduled jobs, the *depth-first* strategy considers first evaluating nodes with 3 jobs. With *breadth-first* is meant, on the contrary, the horizontal exploration of the search tree in the sense that the first active nodes we evaluate are those corresponding to the subproblems of minimum job cardinality. The level l returned is in this case the flattest level of the tree with at least one active node. For instance, if there are active nodes with 1, 2 or 3 scheduled jobs, the *depth-first* strategy considers first evaluating nodes with 1 job. The third and last search principle, the *best-first* strategy, refers to the evaluation of active nodes in non-decreasing order of their associated lower bound, which, in a way, points to the most promising solution. For instance, if there are active nodes with lower bound value 12, 15, 18, the *best-first* strategy considers first evaluating nodes with lower bound 12.

The separation principle uses an assignment of jobs to positions, in a backward order, to generate partial solutions (the subproblems). Let σ_i be a partial schedule containing i jobs scheduled in the last i positions. Each node of the search tree corresponds to a partial solution σ_i and we use a backward branching scheme to generate nodes: for any given node σ_i, children nodes are obtained by assigning unscheduled jobs to position $(n-i)$. In this way, jobs are scheduled from the last position to the first one and since we deal with the no-idle case it is trivial to compute completion times backwards. Backward branching results to be much more efficient with respect to a forward branching, where solutions are build from the first to the last position. In fact, backward branching induces more information since both the objective function and the disruption constraint depend on the completion time of jobs and, often in practice, on the last scheduled jobs. Hence, the backward

branching enables to detect sooner if the total disruption constraint is violated or the L_{max} of a partial solution exceeds the best upper bound.

Algorithm 10 *separation_principle*$(node)$

```
1:  children_list := empty list
2:  σ_i := partial schedule of i jobs associated to input node node
3:  s̄ := list of jobs not in σ_i
4:  if σ_EDD := EDD(s̄) is a feasible schedule then
5:      new_node ← σ_EDD
6:      add new_node to children_list
7:  else
8:      for each job j ∈ s̄ do
9:          new_node ← (j) ∪ σ_i
10:         if pruning_principle(new_node) = true then
11:             delete new_node
12:         else
13:             add new_node to children_list
14:         end if
15:     end for
16: end if
17: Return children_list
```

To effectively search throughout the solution tree, at each node, the algorithm applies a pruning principle that is based on some conditions (see Algorithm 11). If any of the conditions is met, the node is pruned and is deleted from the list of nodes that need to be further explored. The conditions require the following information on the respective partial schedule σ_i with i jobs scheduled from the last to the $(n-i+1)-th$ position:

- the partial maximum lateness $L_{max}(\sigma_i)$;
- the partial disruption $\sum\Delta(\sigma_i)$;
- a lower bound for the maximum lateness, computed as $LB(\sigma_i) = max(L_{max}(EDD(\bar{s}), L_{max}(\sigma_i))$, where $L_{max}(EDD(\bar{s}_i))$ is the maximum lateness of the jobs not in σ_i scheduled at the beginning of the schedule in EDD order.

Given the best current known upper bound solution value UB, a node is pruned if any of the following conditions applies:

- $\sum \Delta(\sigma_i) > Y$,
- $LB(\sigma_i) \geq UB$,
- structural properties of Section 5.1 are violated,
- the current node is dominated by a node in DB (see dominance conditions from memorization, equation 5.3.1, in the next section).

Algorithm 11 *pruning_principle*$(node)$

```
1: UB := best current UB known for the problem
2: σ_i := partial schedule of i jobs associated to input node node
3: if node → ∑Δ > Y then
4:     return true
5: else if node → LB > UB then
6:     return true
7: else if jobs in σ_i violate structural properties then
8:     return true
9: else if node is dominated by a node in DB then
10:    return true
11: end if
12: return false
```

5.3.3 Memorization

The work of Shang et al. (2020) points out memorization as a way to speed up algorithms at the price of an increased space usage. The memorization of previously explored partial solutions is used in branching algorithms to avoid generation of solutions for the same subproblems or to check if permutations of the same subset of jobs are dominated or dominate some others. The authors define the branch and memorize paradigm after showing how memorization can be successfully applied to different scheduling problems. Referring to the definition introduced by Shang et al. (2020), we use *passive node memorization* to speed up the search. The idea is to compare a node with all other active and explored nodes that contain the same subset of jobs, and check if some dominance conditions are met. Given two subsequences σ_i and σ_i' containing two different permutations of the same subset of jobs, let $L_{max}(\sigma)$ be the partial maximum lateness and $\sum \Delta(\sigma)$ be the partial disruption of any

subsequence σ. Let $\bar{\sigma}$ also be the best subschedule containing all unscheduled jobs, and $t = \sum_{j\in\bar{\sigma}} p_j$.

Property 5.3.1. *Given two subsequences σ_i and σ_i' that contain the same subset of scheduled jobs, if*

$$L_{max}(\sigma_i) \leq L_{max}(\sigma_i') \text{ and } \sum\Delta(\sigma_i) \leq \sum\Delta(\sigma_i'), \tag{5.1}$$

σ_i dominates or is indifferent to σ_i'.

Property 5.3.1 states that the best schedule $\bar{\sigma}\sigma_i'$ contained in the subtree rooted at σ_i', cannot be strictly better than the best schedule contained in the subtree rooted at σ_i, since $\bar{\sigma}\sigma_i$ is not worse than $\bar{\sigma}\sigma_i'$.

The algorithm uses memorization to save previously explored nodes and to compare them with the others. Whenever two subschedules are found that contain the same subset of jobs and condition (5.1) is met, then σ_i' is cut and the respective branch is no further explored.

Despite Property 5.3.1 is a rather simple dominance condition, its efficient exploitation during the search requires a database implementation to store and retrieve active and explored nodes, which are relevant for comparison. The ways in which the database is designed and managed (e.g., database dimension, search, cleaning strategies) affect the tree search and make the algorithm different from a branch and bound algorithm.

5.4 Computational results

This section is devoted to show the results of computational experiments run to test the performances of the branch and memorize algorithm with respect to standard integer programming solvers run on the proposed integer programs. First, we show how we generated instances, then we present and discuss the results obtained.

5.4.1 Instances generation

Data have been randomly generated as follows. Instances are considered with $n = n_O + n_N \in \{10;20;30;40;50\}$ and $n_O \in \{0.25n;0.5n;0.75n\}$, leading to 15 couples

$(n;n_O)$. For each couple, the processing times of the old and new jobs are drawn at random following an uniform distribution between $U[10;100]$. Let $P = \sum_j p_j + \sum_j p_j$ be the sum of randomly generated processing times, then two classes of due dates have been considered.

- Class 1: $d_j, d_j \in U[0;P]$. This class of due dates corresponds to *easy* instances.
- Class 2: $d_j \in U[0;\tau P-1]$ and $d_j \in U[\tau P;P]$ with $\tau = 1-\frac{n_O}{n}$. This class of due dates corresponds to *hard* instances since to minimize the maximum lateness new jobs must be scheduled before old jobs inducing a potential conflict with the constraint on the total disruption.

The upper bound Y on the value of the total disruption is also computed depending on the class of due dates. For class 2 instances, ε is determined as the average between the disruption of a non-disrupted schedule and a schedule with the maximum possible disruption so that $Y = \frac{n_O \times \sum_{j \in N} p_j}{2}$.
For class 1 instances, an additional factor of $1/2$ is multiplied, which considers the fact that in an EDD optimal schedule, each old job would be, on the average, followed by just a half of new jobs. For each couple $(n;n_O)$, 20 instances are randomly generated, so leading to 300 instances per class.

The database for memorization is implemented using a hash table with $500n$ entries. When the database is full, a "Least Used, First Out" clean strategy is used (see Shang et al. (2020) for further details).

Tests are run on an i5-8500 3 GHz CPU with 16 Gb of RAM and CPLEX (version 12.9) as MILP solver. A time limit of 900 CPU seconds is given to the exact algorithm and 60 CPU seconds to the upper bound heuristic.

All tables show the results for all or a subset of instances. The first column provides the class of instances, column n provides the total number of jobs, column n_O the number of old jobs. Then, tables show some of the following results. The column denoted with (*#opt*) gives the number of instances solved to optimality before the time limit, (*Avg %g*) gives the average gap in percentage to the optimal or best known solution, (*Avg t(s)*) the average CPU time in seconds, and, whenever relevant, (*Avg #n*) gives the average number of nodes explored.

5.4.2 IP models comparison

The positional and time-indexed IP formulations are compared in Table 5.1. Each model, before solving the IP, checks if the EDD schedule is feasible. If so, it returns the corresponding L_{max} value and stops there.

Since both lower bounds returned by the solver and the analytical lower bound (EDD) do often perform poorly, we evaluate the relative quality of the solutions, with respect to each other, as the relative gap between the best solution found. More precisely, the relative percentage gap is computed as follows. If I is a set of models that are run for a test, and each model i returns a solution σ^i, the relative percentage gap of each model to the best solution known is computed as

$$\frac{L_{max}(\sigma^i) - \min_{i \in I}(L_{max}(\sigma^i))}{\min_{i \in I}(L_{max}(\sigma^i))} \cdot 100$$

Table 5.1 the IP model comparison. For every configuration of n and n_O, and for every model, it indicates how many of the 20 sample instances were solved to optimality, average and maximum percentage gap from the best found solution, average and maximum time in CPU seconds, and average and maximum number of explored nodes. Results are first showed for instances from class 1 of the generation scheme, followed by instances from class 2.

The time-indexed model is very time expensive due to the large increase in the number of variables when n increase, and the results clearly show that the position-based formulation is more efficient in solving the problem. Model IP_{pos} can find most of optimal solutions up to 30 jobs. Model IP_{time} has already issues in finding the optimum within the time limit for 20 jobs. The results clearly show that the positional formulation compares favorably with respect of the time-based model. Therefore, we use the results given by IP_{pos} as a reference for evaluating the efficiency of the branch and memorize algorithm.

5.4.3 Results of the branch and memorize algorithm

In Table 5.2, the quality of the upper bound is evaluated. The upper bound solution σ^{UB} found by the LS heuristic is compared with solutions obtained from the exact algorithms on instances with up to 40 jobs. The column #*inst* gives the number of

Table 5.1 IP comparison

			(IP_{pos})						(IP_{time})					
	n	n_O	# opt	Avg %g	Max %g	Avg t(s)	Max t(s)	Avg #n	Max #n	# opt	Avg %g	Max %g	Avg t(s)	Max t(s)	Avg #n	Max #n
Class 1	10	2	20	0.0	0.0	0.1	0.3	620.0	3162	20	0.0	0.0	0.0	0.0	0.1	1
		5	20	0.0	0.0	0.2	0.7	760.6	2289	20	0.0	0.0	0.3	1.9	6.0	61
		15	20	0.0	0.0	0.1	0.3	756.0	2158	20	0.0	0.0	0.6	6.5	6.3	68
	20	5	20	0.0	0.0	10.2	42.1	29280.3	119923	6	76.9	332.9	810.5	900.0	435.0	976
		10	20	0.0	0.0	153.1	900.0	337417.1	2283206	3	113.2	488.2	842.3	900.0	593.2	1596
		15	20	0.0	0.0	305.2	900.0	350112.5	1179881	3	112.4	280.5	853.1	900.7	403.5	1102
Class 2	10	2	20	0.0	0.0	0.0	0.1	265.9	1028	20	0.0	0.0	0.3	1.6	4.2	15
		5	20	0.0	0.0	0.1	0.3	958.4	3066	20	0.0	0.0	4.2	18.8	57.9	257
		15	20	0.0	0.0	0.2	0.4	1291.5	2324	20	0.0	0.0	7.8	37.2	39.9	212
	20	5	20	0.0	0.0	4.6	14.6	10754.4	41108	0	51.3	190.5	900.0	900.0	454.9	1266
		10	20	0.0	0.0	56.8	888.7	145436.5	2429440	0	78.3	184.3	900.0	900.1	471.4	1108
		15	20	0.0	0.0	76.1	542.1	110665.0	760547	0	103.0	190.3	900.0	900.5	457.6	640

Time limit: 900 CPU seconds

instances for which the optimal solution σ^* is known and the average gap computed. Here, the percentage gap is computed, for each instance, as

$$\frac{L_{max}(\sigma^{UB}) - L_{max}(\sigma^*)}{L_{max}(\sigma^{UB})} \cdot 100$$

These results show that the LS heuristic is efficient as it returns in few milliseconds a solution often close to the optimal one. Swap and insertion operators are shown to be effective in particular for instances from class 2.

Some preliminary experiments were run to see the impact of the structural properties and the memorization on the exact algorithm. Table 5.3 shows the gain in both computational time and number of explored nodes when using the structural properties known for the problem (see Section 5.1) and the memorization technique presented in Section 5.3.3. Only instances with 20 jobs in size are considered.

In this table, the three search strategies are considered (*best-first*, *breadth-first*, *depth-first*) and for each one four different versions are tested:

I the branch and bound scheme, with neither properties nor memorization techniques included;

II the branch and bound scheme with properties included;

III the branch and memorize scheme with only memorization;

IV the branch and memorize with both properties and memorization.

From the table, we see that the gain in exploiting both the structural properties and memorization is clear. The result is mainly evident by looking at the average number of nodes, which decreases at least of one order of magnitude when adding one of the two techniques. Without using them, some instances cannot be solved because of an excessive usage of the RAM.
Note that when neither properties nor memorization are used on instances with only 5 old jobs (first row of each class), the number of nodes explored is the same for all search strategies. This can be explained by the fact that when there are few old jobs and many new jobs, there are many solutions that differ in the way new jobs are scheduled, but are very similar in terms of objective cost. Such solutions appear in

Table 5.2 LS heuristic with respect to optimal solutions

				LS		
	n	n_O	# inst	# opt	Avg t(s)	Avg %g
Class 1	10	2	20	18	0	1.8
		5	20	17	0	1.2
		7	20	19	0	0.3
	20	5	20	20	0	0
		10	20	19	0	0.1
		15	20	12	0	2.3
	30	7	20	19	0	0.9
		15	20	12	0	6
		22	20	10	0	14.9
	40	10	20	17	0	1.1
		20	20	6	0.1	14.4
		30	19	5	0.1	25.9
Class 2	10	2	20	16	0	2.1
		5	20	15	0	2.1
		7	20	17	0	1.3
	20	5	20	20	0	0
		10	20	12	0	2.9
		15	20	7	0	5.1
	30	7	20	16	0	1.6
		15	20	7	0	2.8
		22	20	8	0	2.1
	40	10	20	14	0	2.5
		20	15	1	0.1	8.2
		30	2	1	0.1	3.1

Table 5.3 Increased efficiency through properties and memorization on instances solved to optimality

			(BM_{best})		(BM_{depth})		$(BM_{breadth})$	
		n_O	Avg t(s)	Avg #n	Avg t(s)	Avg #n	Avg t(s)	Avg #n
Class 1	I	5	0.8	233744.2	0.8	233744.2	0.8	233744.2
		10	0.3	64719.3	0.3	64719.9	0.3	64739.4
		15	0.3	67092.0	0.3	68124.6	0.7	208747.6
	II	5	0.0	358.0	0.0	358.0	0.0	358.0
		10	0.0	7822.1	0.0	7822.4	0.0	7830.9
		15	0.2	37008.2	0.2	37495.2	0.2	39577.8
	III	5	0.1	170.7	0.1	120.7	0.1	145.1
		10	0.1	561.9	0.1	882.5	0.1	566.3
		15	0.1	995.5	0.1	1704.3	0.1	1164.3
	IV	5	0.1	143.8	0.1	120.7	0.1	109.6
		10	0.1	560.8	0.1	882.3	0.1	473.8
		15	0.1	874.9	0.1	1700.6	0.1	814.7
Class 2	I	5	0.0	3721.4	0.0	3721.4	0.0	3721.4
		10	0.1	10747.6	0.1	12224.5	19.4	6113046.3*
		15	1.0	122992.6	0.7	126828.6	6.3	2208351.3
	II	5	0.0	188.9	0.0	188.9	0.0	188.9
		10	0.0	2776.7	0.0	2962.2	0.2	37794.8
		15	0.3	41551.7	0.2	43238.1	0.5	97816.9
	III	5	0.1	122.3	0.1	165.3	0.1	155.9
		10	0.1	789.5	0.1	1263.4	0.2	7239.1
		15	0.1	4021.6	0.1	8200.8	0.1	5592.1
	IV	5	0.1	120.2	0.1	165.4	0.1	124.0
		10	0.1	611.8	0.1	1260.2	0.1	2239.6
		15	0.1	3069.5	0.1	8189.8	0.1	4229.6

(*) : RAM exceeded

Instances size: 20 jobs
I : no properties, no memorization
II : only properties
III : only memorization
IV : both properties and memorization

different branches of the search tree, and optimality is guaranteed by exploring all of these branches, i.e. independently from the search strategy.

Let us turn to the comparison between the position-based IP formulation and the branch and memorize algorithm. The algorithms are first compared with respect to the solution gaps in terms of relative deviation from the best solution found in table 5.4. Then, the performances in terms of number of optimal solutions found, time and number of explored nodes are showed in table 5.5 for class 1 and table 5.6 for class 2.

The branch and memorize algorithm strongly outperforms CPLEX whatever is the search principle. For larger 40−jobs and 50−jobs instances, CPLEX has even issues in finding any feasible solution ((-) if none is found or (*) if some of the 20 sample instances were unsolved to optimality) within the time limit of 900 CPU seconds.

Branch and memorize solves all 30−jobs instances to optimality. Most of the 40−jobs instances are also optimally solved and still many 50−jobs instances' optima are found. The *best-first* BM is finally resulting in being the most efficient on class 2, while the *breadth-first* on instances from class 1. On the respective classes, they find more optima and explore in average fewer nodes.

The hardest instances are those where there is a majority of old jobs. There, the fewest number of optima are found and computing times increase.

Table 5.4 Solution quality comparison between BM and IP_{pos}

			(IP_{pos})	(BM_{best})	(BM_{depth})	$(BM_{breadth})$
	n	n_O	Avg %g	Avg %g	Avg %g	Avg %g
Class 1	10	2	0.0	0.0	0.0	0.0
		5	0.0	0.0	0.0	0.0
		7	0.0	0.0	0.0	0.0
	20	5	0.0	0.0	0.0	0.0
		10	0.0	0.0	0.0	0.0
		15	0.0	0.0	0.0	0.0
	30	7	0.6	0.0	0.0	0.0
		15	7.2	0.0	0.0	0.0
		22	85.6	0.0	0.0	0.0
	40	10	12.9	0.0	0.0	0.0
		20	39.3	0.7	0.0	0.0
		30	-	3.9	0.4	2.2
	50	22	16.8	0.0	0.0	0.0
		25	121.7	0.1	1.7	9.3
		37	-	0.7	1.8	0.6
Class 2	10	2	0.0	0.0	0.0	0.0
		5	0.0	0.0	0.0	0.0
		7	0.0	0.0	0.0	0.0
	20	5	0.0	0.0	0.0	0.0
		10	0.0	0.0	0.0	0.0
		15	0.0	0.0	0.0	0.0
	30	7	0.9	0.0	0.0	0.0
		15	1.1	0.0	0.0	0.0
		22	7.7	0.0	0.0	0.0
	40	10	2.0	0.0	0.0	0.0
		20	10.2	2.8	0.4	7.9
		30	38.6	0.0	0.3	0.0
	50	12	9.7	0.0	0.0	2.0
		25	30.7	3.6	0.8	4.8
		37	55.0	0.0	0.0	0.0

Time limit: 900 CPU seconds

Table 5.5 Comparison between *BM* and IP_{pos}- Class I.

			(IP_{pos})			(BM_{best})			(BM_{depth})			$(BM_{breadth})$	
		#	Avg	Avg	#	Avg	Avg	#	Avg	Avg	#	Avg	Avg
	n	n_O	opt	t(s)	#n	opt	t(s)	#n	opt	t(s)	#n	opt	t(s)	#n
Class 1	10	2	20	0.1	620.0	20	0.1	8.4	20	0.1	8.4	20	0.1	9.2
		5	20	0.2	760.7	20	0.1	16.6	20	0.1	20.4	20	0.1	17.7
		7	20	0.1	756.0	20	0.1	15.6	20	0.1	17.8	20	0.1	17.2
	20	5	20	10.2	29280.3	20	0.1	143.8	20	0.1	120.7	20	0.1	109.6
		10	19	153.1	337417.1	20	0.1	560.8	20	0.1	882.3	20	0.1	473.8
		15	17	305.2	350112.5	20	0.1	874.9	20	0.1	1700.6	20	0.1	814.7
	30	7	11	506.4	461143.0	20	0.3	15684.5	20	0.3	17660.3	20	0.2	10323.1
		15	7	672.7	408306.6	20	0.1	4089.7	20	0.2	11673.3	20	0.2	8268.7
		22	4	720.1	414429.6	20	0.7	24315.4	20	2.4	118674.3	20	0.4	24798.0
		10	2	824.5	332100.4	20	14.9	163565.5	20	37.4	479769.1	20	5.9	129317.3
	40	20	0	900.0	402935.4*	19	51.3	163427.8	19	98.6	1158494.3	20	5.8	208498.3
		30	0	-	-	14	236.0	506936.9	18	273.3	3219599.4	19	96.5	781709.6
		12	3	804.3	242369.2	15	251.7	676600.2	16	234.5	2113574.3	18	98.5	535460.5
	50	25	0	674.5	1138859.6*	13	392.5	779193.9	11	493.1	3867649.3	13	396.2	1925002.9
		37	0	-	-	7	642.8	1257802.3	5	678.7	5248235.2	9	614.5	2441605.7

Time limit: 900 CPU seconds

Table 5.6 Comparison between *BM* and IP_{pos}- Class II.

			(IP_{pos})			(BM_{best})			(BM_{depth})			$(BM_{breadth})$		
	n	n_O	# opt	Avg t(s)	Avg #n	# opt	Avg t(s)	Avg #n	# opt	Avg t(s)	Avg #n	# opt	Avg t(s)	Avg #n
Class 2	10	2	20	0.0	265.9	20	0.1	11.3	20	0.1	10.8	20	0.1	12.6
		5	20	0.1	958.4	20	0.1	22.1	20	0.1	25.5	20	0.1	33.7
		7	20	0.2	1291.5	20	0.1	25.6	20	0.1	28.6	20	0.1	31.2
	20	5	20	4.6	10754.4	20	0.1	120.2	20	0.1	165.4	20	0.1	124.0
		10	20	56.8	145436.5	20	0.1	611.8	20	0.1	1260.2	20	0.1	2239.6
		15	20	76.1	110665.0	20	0.1	3069.5	20	0.1	8189.8	20	0.1	4229.6
	30	7	15	291.4	281482.1	20	0.2	4651.2	20	0.3	14418.7	20	5.1	69545.7
		15	10	582.7	492344.3	20	0.7	19916.2	20	5.1	120753.3	20	2.1	72005.2
		22	3	837.1	467099.5	20	7.0	128608.7	20	80.0	883246.9	20	4.8	172594.3
		10	7	677.0	237461.5	20	0.4	16586.9	20	2.6	96407.6	20	1.6	78216.0
	40	20	0	900.0	261904.3	15	292.1	310465.2	11	567.4	2536484.0	10	537.8	2280926.0
		30	0	900.0	973174.0*	2	831.1	620731.5	0	900.0	4469632.6	2	817.0	2712933.1
		12	3	812.2	139588.5	20	28.9	149615.5	19	87.4	734230.2	14	301.5	1449075.4
	50	25	0	900.0	158856.6	4	768.4	391379.4	0	900.0	2630738.8	1	856.4	2326898.4
		37	0	900.0	1936204.9*	0	900.0	322960.6	0	900.0	3658656.7	0	900.0	2261642.6

Time limit: 900 CPU seconds

Chapter 6

Conclusions and future research

In this dissertation, rescheduling problems for new orders were considered from theoretical and algorithmic point of view. Rescheduling problems can provide methods for responding quickly and effectively to the occurrence of unexpected events and there is need of the development and evaluation of methods for dealing with such problems. To this end, the dissertation gives the theoretical background of rescheduling problems to address this class of problems that can be taken as a reference point.

The work contributes to this purpose by discussing underlying structures of optimal solutions, which are used to extract properties and dominance conditions that can be used to solve the problem, analyzing their computational complexity and classifying problems according to relevant differences in their structure. In this sense, the analysis outlines the typical theoretical structures of rescheduling problems for new orders and the difficulty of solving them according to current complexity theory. The development of the dissertation aims at showing the characteristics and critical issues that need to be taken into account when it is necessary to solve a rescheduling problem, and at the same time understand their causes and effects.

Another result that follows the discussion around problem structures is the formalization, for the first time, of timing problems in rescheduling. The fact that there are optimal solutions that require the presence of idle time means that the permutation problem is not sufficient to solve the problem, adding an element of complexity to the resolution of these rescheduling problems. It is shown when the timing problem is relevant and how it affects the complexity of the problems. In

addition to this, the study of the timing problems showed that there are several analogies with other scheduling problems that can be exploited for solving them.

Finally, the dissertation shows that it is possible to use the outlined models and structures to design exact and approximation algorithms, which include exact polynomial algorithms for timing problems, exact and approximation dynamic programming algorithms for a weakly NP-hard problem, a branch and memorize and a local search-based heuristic algorithm for a strongly NP-hard problem.

In summary, the dissertation provides a reference framework for understanding and solving rescheduling problems in the most general way possible. However, the discussion leaves the door open for further investigation on several points. Therefore, in the following section, we conclude with some reflections on the directions for future work, based on the results obtained.

Future work

Problem generalization

We studied rescheduling problems for new orders under some assumptions that are present in most of the current literature. The problem description of Section 1.1 introduces a well-defined set of problems with certain characteristics that can be relaxed. In particular, it considers:

1. the optimality of the initial schedule,
2. a function of the absolute time deviation of jobs as a disruption cost,
3. regular functions as objectives to be minimized.

Each of these settings can be relaxed, leading to new problem formulations. Further research in these sense would help to extend the knowledge of rescheduling methods and generalize the idea of rescheduling for new orders.

For instance, by dropping the optimality assumption of the initial schedule, we enter the category of rescheduling of multiple orders (see Section 1.4), which generalize the problem. Given the intractability of most of the problems considered here, generalizing the definition would make the problems even harder. Studying

these problems would help to understand if there are structural results that can be exploited.

Also, throughout this work, attention has been limited to considering the absolute time deviation as a disruption cost in the system. There are few variations that would be worth investigating. First, we recall that there are few works in the literature that consider different disruption measures, e.g., the deviation in terms of positions in the schedule, or the change in resource allocation in the multi-machine setting. In addition, it would be interesting to investigate other cases. For example, how the models and solutions change when we consider only delays as a source of disruption. In this case, the problem would be simplified (e.g., by the absence of idle time). Another case worth studying is the problem with weighted time deviations. For instance, a job causes a larger disruption in a production facility if the rearrangement required to move it is more complex and involves other resources. The weight can also represent the flexibility of different customers or the number of passengers in train rescheduling. Again, the introduction of weighted deviations is a generalization, and problems are expected to become more difficult to solve. However, this leads to a number of special cases that are still open.

The third extension that we mention is the generalization to non-regular objective functions. In this context, many considerations done on idle time insertion and solution structures (Chapter 2 and 3), change. The interaction between such objectives and the absolute time deviations should be further investigated.

Any extension in this sense, would enrich the knowledge about rescheduling for new orders.

Computational complexity of open problems

The complexity analysis leaves two problems open for weakly or strongly NP-hardness. They are the rescheduling problem to minimize the total tardiness subject to a maximum time disruption constraint and that to minimize the total weighted completion time subject to a total time disruption constraint. A first point for future work directions requires further investigating these two problems.

Extension and improvements of the branch and memorize algorithm

Then, work should be devoted in adapting the branch and memorize algorithm for other hard rescheduling problems, both for the case of preventing and allowing the insertion of idle times on the machine.

For the case when idle times should be prevented, the generation of nodes in the search tree remains the same, as do the dominance conditions used in memorization (except for the objective function and the disruption constraint, which must be adapted for each specific case). What changes are the structural properties. Given the impact of the dominance conditions given by the structural properties (see Table 5.3), for efficient solution, structural properties must be formulated and tested for the other objective functions.

For the other case, where the insertion of idle times is considered beneficial to the production system, the node generation scheme can remain the same, but needs some adjustments. For example, given a partial compact solution at a given node, the timing algorithm can be considered to evaluate whether there is a timing of the jobs of the partial solution such that the sequence is feasible with respect to disruption. Thus, the pruning condition regarding the disruption constraint can be applied only after the evaluation of the idle time insertion. Also, the memorization dominance condition must be adapted to include different start times of different jobs given the same subset of jobs at two different nodes in the tree. So, additional memorization techniques should be implemented and tested.

Furthermore, from a more general perspective, the algorithm lacks efficient lower bounds that can be exploited in the solution tree, so improvements on this side would indeed make the exploration of the search tree more efficient.

Evaluation of the impact of idle times

Another relevant point concerns the timing problem that is underlying to some rescheduling problems and that leaves open the question of its practical impact. Idle time is often seen as a penalization in production planning, however it should be evaluated in a more accurate way if the improvements in terms of the objective function can be high enough to justify the occurrence of idle times on the machine.

Given this open question, we give just few remarks extracted from preliminary results. Instances generated randomly as described in Chapter 5 were also tested allowing the insertion of idle times. However, none of the tested instances resulted in an optimal solution with inserted idle times. This result suggests that the inclusion of idle time is only really helpful in achieving an optimal solution in the case of instances with a very particular structure. A structural pattern that requires the insertion of idle times is the one given in the example illustrated in Figure 2.6 of Chapter 2. We provide the example of a schema that replicates the same pattern on a larger instance in the following. The pattern is repeated n times and is made up of three old job and one new job. For each $i-th$ repetition, for any $i = 1,\ldots,n$, we have processing times

$$p_{3(i-1)+1} = 2i$$

$$p_{3(i-1)+2} = p_{3(i-1)+3} = p_{3n+i} = 1,$$

due dates

$$d_{3(i-1)+1} = M + 3(i-1) + 1$$

$$d_{3(i-1)+2} = M + 3(i-1) + 2$$

$$d_{3(i-1)+3} = M + 3(i-1) + 3$$

$$d_{3n+i} = 0,$$

with M a sufficiently large value, e.g. $M > P$, such that old jobs are never responsible for giving the maximum lateness, and a constraint on the total disruption of

$$Y = \sum_{i=1}^{n} p_{3(i-1)+1} + 2.$$

An optimal idle schedule can be sketched, for instance, as follows:

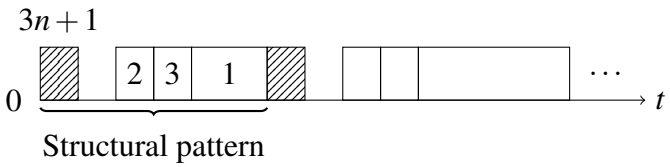


Figure 6.1 Sample pattern repetition for an instance that requires idle time insertion.

Modifying the branch and memorize algorithm to integrate the timing algorithm at a node, whenever relevant, would be the first step for evaluating through experiments the impact of allowing idle times on these or other instances.

Another way of considering the timing issue would be to provide bounds on the amount of idle time required in optimal solution or on the worst case loss in the objective function when considering compact solutions. As opposed to the previous point, this would well serve to the purpose of understanding the theoretical and worst case impact of neglecting idle time.

Exploration of the full Pareto frontier

We considered solving the rescheduling problem, which is intrinsically a bi-objective problem, with a so-called *a priori* method (see T'kindt and Billaut (2006)). In other words, we assumed that the decision maker makes an a priori decision on the disruption objective, and solves the remaining single objective problem to optimality. However, we do not investigate the shape of the full Pareto frontier, which would certainly provide more information about the relationship between the two objectives. In fact, the following questions are still open: How much does f increase when the value Y decreases by a certain amount? For how many different values/range of Y are there efficient solutions? All of these points should be studied further in order to have a more precise characterization of the rescheduling problem in terms of its bi-objective nature. Identifying methods to compute the Pareto frontier would allow the application of *a posteriori* methods that allow the decision maker to base his/her decision on the observed behaviour of the two objectives.

Heuristics and multi-machines settings

The literature review in Section 1.4 revealed the lack of works devoted to test heuristic approaches to solve large size instances. The exact algorithm proposed in Section 5.3 is able to solve only small instances therefore, further directions of work should consider developing fast approximation and heuristic algorithms for the hard problems. Also, another lack in the literature worth to be mentioned is need of extending rescheduling formulations for multiple machine environments, to which just few works in the literature are dedicated.

Bibliography

Agnetis, A., Billaut, J.-C., Gawiejnowicz, S., Pacciarelli, D., and Soukhal, A. (2014). Multiagent scheduling. Springer Berlin Heidelberg.

Ausiello, G., Protasi, M., Marchetti-Spaccamela, A., Gambosi, G., Crescenzi, P., and Kann, V. (1999). Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. Springer-Verlag.

Aytug, H., Lawley, M. A., McKay, K., Mohan, S., and Uzsoy, R. (2005). Executing production schedules in the face of uncertainties: A review and some future directions. European Journal of Operational Research, 161(1):86–110.

Azizoğlu, M. and Alagöz, O. (2005). Parallel-machine rescheduling with machine disruptions. IIE Transactions, 37(12):1113–1118.

Bagchi, U., Sullivan, R. S., and Chang, Y.-L. (1986). Minimizing mean absolute deviation of completion times about a common due date. Naval research logistics quarterly, 33(2):227–240.

Baker, K. R. (1974). Introduction to Sequencing and Scheduling. Wiley.

Baker, K. R. and Scudder, G. D. (1990). Sequencing with earliness and tardiness penalties: A review. Operations Research, 38(1):22–36.

Boeckmann, J., Thielen, C., and Pferschy, U. (2023). Approximating single- and multi-objective nonlinear sum and product knapsack problems. Discrete Optimization, 48:100771.

Brucker, P. (2007). Scheduling Algorithms. Springer Berlin Heidelberg.

Chrétienne, P. and Sourd, F. (2003). PERT scheduling with convex cost functions. Theoretical Computer Science, 292(1):145–164.

Cook, S. (1971). The complexity of theorem proving procedure. In Proc. 3rd Symp. on Theory of Computing, pages 151–158.

Croce, F. D. and Trubian, M. (2002). Optimal idle time insertion in early-tardy parallel machines scheduling with precedence constraints. Production Planning & Control, 13(2):133–142.

Davis, J. S. and Kanet, J. J. (1993). Single-machine scheduling with early and tardy completion costs. Naval Research Logistics (NRL), 40(1):85–101.

Du, J. and Leung, J. Y.-T. (1990). Minimizing total tardiness on one machine is np-hard. Mathematics of Operations Research, 15(3):483–495.

Fang, K., Luo, W., Pinedo, M. L., Jin, M., and Lu, L. (2023). Rescheduling for new orders on a single machine with rejection. Journal of the Operational Research Society.

Gao, K., Suganthan, P., Chua, T., Chong, C., Cai, T., and Pan, Q. (2015). A two-stage artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job insertion. Expert Systems with Applications, 42(21):7652–7663.

Garey, M. R. and Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman San Francisco.

Garey, M. R., Tarjan, R. E., and Wilfong, G. T. (1988). One-processor scheduling with symmetric earliness and tardiness penalties. Mathematics of Operations Research, 13(2):330–348.

Graham, R., Lawler, E., Lenstra, J., and Rinnooy Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics, 5:287–326.

Guo, Y., Huang, M., Wang, Q., and Leon, V. J. (2021). Single-machine rework rescheduling to minimize total waiting time with fixed sequence of jobs and release times. IEEE Access, 9:1205–1218.

Hall, N., Liu, Z., and Potts, C. (2007). Rescheduling for multiple new orders. INFORMS Journal on Computing, 19(4):633–645.

Hall, N. and Potts, C. (2004). Rescheduling for new orders. Operations Research, 52(3):440–453.

Hall, N. and Potts, C. (2010). Rescheduling for job unavailability. Operations Research, 58(3):746–755.

Hoogeveen, H., Lentè, C., and T'kindt, V. (2012). Rescheduling for new orders on a single machine with setup times. European Journal of Operational Research, 223:40–46.

Karp, R. (1972). Reducibility among combinatorial problems. Complexity of Computer Computation, pages 85–104.

Katragjini, K., Vallada, E., and Ruiz, R. (2013). Flow shop rescheduling under different types of disruption. International Journal of Production Research, 51(3):780–797.

Kellerer, H., Mansini, R., Pferschy, U., and Speranza, M. G. (2003). An efficient fully polynomial approximation scheme for the Subset-Sum Problem. Journal of Computer and System Sciences, 66(2):349–370.

Kellerer, H., Pferschy, U., and Pisinger, D. (2004). Knapsack Problems. Springer.

Lawler, E., Lenstra, J., Rinnooy Kan, A., and Shmoys, D. (1993). Sequencing and scheduling: Algorithms and complexity. Handbooks in operations research and management science, 4:445–522.

Lee, C.-Y. (1996). Machine scheduling with an availability constraint. Journal of Global Optimization, 9:395–416.

Lendl, S., Pferschy, U., and Rener, E. (2023). Rescheduling with new orders under bounded disruption. Manuscript accepted for publication, subject to minor revision.

Liu, L. (2019). Outsourcing and rescheduling for a two-machine flow shop with the disruption of new arriving jobs: A hybrid variable neighborhood search algorithm. Computers & Industrial Engineering, 130:198–221.

Liu, L. and Zhou, H. (2015). single-machine rescheduling with deterioration and learning effects against the maximum sequence disruption. International Journal of Systems Science, 46(14):2640–2658.

Liu, Z., Lu, L., and Qi, X. (2018). Cost allocation in rescheduling with machine unavailable period. European Journal of Operational Research, 266(1):16–28.

Liu, Z. and Ro, Y. (2014). Rescheduling for machine disruption to minimize makespan and maximum lateness. Journal of Scheduling, 17(4):339–352.

Luo, W., Jin, M., Su, B., and Lin, G. (2020). An approximation scheme for rejection-allowed single-machine rescheduling. Computers & Industrial Engineering, 146:106574.

Luo, W., Luo, T., Goebel, R., and Lin, G. (2018). Rescheduling due to machine disruption to minimize the total weighted completion time. Journal of Scheduling, 21:565–578.

Morrison, D. R., Jacobson, S. H., Sauppe, J. J., and Sewell, E. C. (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. Discrete Optimization, 19:79–102.

Nicosia, G., Pacifici, A., Pferschy, U., Resch, J., and Righini, G. (2021). Optimally rescheduling jobs with a Last-In-First-Out buffer. Journal of Scheduling, 24:663–680.

Özlen, M. and Azizoğlu, M. (2011). Rescheduling unrelated parallel machines with total flow time and total disruption cost criteria. Journal of the Operational Research Society, 62(1):152–164.

Pai, C. M., Liu, Y. L., and Hsu, C. J. (2014). Single-machine rescheduling of new orders with learning and deterioration eeffects consideration. Applied Mechanics and Materials, 565:198–204.

Peng, K. (2019). A multi-start variable neighbourhood descent algorithm for hybrid flowshop rescheduling. Swarm and Evolutionary Computation, 45:92–112.

Pferschy, U., Resch, J., and Righini, G. (2022). Algorithms for rescheduling jobs with a LIFO buffer to minimize the weighted number of late jobs. Journal of Scheduling. to appear.

Pinedo, M. L. (2012). Scheduling. Theory, Algorithms, and Systems. Springer, 4th edition.

Rahmani, D. (2016). A stable reactive approach in dynamic flexible flow shop scheduling with unexpected disruptions: A case study. Computers & Industrial Engineering, 98:360–372.

Rener, E., Salassa, F., and T'kindt, V. (2022). Single machine rescheduling for new orders with maximum lateness minimization. Computers & Operations Research, 144:20.

Rener, E., Salassa, F., and T'kindt, V. (2023). Single machine rescheduling for new orders with maximum and total time disruption constraints. Manuscript submitted for publication.

Sewell, E. C. and Jacobson, S. H. (2012). A branch, bound, and remember algorithm for the simple assembly line balancing problem. INFORMS Journal on Computing, 24(3):433–442.

Shang, L., T'kindt, V., and Della Croce, F. (2020). Branch & Memorize Exact Algorithms for Sequencing problems: Efficient embedding of Memorization into Search Trees. Computers & Operations Research, 128(105171).

Sidney, J. (1977). Optimal single-machine scheduling with earliness and tardiness penalties. Operations Research, 25(1):62–69.

Szwarc, W., Grosso, A., and Della Croce, F. (2001). Algorithmic paradoxes of the single-machine total tardiness problem. Journal of Scheduling, 4(2):93–104.

Tanaev, V. S., Gordon, V. S., Shafransky, Y. M., Tanaev, V. S., Gordon, V. S., and Shafransky, Y. M. (1994a). Scheduling Theory. Single-Stage Systems. Springer Dordrecht.

Tanaev, V. S., Sotskov, Y. N., and Strusevich, V. A. (1994b). Scheduling Theory. Multi-Stage Systems. Springer Dordrecht.

Teghem, J. and Tuyttens, D. (2014). A bi-objective approach to reschedule new jobs in a one machine model. International Transactions in Operational Research, 21:871–898.

T'kindt, V. and Billaut, J.-C. (2006). Multicriteria scheduling: theory, models and algorithms. Springer Science & Business Media.

T'kindt, V., Della Croce, F., and Esswein, C. (2004). Revisiting branch and bound search strategies for machine scheduling problems. Journal of Scheduling, 7:429–440.

Unal, A., Uzsoy, R., and Kiran, A. (1997). Rescheduling on a single machine with part-type dependent setup times and deadlines. Annals of Operations Research, 70:93–113.

Valledor, P., Gomez, A., Priore, P., and Puente, J. (2018). Solving multi-objective rescheduling problems in dynamic permutation flow shop environments with disruptions. International Journal of Production Research, 56(19):6363–6377.

Vidal, T., Crainic, T., Gendreau, M., and Prins, C. (2015). Timing problems and algorithms: Time decisions for sequences of activities. Networks, 65(2):102–128.

Vieira, G., Herrmann, J., and Lin, E. (2003). Rescheduling manufacturing systems: a framework of strategies, policies, and methods. Journal of Scheduling, 6:39–62.

Wang, D., Yin, Y., and Cheng, T. (2018). Parallel-machine rescheduling with job unavailability and rejection. Omega, 81:246–260.

Wang, G., Sun, H., and Chu, C. (2005). Preemptive scheduling with availability constraints to minimize total weighted completion times. Annals of Operation Research, 133:183–192.

Yang, B. (2007). Single machine rescheduling with new jobs arrivals and processing time compression. International Journal of Advanced Manufacturing Technologies, 34:378–384.

Yin, Y., Cheng, T. C. E., and Wang, D.-J. (2016). Rescheduling on identical parallel machines with machine disruptions to minimize total completion time. European Journal of Operational Research, 252(3):737–749.

Yuan, J. and Mu, Y. (2007). Rescheduling with release dates to minimize makespan under a limit on the maximum sequence disruption. European Journal of Operational Research, 182:936–944.

Yuan, J., Mu, Y., Lu, L., and Li, W. (2007). Rescheduling with release dates to minimize total sequence disruption under a limit on the makespan. Asia-Pacific Journal of Operations Research, 24(06):789–796.

Zhang, X., Lin, W.-C., and Wu, C.-C. (2022). Rescheduling problems with allowing for the unexpected new jobs arrival. Journal of Combinatorial Optimization, 43:630–645.

Zhao, C. and Tang, H. (2010). Rescheduling problems with deteriorating jobs under disruptions. Applied Mathematical Modelling, 34:238–243.

Zhao, Q., Lu, L., and Yuan, J. (2016). Rescheduling with new orders and general maximum allowable time disruptions. 4OR, 14:261–280.

Zhao, Q. and Yuan, J. (2013). Pareto optimization of rescheduling with release dates to minimize makespan and total sequence disruption. Journal of Scheduling, 16(3):253–260.

Zhao, Q. and Yuan, J. (2017). Rescheduling to minimize the maximum lateness under the sequence disruptions of original jobs. Asia-Pacific Journal of Operational Research, 34(05):1750024.