Scheduling with due dates and time-lags: new theoretical results and applications

by

Alessandro Condotta

Submitted in accordance with the requirements for the degree of Doctor of Philosophy.

UNIVERSITY OF LEEDS

The University of Leeds
School of Computing

May 2011

The candidate confirms that the work submitted is his own, except where work which has formed part of jointly-authored publications has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

Acknowledgements

I express my gratitude to my supervisor Dr. Natasha Shakhlevich for her continuous support and valuable guidance at all the stages of this research. I am in debt to her for the trust she gave me from the beginning and the extraordinary effort she put into guiding me.

I would also like to thank Prof. Peter Brucker for providing useful suggestions and recommendations during his visits to Leeds.

Special thanks go to all the friends that I have met here in Leeds during this long and rewarding period. Each of them contributed to making this an experience of inestimable value and to enriching me as a person.

Finally, I dedicate this thesis to my family who have given me strength and the tools to achieve my targets during this challenging journey to the doctorate.

Abstract

Manufacturing and service environments involve decisions on sequencing activities. Some examples are assembly operations in workshops, the elaboration of data by computer systems and the handling of products by operators in warehouses. Scheduling theory studies the mathematical structures of such problems with the objective of designing theoretical models and solution algorithms that can be used in practice. In this thesis we investigate scheduling models with time-lags and release/due dates inspired by two real-world problems: transportation of goods and appointment scheduling for chemotherapy patients.

The first part of this thesis studies the minimisation of the maximum lateness for two batch scheduling problems with release/due dates and equal processing times: one with a single machine and one with parallel machines. These theoretical models represent the problem of scheduling the delivery of goods within given time windows using one or more limited capacity vehicles. We design two enhanced polynomial-time algorithms that, for the single machine case, outperform the best algorithms known in literature, and, for the parallel machine case, establish the first solution algorithm.

In the second part we investigate the coupled-operation scheduling model with time-lags which characterises some important features of the problem of booking treatment appointments for chemotherapy patients. The objective is to develop a solution algorithm that minimises the maximum completion time (makespan). Initially we investigate a possible compact representation of a solution considering the sub-problem with a fixed sequence of the first operations of the jobs. We prove that this special case of the problem is NP-hard in the strong sense even in the case of unit processing times. Then we adapt a technique used for solving job shop problems with no-wait constraints to our coupled-operation problem and develop an efficient tabu-search heuristic that outperforms the algorithms known in literature.

In the last part, we introduce the problem of booking appointments for chemotherapy treatments in an outpatient clinic which is an example of real-world scheduling problem with complex time-lags and release/due dates constraints. We design an innovative 4-stage approach based on the concept of a multi-level template schedule which is generated solving a number of multi-objective integer linear programs. The evaluation of our approach using historical data shows that, using available resources, 20% additional appointments could be scheduled in the clinic eliminating peaks of workloads, maintaining short waiting days/times and improving the overall patient and staff experience.

Declarations

Some parts of the work presented in this thesis have been submitted to scientific journal:

Condotta, A., Knust, S. and Shakhlevich, N., "Parallel batch scheduling of equal-length jobs with release and due dates", *Journal of Scheduling*, 13 October 2010, 463–477.

My contributions: I developed the algorithms and the proofs for the single-machine p-batching problem, the extension with start-start precedence constraints, the enhanced binary search. I contributed to the development of the proofs for parallel-machine p-batching problem and to paper write-up.

Other authors contributions: Knust S. and Shakhlevich N. developed the ideas and the algorithm for parallel p-batching problem and the proofs of NP-Hardness for the problem with completion-start precedence constraints. They also provided supervision, feedback, general guidance and they contributed to paper write-up.

Chapters based on this paper: Chapter 2.

Condotta, A. and Shakhlevich, N., "Scheduling Patient Appointments Via Multi-level Template: A Case Study in Chemotherapy", *Operations Research*, submitted in October 2010.

My contributions: I identified the problem, gathered the requirements, developed the ideas, designed all the algorithms, conducted the experiments and interpreted the results with the contribution of my supervisor.

Other authors contributions: Shakhlevich N. discussed and developed the ideas, provided supervision, feedback and general guidance, and contributed to paper write-up.

Chapters based on this paper: Chapter 4.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Operational research is a multidisciplinary science aimed at improving the decision making process by the use of advanced analytical techniques. It is applied in almost all sectors ranging from manufacturing to the service industry and it is divided into many disciplines, one of which is scheduling. In scheduling the decision maker needs to determine the sequence and the times at which some resources are used in order to complete given tasks efficiently. A classic example is determining the best processing sequence of a set of operations to be performed in a manufacturing machine in order to obtain finished goods as quickly as possible.

The solution of a real-world scheduling problem goes through a series of steps. The first is to understand the problem which is usually achieved by observing the operations and practices in their environment. The second step consists of the problem modelling where some aspects are simplified to produce a formal mathematical model. In the third stage, the model is solved by the design of solution algorithms which use mathematical and computational techniques. Finally, in the last stage, the algorithms are tested, evaluated and passed to the final decision maker. Note that at the third stage, the design of solution algorithms sometimes goes through the reuse or extension of algorithms and models already studied in literature.

The main objective of this thesis is the development of theoretical scheduling models for real-world applications. Two major problems inspired our work: the delivery of goods by limited capacity vehicles and the scheduling of treatment appointments for chemotherapy patients. The first problem arises in the transportation sector where goods have to

be delivered to customers by limited capacity vehicles within given due dates; the goods become available at different times at the manufacturer site. The problem is to determine vehicle loading times and sequences that minimise the maximum delivery lateness. Such a problem is often part of more complex multi-stage problems involving coordination issues between production and transportation.

The second problem arises in chemotherapy outpatient clinics where complex treatments involving multiple time-constrained appointments have to be scheduled efficiently. Among other constraints, what makes the problem challenging is that the number of days interleaving two appointments is strictly regulated by the type of chemotherapy treatment and that each appointment involves a set of tasks to be performed by a nurse at specific times. The problem is to schedule appointments for arriving patients such that appointment dates and times respect the given constraints, patient waiting times are minimised and the clinic efficiency is maximised.

In the next sections we give fundamental definitions and the necessary background information to understand and appreciate the results in this thesis and the context into which it fits. An outline of the thesis and its contributions is given in Section 1.6.

1.1 Computational Complexity

In computational theory, a *problem* can be seen as an abstract question to be answered. Usually the same question can be applied to different entities which are called *instances* of the problem. An answer to the question is referred to as a *solution* of the problem and is the result of a detailed step-by-step procedure called a *solution algorithm.* The problems for which the answer can be only *YES* or *NO* are called *decision problems.* A problem instance $\mathscr{I}$ is encoded by a string in an alphabet with two symbols 0,1. The *size* of an instance is defined as the length of the representing string with such encoding. The sequence of the steps that lead to a *YES* answer in a decision problem is called a *certificate* of the problem instance.

The solution algorithms of interest in this thesis are the algorithms made of operations that are executed by a computational device such as the Deterministic Turing Machine and Random Access Memory (RAM) (see [38,76,95]). The *running time* (or time complexity) of an algorithm is the number of basic operations performed by the device to execute the algorithm on a given instance. The running time depends on the instance and it is expressed by a function $\phi(n)$ of its size n.

The *worst case analysis* is commonly used to characterise the performance of algorithms. It considers the longest running time $T(n)$ performed by an algorithm among all

possible instances of a given size n. Similar running times are grouped into complexity classes and they are represented using the "Big O" notation: by definition, a running time function $\phi(n)$ belongs to the class $O(T(n))$ if and only if there exists two constants c and n_0 such that $\phi(n) \leq cT(n)$ for each $n \geq n_0$.

A problem is solvable in polynomial-time if there exists an algorithm whose running time function belongs to the class $O(p(n))$ where $p(n)$ is a polynomial in n. Such an algorithm is called a *polynomial-time algorithm.*

A decision problem $\mathscr{P}_1$ is *reducible* to a problem $\mathscr{P}_2$ if there exists an algorithm (the *reduction*) that transforms any instance I_1 of the problem $\mathscr{P}_1$ into an instance of I_2 for the problem $\mathscr{P}_2$ such that I_2 is a YES instance for $\mathscr{P}_2$ if and only if I_1 is a YES instance for $\mathscr{P}_1$. If such an algorithm is polynomial, then the reduction is polynomial.

The complexity of a decision problem can be classified further using problem complexity classes P and NP. The class NP consists of all decision problems such that for each *YES*-instance of a problem there exists at least one certificate that can be verified in polynomial time. The class P consists of all the decision problems solvable in polynomial time. Clearly, all problems belonging to P also belong to NP, i.e. $P \subseteq NP$. Most researchers strongly believe that $NP \subsetneq P$ although it has not been formally proved.

A decision problem $\mathscr{P}$ is NP-complete if it belongs to the class NP and all problems in NP are reducible to $\mathscr{P}$. Thus, if a NP-complete problem can be reduced in polynomial time to a problem $\mathscr{P}$, then $\mathscr{P}$ is also NP-complete. Moreover, a problem is NP-hard if its decision version is NP-complete. In other words, NP-complete and NP-hard problems are at least as hard as any problem in NP. A decision problem is considered to be difficult to solve if it is NP-complete. We refer the reader to Garey and Johnson [38] and Papadimitiou [76] for a more detailed classification of the problems and a list of known NP-complete problems.

In Chapters 2 and 3 we use computational complexity theory to prove the NP-hardness of considered scheduling problems.

1.2 Combinatorial Optimization

In a *combinatorial optimization problem*, given a set of elements and an evaluation criterion, the question to be answered can be simplified to 'what is an element in the set that minimises the criterion ?'. This thesis considers combinatorial optimization problems where each element s is a solution belonging to a set of feasible solutions S (also called *solution space*) and the criterion is a function $f(s) : S \rightarrow \mathbb{Z}$ called *objective function.*

Formally, a combinatorial optimization problem $\mathscr{P}$ can be described as follows:

$$\mathscr{P} : \min\{f(s) : s \in S\}.$$

A feasible solution $s \in S$ is called *optimal* if for all $s' \in S$ we have $f(s) \leq f(s')$. In the *decision version* of a combinatorial optimization problem, an instance is a *YES instance* if there exists a feasible solution $s \in S$ such that $f(s) \leq k$, where k is a value given as input.

The study of a combinatorial optimization problem starts with an attempt to design an algorithm that finds an optimal solution in polynomial time. Although there exist many standard algorithmic tools, this step requires experience and creativity. If the attempt fails, researchers try to show that it is unlikely that such an algorithm exists by demonstrating that the problem is *NP*-hard. If the problem is *NP*-hard, then heuristics, exact exponential-time algorithms or approximation algorithms are developed as possible solution approaches.

In what follows we give a brief introduction to heuristics and exact algorithms used in this thesis. These include constructive heuristics (used in Chapter 3), local search algorithms (also used in Chapter 3) and integer linear programs (used in Chapter 4).

1.3 Heuristics

Given an instance $\mathscr{I}$ for a combinatorial optimization problem $\mathscr{P}$, a *lower bound* $LB_{\mathscr{P}}(\mathscr{I})$ is a value which is guaranteed to be less than or equal to the value that the objective function has in the optimal solution, i.e. $LB_{\mathscr{P}}(\mathscr{I}) \leq \min\{f(s) : s \in S\}$. The lower bound is useful to measure the quality of a solution s by means of an *approximation ratio* $\frac{f(s)}{LB_{\mathscr{P}}(\mathscr{I})}$.

An *approximation algorithm* is an algorithm that generates for any instance a solution with a guaranteed approximation ratio. A special case of approximation algorithms are exact algorithms. An *exact* algorithm is an approximation algorithm that always finds a solution s with approximation ratio 1 within a limited amount of time (although it may be very large), i.e. $\frac{f(s)}{LB_{\mathscr{P}}(\mathscr{I})} = 1$. Differently, a *heuristic* algorithm is one that generates solutions for which no approximation ratio has been proved.

The classification of heuristic algorithms is very wide and distinguishes the techniques used to generate feasible solutions. In the following subsections we introduce two of them: local search and constructive heuristics.

1.3.1 Constructive Heuristics

For some combinatorial optimization problems it is easy to design one or more procedures to generate a feasible solution or part of it. We call such procedures *construction rules*. A constructive heuristic is an algorithm that builds a solution by repeatedly applying one or more construction rules.

We distinguish between *deterministic* and *non-deterministic* (or *random*) constructive heuristics. In the first type, given a problem instance, the construction rules used and their execution sequence are fixed a priori. Typically a deterministic constructive heuristic is executed only once for each problem instance since multiple runs would lead to the same solution. In the second type, the rules used and their execution sequence are chosen randomly from a known distribution. Typically the algorithm is run multiple times in order to generate many different solutions. The solution considered is usually the one with the best objective function among the ones generated.

Notice that, differently from Local Search heuristics (see Section 1.3.2), in a constructive heuristic there is no improvement process: each time a new solution is generated, it is re-built from scratch without using knowledge from the solutions previously generated. Constructive heuristics are preferred when fast computations are necessary and an easy procedure to build a feasible solution is known.

A constructive heuristic for the coupled-operation scheduling problem is presented in Chapter 4.

1.3.2 Local Search

For some combinatorial optimization problems it is easy to define a set of actions M that transform a feasible solution $s \in S$ into another feasible solution $s' \in S$ called a *neighbour* of s. Such actions are usually referred to as *moves* and can be represented by a function $m(s,a) : S \times M \rightarrow S$, where a is an indicator of the move to apply. Given a solution s, the function m defines the set $\mathcal{N}(s) = \{m(s,a) : a \in M\}$ of all possible neighbours of s called the *neighbourhood* of s.

Two solutions $s, s' \in S$ are said to be connected over a neighbourhood function m if there exists a sequence of solutions $s_0, s_1 \ldots, s_{k-1}, s_k$ such that $s_0 = s$, $s_k = s'$ and $s_{i-1} \in \mathcal{N}(s_i)$, $0 < i \leq k$. The solution space S is said to be *optimally connected* over a neighbourhood function m if any solution s is connected to at least one optimal solution. A solution s' is said to be a *local optimum* for the neighbourhood $\mathcal{N}(s)$ if $f(s) \leq f(s') \;\; \forall s' \in \mathcal{N}(s)$. Clearly an optimal solution is also a local optimum, but not vice versa. Notice that, differently from a construction rule (see Section 1.3.1) which is

used to complete a partial solution, a move is usually applied only to a complete solution in which the objective function can be evaluated in full.

A *local search* algorithm explores the solution space moving from one solution to another one in its neighbourhood. The search process continues until a *termination condition* is triggered or an optimal solution is found. A termination condition usually consists of a time limit, a maximum number of iterations or a required approximation ratio to be reached.

For a current solution s, the next neighbour to visit is determined by an *acceptance policy* which varies depending on the algorithm design. *Best improvement* is an example of a common acceptance policy which accepts at each iteration the neighbour solution with the minimum objective value. Often, local search algorithms using such a policy terminate having found only local optima. To overcome such a problem, local search algorithms implement special *escape procedures* (usually in the acceptance policy) in order to allow the algorithm to move from a local optimum to other solutions not in its neighbourhood or with higher objective values.

During the search process, local search heuristics may visit the same solution multiple times. Often, this is due to the escape procedures which allow visiting of solutions with objective values higher than the current one. To overcome this problem local search algorithms implement an *anti-cycle* mechanism. A representative example is the list of 'forbidden' solutions (*tabu list*) used in *Tabu search* heuristics: every time a new solution is generated, its compact description is put in the tabu list so that it is not visited again in the future. Since the list has a limited capacity, its implementation defines a specific list management policy which determines the solution to be removed from the list when the length exceeds the capacity. For additional information about local search algorithms we refer the reader to Aarts and Lenstra [47] and Glover and Laguna [40].

In Chapter 4 we present a tabu search algorithm for the coupled-operation scheduling problem.

1.4 Integer Linear Programming

Linear programming is a technique to model combinatorial optimization problems where:

- a solution of the problem is determined by the values of numeric variables called *decision variables*;
- the solution space S is determined by a set of linear inequalities called *linear constraints* defined over the decision variables;

- the objective function is a linear function of the decision variables.

Thus a linear program can be represented in a matrix form as follows:

$$\begin{aligned} LP: \quad & \min c^T x \\ & Ax \leq b, \end{aligned}$$

where x are the decision variables, c is a vector of coefficients defining the linear objective function f, A is a matrix defining the left-hand side coefficients of the constraints and b is the vector of constants representing their right-hand sides.

A linear program has many important properties that derive from the *convex analysis* of its solution space and from the *duality theory*. There exists three main classes of solution algorithms for linear programs: simplex method, interior point and ellipsoid algorithms. Since the ellipsoid algorithm has a polynomial running time [54, 55] any combinatorial optimisation problem that can be represented by a linear program belongs to the complexity class P. Despite this, most research has been focused on techniques to improve the simplex method which is simpler and often more efficient in practice than the others.

An *integer linear program* is a linear program where decision variables are forced to be integer. Thus an integer linear program has the following form:

$$\begin{aligned} ILP: \quad & \min c^T x \\ & Ax \leq b \\ & x \in \mathbb{Z}. \end{aligned}$$

The LP obtained by removing the integer constraints $x \in \mathbb{Z}$ is a *relaxed* version of the original ILP and is called *LP-relaxation*. The optimal solution of a LP-relaxation is always a lower bound of the optimal solution of the related ILP, i.e. $\min\{f(s) : s \in S_{ILP}\} \geq \min\{f(s) : s \in S_{LP}\}$ where S_{ILP} and S_{LP} are the solution spaces defined by the ILP and its LP-relaxation respectively. An integer linear program in which some variables are forced to be integer and some others are not, is called Mixed Integer Linear Program *MILP*.

The presence of the integrality constraints $x \in \mathbb{Z}$ makes the solution of an integer linear program a *NP*-hard problem. However, some special integer linear programs can be solved in polynomial time. This is the case for problems whose coefficient matrix A has the *total unimodularity property*, see [77]. Since the solution of an integer linear program is an *NP*-hard problem, mainly heuristics and branch-and-bound methods are used as solution algorithms. Both types of algorithms use lower bound calculations to drive the search and reduce the computational efforts. For these reasons research into

integer linear programming is largely focused on strengthening the lower bound obtained by LP-relaxation of the problems. A more comprehensive survey of fundamental results for linear and integer linear programs can be found in [14, 77, 106].

In Chapter 4 integer linear programs are used extensively for the construction of the template schedule and for the re-allocation of nurses in the appointment scheduling problem.

1.5 Scheduling Theory and Applications

In this thesis, the term *scheduling* refers to the problem of determining the assignment of a set of activities to some resources over time such that a given criteria is optimised. The solution of such a problem is called *schedule* and consists of a description of the times and the resources each activity is assigned to. With some exceptions in Chapter 4, we consider the case in which the activities and the availability of resources are known in advance. Thus the scheduling problems that we consider are combinatorial optimization problems in which the solution space is a set of feasible schedules.

Studies on scheduling problems started in the early fifties for the design of efficient production plans in the manufacturing industry. Typically in this context, activities represent production steps (called *jobs*) to be performed in production *machines* which constitute the limited resources. The classical goal is to reduce the production time and to maximise the efficiency of the machine usage. Nowadays scheduling problems are studied for a wide range of applications such as project planning, train and flight scheduling, sport timetabling, staff rostering and many others. The solution models generated for such different environments share similar structures that justify the existence of a general scheduling theory in which the models are simplified and analysed from a mathematical and computational perspective.

Different terminologies exist that are used in scheduling literature are derived from various application contexts such as *project planning*, *resource-constrained scheduling* and *manufacturing scheduling*. In this thesis we use the classical terminology from the manufacturing scheduling where a set N of jobs (activities) have to be scheduled to production machines (resources). In addition we use the wording '*operation* of a job' to indicate a single activity of a job that has to be performed by a production machine. Each job can have one or more operations but each operation belongs to only one job.

Traditionally, scheduling problems are classified according to the *three-field scheme* $\alpha|\beta|\gamma$ presented in [42], where α indicates the *machine environment*, β describes the *processing restrictions* and γ specifies the *optimality criteria*.

The machine environment α mainly characterises the type of the machines involved and their number m. Some examples are the *unrelated machines environments* such as the single machine 1, the identical parallel Pm and the parallel machines with different speeds Qm environments; others are the *shop environments* such as flow shop Fm, job shop Jm and open shop Om. In the unrelated machines environment, all machines are of the same type and each job can be processed in any of them, although at different posssible speeds. Differently, in shop environments, machines are of different types and operations of jobs have specific constraints determining the machines they have to be processed on. The most general case is job shop where each job has a specific machine order to follow.

The problems studied in this thesis involve single and parallel machine environments. For a more detailed classification of machine environments see [19, 80].

The processing restrictions β characterise the constraints involved in the job processing. Some examples of job constraints are:

- r_j indicates the presence of a release date for each job $j \in N$; job j cannot be processed before its release date r_j.
- d_j indicates the presence of a due date for each job $j \in N$; job j should not be processed after its due date d_j.
- p-batch or s-batch indicate that multiple jobs are processed together in batches; the word p-batch is used when the jobs are processed simultaneously and s-batch when jobs in a batch are processed in series.
- *prec* indicates that the processing of jobs is subject to *precedence constraints*.

Field β contains combinations of many other constraint symbols which result in self-explanatory formula. For example, the writing $p_j = p$ indicates that all job processing times are equivalent to a value p.

The optimality criteria γ is an *objective function* usually defined over the completion times and the job sequence. The objective functions that interest us in this thesis are called *min-max* objectives (or *bottleneck* objectives) and are of the form $f_{\max} = \max_{j \in N} \{f(c_j)\}$ where N is the set of jobs to be scheduled. In particular, in Chapter 2 we consider the minimisation of the maximum lateness calculated as

$$L_{\max} = \max_{j \in N} \{c_j - d_j\} \tag{1.1}$$

where $c_j - d_j$ is the lateness of job j. The maximum lateness is used as a fair representation of the overall punctuality of a process to given due date targets. In Chapter 3 we

focus on the minimisation of the *maximum completion time* (*makespan*) defined by the following formula:

$$C_{\max} = \max_{j \in N} \{c_j\} \tag{1.2}$$

The maximum completion time is a sensible measure for the speed and efficiency of a system with possible idle times.

An example of three-field notation is given by $1|p-batch, b=4, p=1, r_j, d_j|C_{\max}$ which denotes the problem of minimizing the makespan on p-batching machines where at most 4 jobs can be processed simultaneously in batches of unit processing time and jobs are released at given release dates and should be processed before given due dates. More examples and detailed explanations of specific machines and their processing restrictions are introduced in later sections and chapters.

In Chapters 2, 3 and 4 we study scheduling problems with release and due dates, and scheduling problems with time-lags. The following sections give an overview of such scheduling areas.

1.5.1 Scheduling with Release and Due Dates

In scheduling with release and due dates, each job j of a set N is available to be processed after a release date r_j and should be completed before a due date d_j. There exists a *hard* and *soft* version of due dates: in the hard version the due dates are called *deadlines* and no job in a feasible schedule can be completed after them; in the soft version jobs that are completed after their due dates incur penalties. The due date version and the penalties eventually involved are determined by the objective function in use. In the three-field scheme, problems with release and due dates are usually denoted with the symbols r_j, d_j in the β-field. When the presence of due dates is clear by the nature of the objective function, d_j is omitted.

Originally, such constraints arose in the manufacturing industry where production processes use materials and resources which have restricted availability and the goods produced have to be delivered within given due dates in order to meet production and market requirements. In this context scheduling techniques help to maximise the efficiency of machine usage together with the delivery performances. Modern applications such as service scheduling also involve release and due dates constraints. Some examples can be found in healthcare and, in particular, in appointment scheduling where clinical referrals establish the dates at which patients can start to be treated and reasonable waiting day targets determine the latest dates when the treatment is due. A measure of patient waiting days can be modelled in the objective function as the average or maximum lateness of pa-

tients. Successful appointment scheduling is shown in literature to substantially improve the quality of the service and to reduce costs [25]. Release and due dates are also fundamental constraints in logistics where commodities are available at different times/dates and have to be delivered to a destination before agreed due dates. Notice that in this case the processing of a job represents the transportation of some goods. A study on this context is presented in chapter 2 where we consider a special case in which goods are transported by limited capacity vehicles.

Most of the traditional scheduling problems such as parallel machines or shop problems have been studied in the version with release and due dates (see examples in [19,27,80,86,87]). In the rest of the section, we present some of the fundamental problems which have deeply influenced literature in the field and whose study has led to important solution techniques, some of which are exploited in later chapters.

One of the simplest but very challenging problems in scheduling with release and due dates is the single machine problem $1|r_j|L_{max}$ described in [19]. Given a set N of jobs, one has to determine the sequence and the starting times of the jobs to be processed by a single machine such that the maximum lateness is minimised. Each job $j \in N$ can start only after its release date r_j and no preemption is allowed. Although, $1|r_j|L_{max}$ is known to be NP-hard in the strong sense (see [60]), there are some special cases of intuitive solutions:

- $1|r_j = r|L_{max}$: single machine problem with equal release dates; it is solved scheduling the jobs in non-decreasing order of due dates (Jackson's algorithm [51]).
- $1|r_j, d_j = d|L_{max}$: single machine problem with equal due dates; it is solved scheduling the jobs in non-decreasing order of release dates (Jackson's algorithm [51]).
- $1|r_j, p = 1|L_{max}$: single machine problem with unit processing times and integer release dates; it is solved by scheduling in each time slot the available job with the earliest due date (Horn's rule [48]).

All the problems mentioned are solved by simple rules (called *dispatching rules*) that are repeatedly applied for each job and that find an optimal solution in $O(n \log n)$ time.

Example Consider a retailer warehouse where every day the products are delivered to customers in the local area. The transportation is performed by a van that has to deliver the goods no later than the due times agreed with the customers. Due to the limited capacity, the van can deliver one product a time. The amount of time necessary for the loading/unloading and the round-trip is well known and is called delivery duration. On

a typical day, 10 costumers are served by one van. Table 1.1 shows the time at which the products become available, the delivery duration and the due times agreed with the customer. The operations manager wants to determine the vehicle loading sequence that

Customers	Availability time	Delivery duration	Due times
1	09:00	00:45	14:15
2	09:00	00:15	13:15
3	10:00	00:45	12:00
4	10:15	01:15	11:45
5	12:00	01:30	14:30
6	12:30	00:15	13:45
7	12:45	00:30	14:30
8	13:15	00:30	15:45
9	14:00	00:45	15:30
10	16:30	00:30	17:00

Table 1.1: Product availability times, due times and delivery durations

guarantees a timely delivery to customers.

A fair measurement for the punctuality is the maximum lateness of deliveries among all the customers served on that day. Thus, such a problem can be modelled as a single machine scheduling problem with release and due dates with maximum lateness objective function, i.e. $1|r_j|L_{\max}$. Notice that in this case the release and due dates are actually release and due times.

A feasible solution to the problem is given in Table 1.3 and by the Gantt chart is shown in Figure 1.1. The maximum lateness of the schedule is 1 and it is determined by the completion time of jobs 3, 7 and 8. This means that each customer would receive the goods with a delay no longer than 15 minutes. □

Another important version is the single machine problem with release and due dates where the processing of a job can be stopped and resumed at any time $1|r_j, pmtn|L_{max}$. The possibility of interrupting the processing of a job is called *pre-emption* and is denoted by the word 'pmtn' in the β-field of the three-field scheme. Pre-emption makes the single machine problem with release and due dates easy to solve by the use of the following simple algorithm: at each decision point, which corresponds to the release date of a job

Job	Release date (r_j)	Processing time	Due date (d_j)
1	0	3	21
2	0	1	17
3	4	3	12
4	5	5	11
5	12	6	22
6	14	1	19
7	15	2	22
8	17	2	27
9	20	3	26
10	30	2	32

Table 1.2: Instance data of the relative single machine scheduling problem $1|r_j|L_{max}$; the customer deliveries are now jobs and the product availability times, due times and delivery durations are expressed in 15-min time slots

Seq. No	1	2	4	3	6	5	7	9	8	10
Time of delivery	9:00	9:45	10:15	11:30	12:30	12:45	14:15	14:45	15:30	16:30
Starting slot	0	3	5	10	14	15	21	23	26	30

Table 1.3: Example of a feasible non-optimal solution

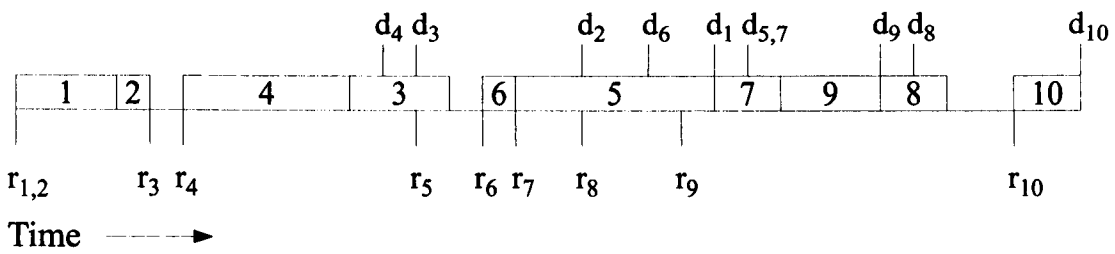

Figure 1.1: Gantt Chart of the solution given by Table 1.3

or its completion time, schedule an available job with the smallest due date. Problems with pre-emption are of great interest since their optimal solutions are lower bounds for the problem versions with no pre-emption and they can be used to drive the search in the branch-and-bound and local search algorithms.

The single machine problem with precedence constraints, release and due dates $1|r_j, prec|L_{max}$ is also of great interest since it models many sub-problems in complex multistage scheduling machine environments such as flow shops, job shops and open shops (see Section 1.5). A precedence constraint between two operations determines the order in which the operations have to be processed, i.e. $i \rightarrow j$ indicates that operation i must start the processing before j. This type of constraint appears in the representation of partial solutions generated during the search in branch-and-bound and local search algorithms. Notice that the problem $1|r_j, prec|L_{max}$ is NP-hard since its relaxed with no precedence constraints $1|r_j|L_{max}$ has been proved to be NP-hard in [60].

The most important studies in relation with the problem $1|r_j, prec|L_{max}$ involved the development of two efficient solution techniques for the problem $1|r_j|L_{max}$: *block approach* and *immediate selection.*

Given a feasible schedule, the block approach determines a set of contiguous jobs (called a block) inducing the makespan. The technique search for better schedule moving outside the block some of the jobs that were scheduled inside the block and vice versa. Notice that imposing a job to be processed before or after a block is equivalent to setting specific precedence constraints between jobs inside and outside the block leading to a problem of the form $1|r_j, prec|L_{max}$. An example of a successful application of this technique can be found in the branch-and-bound algorithms from [41]. Immediate selection is a technique used in local search and branch-and-bound algorithms to speed up the search of the optimal solution. It is based on simple constraint propagation techniques that use lower and upper bounds to increase release dates and decrease due dates of the jobs to be scheduled. Once the order of two jobs in a optimal solution is evident by the value of the release and due dates, the corresponding precedence constraint is immediately fixed. An example of the successful application of this technique can be found in [22].

The algorithms for the single machine problems also give rise to advanced techniques for more complex problems. Clear examples are the block approach and the immediate selection which have been extended to the job shop problems and used to tackle the famous *10 jobs 10 machines* benchmark problem which remained unsolved for more than two decades see [20,24,59]. Another important example is the *shifting bottleneck heuristic* for the job shop (see [2, 80]). Such a heuristic reduces the solution of the job shop problem to the solution of many conveniently adapted single machine problems . In par-

ticular, the heuristic searches for a solution by iteratively relaxing each stage of the shop to appropriately defined single machine problems $1|r_j|L_{max}$ and solving them with available algorithms (see [80]).

To complete our overview of influencing scheduling problems with release and due dates, we describe the single machine problem $1|r_j, p_j = p|L_{\max}$ where all jobs require the same processing time p. This special case is solvable in polynomial time by complex combinatorial algorithms described in [93, 94]. The algorithm works solving iteratively many instances for the decision version of the problem $1|r_j, p_j = p|L_{\max} \leq k$ with different k values. For each job $j \in N$, the decision version involves modified deadlines $d'_j = d_j + k$ after which a job j cannot be completed. Then a solution of the decision version of the problem is computed in two steps: the generation of *forbidden regions* and the actual scheduling of the jobs. In the first step, the algorithm determines intervals of times during which, if a job starts to be processed, it exceeds the given deadlines, i.e. the forbidden regions. In a second stage, jobs are sequenced so that no starting time falls in a forbidden region. Such algorithms have also been used to solve the parallel machine problem $P|r_j, p_j = p|L_{\max}$ in [94].

In Chapter 2 we solve the parallel batching scheduling problem with release and due dates for single and parallel machines extending the idea of forbidden regions. Similarly to what happened with the study of different versions of the problem $1|r_j|L_{\max}$, we expect that our results will influence future work on the batch scheduling problems in complex shop environments.

1.5.2 Scheduling with Time-Lags

In scheduling problems, time-lag constraints impose restrictions on the difference between the starting times of pairs of operations. Formally, given two operations v, w with starting times s_v and s_w, a time-lag constraint imposes that

$$l \leq s_v - s_w \leq u \tag{1.3}$$

where l and u are given integer values and $l \leq u$. We refer to such a general form of constraint as *start-start* constraints. If the value l is greater than the processing time p_w of the operation w, then the constraint is of the type *completion-start* and formula 1.3 can be transformed to

$$l - p_w \leq s_v - c_w \leq u. \tag{1.4}$$

Notice that this is not always the case since in problems with parallel machine and batching environments, or in problems where pre-emption is allowed, an operation can start to be processed before the completion of another one already started.

We distinguish between *coupled* and *uncoupled* operation scheduling problems (see Figure 1.2). In coupled-operations problems, there is a constraint between pairs of operations belonging to the same job (see examples in [74, 81]). In uncoupled-operations scheduling problems, the time-lag constraints can link any pair of operations, even if they belong to different jobs (see examples in [9], [18], [49] and [89]).

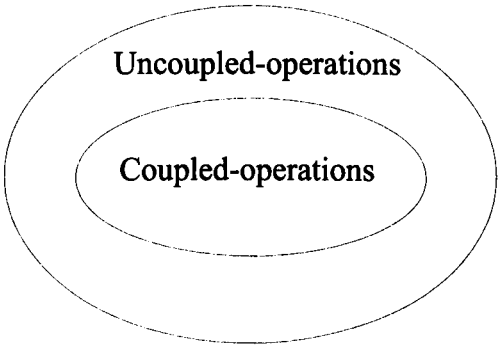

Figure 1.2: Classification of problems according to the operations constrained by time-lags

Problems with time-lags are also classified according to the characteristics of the upped bound u and lower bound l values (see Figure 1.3). In particular we distinguish between problems with *flexible*, *minimum*, and *fixed* time-lags. In problems with flexible time-lags the values of l and u are distinct, allowing some variability in the starting time difference between operations; i.e. $l < u$ (see examples in [18, 36, 81]). In problems with minimum time-lags the difference between operation starting times can grow indefinitely, i.e. $u = \infty$ (see examples in [9, 35, 82]). Finally, if the time-lags are classified as fixed, the possible starting time differences degenerate to single values, i.e. $l = u$ (see examples in [61, 74]).

Extending the notation from [42] and [74], in this thesis we denote coupled-operation problems including the letters l_j and u_j in the β-field. However, if $l_j = u_j$ as for the fixed time-lag case, we use the upper case letter L_j; for example $1 \mid a_j, b_j, L_j \mid C_{\max}$ indicates the coupled-operation single machine problem with fixed time-lags and a maximum lateness objective function.

Time-lags constraints arise in many different areas ranging from command-and-control systems in electronics to process and resource scheduling in the manufacturing

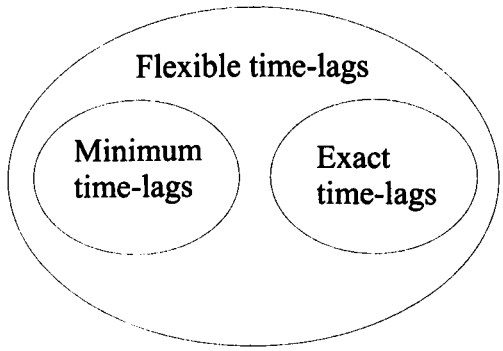

Figure 1.3: Classification of problems according to the time-lags characteristics

and service industries. In radar controllers scheduling, the problem is to schedule the emission of multiple wave pulses such that all bounced signals can be captured accurately. The time-lags constraints are caused by the amount of time between the emission and the reception of a bounced signal which is dependent on the distance of the object that is to be tracked (see examples in [32, 33, 75, 104]). However, radar controllers are only an example of command-and-control systems where electronic boards request data from sensors that takes an considerable amount of time to be retrieved, other examples can be found in [90–92]. A similar problem arises in modern high performance computing infrastructures where complex calculations are divided between many processors. A server acting as coordinator, sends the data to many client computers which process and return it to the server. The server has to wait a minimum time-lag before which a processor can return the elaborated data. The problem is to determine the time at which the transmission of the data should be performed such that the use of the overall computation time is minimised (see example in [62, 64, 90]).

Typically, in manufacturing, time-lags constraints appear between different processing stages. In some cases, fixed or flexible time-lags are imposed by the cooling and deterioration of the material used. One example can be found in the steel industry where iron should be handled while warm (see examples in [81, 105]). In flexible manufacturing time-lags are due to manual handling or transportation issues. Unfinished goods, for example, need to be transported from one workstation to another by a human operator or robots which add considerable delay in the production chain [16, 23]. In other cases the production process requires some human intervention during the process to check or add specific customisations to the products. These interventions determine a minimum delay between two successive processing stages that could result in efficiency loss if not scheduled properly (see examples in [66, 82]). Time-lags also occur in single-stage processes where, after the production, goods need to be reprocessed by the same machine

for some adjustments to be made [83]. In literature, this feature is represented by models with *re-entrant* jobs.

Examples of models with time-lags in the service industry can be found in maintenance service and healthcare. In maintenance service, after an on-the-spot investigation identifies some faults at a customer site, some replacement parts need to be ordered to a distributor which requires a certain amount of time to fulfil the request. Clearly, a technician can replace the faulty part only after it has been delivered. Thus, the problem for a maintenance company is to schedule the technician visits to multiple customers minimising the technician idle times due to unavailability of replacement parts. An example of application in healthcare service area is mentioned in [108] where a minimum time-lag has to elapse between the administration of anaesthetic and the introduction of the patient into the operating room. In Chapter 4 we introduce a new scheduling problem with time-lags in healthcare.

Example Consider a small facility where printed circuits boards (PCBs) are coated with thin films using a chemical bath deposition method. A single hoist grasps each PCB, takes it to a preparation stage and then dips it into the appropriate tank in which the coating agents are placed. Depending on the size of the PCB and on the type of coating needed, each PCB remains in the tank for a fixed amount of time after which it must be removed. While a PCB is inside a tank, the hoist can load or remove other PCBs from the same or other tanks. After the immersion, the PCB is taken through a post-processing stage where additional actions such as drying take place. During the preparation and the post-processing stage the PCB is held by the hoist.

	Durations		
PCB No.	Preparation/Loading	Coating	Removal/Post-processing
1	00:45 (3)	02:00 (8)	01:00 (4)
2	01:00 (4)	03:30 (14)	01:00 (4)
3	00:15 (1)	02:00 (8)	00:15 (1)
4	00:15 (1)	02:15 (9)	00:30 (2)
5	00:30 (2)	02:45 (11)	00:30 (2)
6	00:30 (2)	01:45 (7)	00:45 (3)

Table 1.4: Example of hoist scheduling problem; numbers in parenthesis represent the durations expressed in time slots

Assuming that Table 1.4 summarises the number of PCBs coated on a typical day

and their coating times, what is the sequence of hoist loads/unloads that minimises the amount of time that the whole process takes? This problem can be modelled as a coupled-operation scheduling problem where the processing machine is the hoist and each job is made of two holding operations: the preparation and the post-processing of the PCB. The objective function is the minimisation of maximum completion time (makespan).

A feasible solution sequence is represented in Table 1.5 and in the Gantt chart shown in Figure 1.4. In such a solution the total amount of time required to process all PCBs is 32 unit-time slots (8 hours). ■

Job	6	4	2	1	5	3
Starting slot	0	2	3	14	17	20
Starting time	00:00	00:30	00:45	03:30	04:15	05:00

Table 1.5: Sequence and starting times of hoist operations of a feasible solution

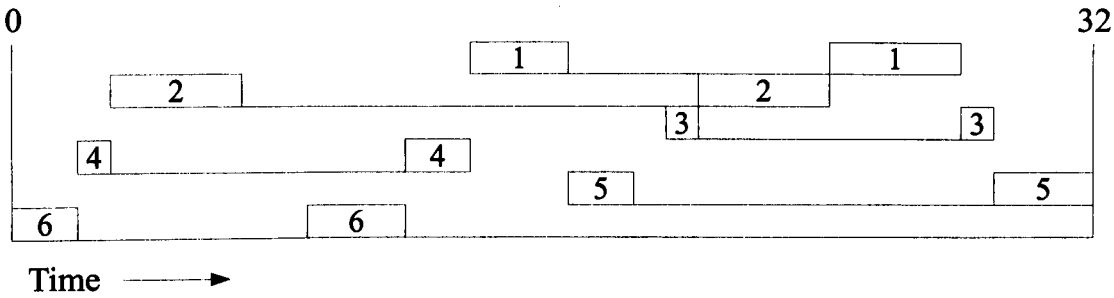

Figure 1.4: The Gantt chart representation of the solution in Table 1.5

A natural optimization criteria for problems with time-lags is the minimisation of the total idle time in the resulting schedule. This is equivalent to the minimisation of the maximum completion time of the jobs on the machines involved. In fact, most of the studies in literature focus on the minimisation of the maximum makespan $L_{\max}$ or, less frequently, on the total weighted completion time $\sum_{j\in J} w_j c_j$. Unfortunately, many of the general problems with such objective functions have been proved to be NP-hard (see [74, 102, 109]).

Although, many scheduling problems with time-lags have been studied in the past, there exists no standard nomenclature used by authors. This makes a comprehensive survey of the literature and a strict classification difficult to achieve. For example the time-lags are referred using the term *delays* in [9, 21, 69, 70, 88, 109] and *limited waiting*

time constraints in [107]; coupled-operation jobs are called *coupled-tasks* in [74], *two-phased* jobs in [90] and *chains* in [21, 68]. In some cases even the same authors used different names in different paper; some examples are Brucker [18, 21], Munier [68, 70] and Potts [74, 81]. In the rest of the section we provide a general overview of the studies in literature related to scheduling models with time-lags. A detailed survey of complexity results and open problems specific to coupled-operation scheduling problems is presented in the first part of Chapter 3.

Initially, time-lags constraints were introduced by Roy [85] during the development of the Metra Potential Method for project planning. The methodology introduced to describe the interdependence between activities in a project included the possibility of establishing a minimum and a maximum delay between the execution of two activities. This inspired later studies in theoretical models with time-lags.

The most general formulation for the problems involving time-lags is the single machine problem with flexible time-lags and uncoupled operations. Such a problem was studied in the original form by Wikum et al [102] and by Brucker et al [18]. The first investigated the complexity of the problem showing that it is NP-hard and designing a few polynomial-time algorithms for very special cases. Differently, Brucker et al [18] designed a general branch-and-bound solution algorithm using immediate selection (see [22]) and other techniques developed for the single machine problem with release and due dates. Differently, Balas et al in [9] study the problem where only positive time-lags are allowed proving the NP-hardness of the problem and developing an efficient heuristic. Their work was inspired by the application that such a problem has in the shifting bottleneck solution procedure for job-shop. Other studies on the general formulation has been done in the area of resource constrained project scheduling with the studies of Bartush et al [12], Brinkman and Neumann [17] who developed heuristics procedures for variants of the problem.

The importance of studying the general formulation of the single-machine problem is strengthened by the results in [18] which show that many other scheduling problems such as general shop problems and scheduling problems with multi-purpose machines, with parallel dedicated machines and with multi-processor tasks can be reduced to the single machine problem with time-lags.

A large number of theoretical scheduling models with time-lags are considered for multi-stage environments such as flow-shops, job-shops and open-shops. The common characteristic of such models is that time-lags constraints link operations of the same jobs (coupled-operation). In some models coupled-operations must be processed in different machines [30, 37] while in others they must be processed in the same machine [84]. Since

most of the problems in this area are NP-hard, accurate lower bounds, heuristics and branch-and-bound algorithms have been mainly developed.

Many studies of scheduling in multi-stage environments are linked to models with fixed time-lags even if it was not explicitly stated by the authors. Of this type are the problems with *no-wait* constraints. Such constraints dictate that two operations of the same job must be processed contiguously with no time delay between the completion of the first and the starting of the second operation [16, 53, 67]). A no-wait constraint is equivalent to a fixed time-lag constraint with a value equal to the processing time of the first operation. Another similar form of constraint is the so called *blocking* constraint where, given two operations of the same job, after the processing of the first operation a machine remains idle until the processing of the second operation is started (possibly on a different machine). Such a restriction is common when no buffering or storage capacity is available between two processing machines [44, 67]. Clearly, a feasible solution for a problem with no-wait constraints can be transformed into a feasible solution for the blocking version of the problem and vice versa.

Many successful studies in this area focused on local search heuristics based on the reinsertion neighbourhood. Such a technique consists of generating a better schedule from an existing one by removing some operations and reinserting them in better positions in the schedule. Due to the similarity in the constraints structure, solution algorithms for shops with blocking and no-wait restrictions can be extended to scheduling problems with time-lags. In fact, the recent results in reinsertion techniques developed by Groflin et al in [43–45] gave rise to our work in coupled-operation scheduling with time-lags described in Chapter 3.

From the complexity perspective, literature shows that problems involving time-lags are generally hard to solve and that polynomial-time algorithms could be designed only for special cases. Solution approaches attempted are mainly heuristics and branch and bound algorithms. The difficulty in finding efficient algorithms results from the fact that even feasible solutions are hard to construct. This forced the design of solution algorithms which allow infeasible solutions to be considered in the search [49, 81] or which limit the search to a subset of feasible solutions [63]. This has been conjectured to be the cause of the limited performance of the algorithms. In the case of coupled-operation problems, constructive heuristics have been recognised as the best approach to achieve good solutions in a short amount of time [81].

In summary, time-lags constraints of different forms find applications in various real-world problems and, because of this, they have been considered in many theoretical scheduling models. It has been recognised that problems with time-lags involv-

ing single-machine environments are among the more difficult problems. Solution approaches adopted are mainly heuristics and branch-and-bound algorithms which include in the search unfeasible solutions or limit the search to only subsets of feasible solutions. As a result, many improvements can still be brought to this area of research. In particular, the design of theories which could support the design efficient solution algorithms for general scheduling models would be of great benefit. This is the aim our work in Chapter 3 where we study a possible compact representation for solution schedules and we adapt the theory developed for the no-wait job-shop to the single-machine scheduling problem with coupled-operations.

1.6 Contributions of the Thesis

The contributions of this thesis to scheduling area are manifold. We study issues in the fields of combinatorial optimization, operational research and computing. In particular we provide new complexity results for combinatorial optimization problems (Chapters 2 and 3), we design efficient algorithms for scheduling problems (Chapter 2 and 3), we introduce a new real-world problem and we conceive an innovative solution technique for it (Chapter 4).

This thesis is structured as follows. In Chapter 2 we present new results on scheduling with release/due date constraints for batching machines. We introduce algorithms for the case of single and parallel machines that outperform previously known algorithms. In addition we determine the computational complexity of the problem with two types of precedence constraints: start-start and completion-start precedence constraints.

In Chapter 3 we study coupled-operations scheduling problems. In particular, we concentrate on the special case where each job consists of exactly two operations whose starting time difference is determined by a fixed time-lag. Among other results, we prove that the problem of determining the sequence of operations that minimises the makespan is NP-hard in the strong sense even if a partial sequence of the operation is given and processing times are unitary. We show the relation between coupled-operation scheduling problems and the short cycle property used in [43] for the solution of the blocking job shop scheduling problems. Using such a theory we design a tabu-search algorithm for our problem and we compare it with other algorithms in literature.

In Chapter 4 we study a real-world scheduling problem arising in outpatient chemotherapy clinics. Specifically, we consider the problem of scheduling appointments with complex time patterns of patients arriving over a period of time. We design an innovative approach that schedules patient appointments and allocates them to a limited

number of nurses such that given waiting targets are respected and workload peaks are minimised. Our contribution consists of the introduction of the new concept of the multi-level template schedule and the design of specific integer linear program formulations for its construction. Such a problem is a relevant example of a real-world scheduling problem with release/due dates and time-lags.

Conclusions of the thesis and possible future research directions are discussed in Chapter 5.

Chapter 2

Parallel Batch Scheduling of Equal-length Jobs with Release and Due Dates

2.1 Overview

In this chapter we study scheduling problems in which jobs can be processed simultaneously in batches. It is assumed that all jobs have equal processing times and the same value defines the processing time of every batch. Due to processing restrictions, a limited number of jobs can be allocated to one batch. In the scheduling literature, such a model is often called a *parallel batching* model or *p-batching*, see [19].

Such a scenario is typical for transportation area when deliveries between two locations are arranged by vehicles which can carry several items at the same time. In such a problem a set of items are sited in one location and have to be moved to another location before given due dates. Each item is available to be moved only after known release dates. Since the distance between the locations is fixed, the duration of all the deliveries is the same. This problem is represented by a p-batching scheduling model where each machine represents a limited capacity vehicle and the processing of a batch of jobs by a machine represents a delivery of items by the associated vehicle.

Formally the scheduling problem can be described as follows. There are n jobs of a

set N which have the same processing times $p_j = p$, $j \in N$. Each job $j \in N$ becomes available for processing at its release date r_j and it should be completed by a deadline (or due date) d_j. The jobs can be processed in any order and up to b jobs can be processed jointly in one batch.

If there is a single processing machine, then a schedule is defined by a sequence of batches $B_1, B_2, \ldots, B_\alpha$, their starting times $s(B_q)$ for $1 \leq q \leq \alpha$, and the sets of jobs assigned to each batch B_q. If there are $m > 1$ parallel batching machines, then in addition the batches have to be allocated to the machines.

Since no job can be processed before its release date, we must have $s(B_q) \geq \max_{j \in B_q}\{r_j\}$ for any batch B_q. All jobs $j \in B_q$ of the same batch are completed simultaneously at time $c_j = s(B_q) + p$.

A schedule is *feasible* if the jobs are completed before their deadlines, i.e., if $c_j \leq d_j$ for all $j \in N$. If no feasible schedule exists, then the deadlines are considered as (soft) due dates and the overall performance of a schedule is measured in terms of the *maximum lateness* $L_{\max} := \max\{L_j \mid j \in N\}$, where $L_j := c_j - d_j$ is the lateness of job j.

Using the standard three-field notation of [57] we denote the feasibility version of our problem with a single batching machine or several batching machines by $1|\text{p-batch}, b < n, r_j, p_j = p, c_j \leq d_j|-$ and $P|\text{p-batch}, b < n, r_j, p_j = p, c_j \leq d_j|-$, respectively. Here the first field specifies whether a single batching machine or several parallel batching machines exist. We include 'p-batch' in the second field to indicate that the jobs are processed in batches in accordance with the parallel batching discipline. The entry $b < n$ means that the batch capacity b is limited. Parameter r_j specifies release dates of the jobs, while $p_j = p$ means that the jobs have equal processing times. In addition, $c_j \leq d_j$ is included to denote that in a feasible schedule all jobs should meet their deadlines. The third field is empty since the objective is to find any feasible schedule (if one exists) without giving preference to any particular schedule.

The problem of minimizing the maximum lateness (where the values d_j are treated as due dates) is denoted similarly by $1|\text{p-batch}, b < n, r_j, p_j = p|L_{\max}$ if there is a single batching machine, and $P|\text{p-batch}, b < n, r_j, p_j = p|L_{\max}$ in the case of parallel p-batching machines.

Parallel batching problems with equal processing times have been studied since the eighties. The main focus of the early papers was on special cases with equal due dates or agreeable release and due dates.

If all due dates are equal, then $L_{\max}$-minimisation is equivalent to $c_{\max}$-minimisation. This problem can be solved in $O(n \log n)$ time by the algorithm due to Ikura and Gimple [50] known as *First-Only-Empty* (FOE) rule. It sequences the jobs in the order of their

release dates and generates full batches consisting of exactly b jobs except, perhaps, for the first batch which may have less jobs.

If the values r_j and d_j are agreeable, i.e. $r_1 \leq r_2 \leq \cdots \leq r_n, d_1 \leq d_2 \leq \cdots \leq d_n$, then the feasibility problem is solvable in $O(n^2)$ time by an algorithm due to the same authors [50]. The algorithm is based on the repeated application of a greedy procedure, which sequences as many jobs as possible in one batch. Another approach based on dynamic programming is due to Lee et al. [58]; it solves the problem in $O(nb)$ time. For the $L_{\max}$ -minimisation problem with agreeable release and due dates, Lee et al. [58] developed an $O(nb\log(np))$-time algorithm by combining their $O(nb)$ feasibility algorithm with binary search.

The general case of the batching problem with arbitrary r_j- and d_j-values was studied by Baptiste [10]. For the feasibility problem $1|\text{p-batch}, b < n, r_j, p_j = p, c_j \leq d_j|-$, an $O(n^8)$-time dynamic programming algorithm was developed; when combined with binary search, it solves the $L_{\max}$-minimisation problem $1|\text{p-batch}, b < n, r_j, p_j = p, d_j|L_{\max}$ in $O(n^8 \log n)$ time.

Many extended versions of the batching problem are NP-Hard. One example is the problem of scheduling batching machine with perishability time windows and incompatible job families presented in [26] where due dates are considered as strict deadlines. Such problem has been tackled by the design of a dynamic program and a greedy heuristic.

The results listed above deal with the bounded version of the p-batching problem with batch capacity $b < n$. The unbounded version of problem $1|\text{p-batch}, r_j, p_j = p|f$ with $b \geq n$ is studied in [28] where an $O(n^3)$-time algorithm is developed for minimizing an arbitrary regular objective function f.

In our research, we study the bounded version of the problem. We assume that all input data are integer. The algorithms we develop outperform those known previously. To achieve this, we use the idea of forbidden regions formulated in [39] for the classical problem with a single (non-batching) machine and the notion of barriers formulated in [93] for the case of parallel (non-batching) machines. We also discuss how the suggested algorithms can be generalized to solve the corresponding problems with precedence constraints. We distinguish between completion-start constraints (denoted by *prec*) traditionally studied in the scheduling theory and start-start constraints (denoted by *prec'*) which are useful in developing constraint propagation techniques, branch-and-bound and local search algorithms.

This chapter is organised as follows. In Section 2.2 we consider the single-machine batching problem. We start with an $O(n^2)$-time algorithm for solving the feasibility problem $1|\text{p-batch}, b < n, r_j, p_j = p, c_j \leq d_j|-$ (Section 2.2.1); we then explain how the al-

gorithm should be modified to solve the feasibility problem with precedence constraints (Section 2.2.2) and the L_{max}-minimisation problem (Section 2.2.3). Observe that our algorithms compare favourably with those previously known in terms of their worst-case running times: the algorithm for the feasibility problem has the same time complexity as the best algorithm developed for the case of agreeable r_j- and d_j-values [50] and it is faster than the $O(n^3)$-time algorithm for the unbounded version of the problem [28]. Our $O(n^2 \log n)$ algorithm for minimizing L_{max} solves, in strongly polynomial time, a more general problem in comparison with the earlier $O(nb\log(np))$-time algorithm developed for the case of agreeable release and due dates [58]. Furthermore, it is faster than the $O(n^8 \log n)$-time algorithm from [10] developed for arbitrary release and due dates and a range of objective functions.

In Section 2.3 we consider the problem with parallel batching machines. We show how the idea of barriers formulated in [93] for the case of parallel (non-batching) machines can be generalized to parallel batching machines solving the feasibility problem in $O(n^3 \log n)$ time and the L_{max}-minimisation problem in $O(n^3 \log^2 n)$ time. Finally, conclusions are discussed in Section 2.4.

2.2 Batch Scheduling with a Single Machine

In this section we study p-batching problems with a single machine where all time values are integer. Baptiste [10] developed a dynamic programming algorithm for the feasibility version $1|\text{p-batch}, b < n, r_j, p_j = p, c_j \leq d_j|-$, which runs in $O(n^8)$ time. Furthermore, he showed that problem $1|\text{p-batch}, b < n, r_j, p_j = p|f$ can be solved by applying the feasibility algorithm in conjunction with binary search to minimise a range of objective functions f including the weighted sum of completion times, total tardiness and maximum tardiness. The range of objective functions he consider is the reason of the higher time complexity of his algorithm in comparison to the ones presented in this chapter.

In what follows we develop a faster $O(n^2)$-time algorithm for the feasibility version of the problem (Section 2.2.1), adjust it for solving the problem with start-start precedence constraints (Section 2.2.2) and discuss an $O(n^2 \log n)$-time approach for solving the L_{max}-minimisation problem (Section 2.2.3).

2.2.1 Feasibility Problem

In this subsection we consider the feasibility problem $1|\text{p-batch}, b < n, r_j, p_j = p, c_j \leq d_j|-$ of finding a schedule meeting all deadlines where all time values are integer. The

corresponding single-machine problem $1|r_j, p_j = p, c_j \leq d_j|-$ in which batching is not allowed, is studied in [39] where an $O(n \log n)$-time algorithm is developed based on the idea of *forbidden regions*. We demonstrate how the concept of forbidden regions can be extended to the batching problem leading to an $O(n^2)$-time solution algorithm.

A set of forbidden regions $\mathcal{F}$ is a collection of time intervals with open left and right ends. Its fundamental property can be described as follows: in any feasible schedule with all jobs meeting their deadlines no batch can start in a forbidden region. Observe that a batch starting outside a forbidden region can continue its processing in a forbidden region.

The idea of the algorithm is to determine all forbidden regions for the given instance and then schedule the jobs in non-decreasing order of due dates such that no batch starts in a forbidden region.

Forbidden regions are determined considering for each release date r_i the latest time s_i at which the jobs released after r_i should start to not exceed the deadline. Clearly if s_i is less than r_i then no feasible schedule exists since there exists a job that cannot be scheduled after his release date without exceeding its deadline. If $r_i \leq s_i \leq r_i + p$ then any batch of jobs released before r_i and scheduled in the interval of time $[s_i - p, \ldots, s_i]$ would force later jobs to start after s_i and consequently to exceed their deadlines. Thus, interval $[s_i - p, \ldots, s_i]$ is declared forbidden.

Our algorithm consists of two stages: constructing forbidden regions (procedure FR) and constructing a feasible schedule which respects forbidden regions, release dates and deadlines (procedure FS). It can be formally described as follows.

Algorithm 'Batch-Scheduling'
Initialize $\mathcal{F} := \emptyset$;
Renumber the jobs in N so that $r_1 \leq r_2 \leq \ldots \leq r_n$;
FOR $i := n$ DOWNTO 1 DO
    Call $FR(r_i, \mathcal{F})$; // Calculate forbidden regions and store them in $\mathcal{F}$
END FOR
Call $FS(\mathcal{F})$; // Construct a schedule respecting the forbidden regions $\mathcal{F}$

Procedure FR is called n times, once for each release date, starting from the largest one. Each call $FR(r_i, \mathcal{F})$ either keeps the previously found set of forbidden regions $\mathcal{F}$ unchanged or adds a new forbidden interval with the right end at r_i. This is done by calling a backscheduling subroutine $BS([r_i, d_j], \mathcal{F}, N_{r_i, d_j})$ for all jobs j with $d_j \geq d_i$.

Procedure $FR(r_i, \mathcal{F})$ 'Find Forbidden Regions $\mathcal{F}$'
// It is assumed that $\mathcal{F}$ contains forbidden regions found so far by considering all jobs ℓ

// with $r_\ell > r_i$
Introduce an auxiliary numbering of the jobs so that $d_{j_1} \leq d_{j_2} \leq \cdots \leq d_{j_n}$;
$q := n$;
WHILE $d_{j_q} \geq d_i$ DO
    Define $N_{r_i,d_{j_q}} := \{u \mid r_i \leq r_u,\ d_u \leq d_{j_q}\}$;
    $s_{i,j_q} := BS\left([r_i,d_{j_q}],\mathscr{F},N_{r_i,d_{j_q}}\right)$; // latest time when the earliest task // from $N_{r_i,d_{j_q}}$ should start
    $q := q-1$;
END WHILE
Find $s_i := \min\{s_{i,j_q} \mid j_q$ is a job with $d_{j_q} \geq d_i\}$;
IF $s_i < r_i$ THEN
    Terminate declaring failure: no feasible schedule exists;
IF $r_i \leq s_i < r_i + p$ THEN
    Declare interval $(s_i - p, r_i)$ 'forbidden' and add it to the set $\mathscr{F}$;

The back-scheduling subroutine $BS([r_i,d_j],\mathscr{F},N_{r_i,d_j})$ considers a subset of jobs $N_{r_i,d_j} \subseteq N$ which have to start and complete in interval $[r_i,d_j]$, i.e.,

$$N_{r_i,d_j} = \{u \in N \mid r_u \geq r_i,\ d_u \leq d_j\}.$$

In order to find the latest time s_{ij} when the jobs N_{r_i,d_j} should start to be completed within the time interval $[r_i,d_j]$, the subroutine temporarily ignores individual values of release dates and deadlines for all $u \in N_{r_i,d_j}$ and schedules the jobs in N_{r_i,d_j} backwards from right to left into full batches of b jobs (except for the first batch which may have less jobs). Each batch is started as late as possible. If the starting time of a batch B_q falls in a forbidden region $(\lambda,\mu) \in \mathscr{F}$, then its starting time is shifted towards the beginning of that region, i.e., $s(B_q) = \lambda$. Thus, the resulting schedule respects all forbidden regions in $\mathscr{F}$ found earlier.

The back-scheduling procedure implementation is the main difference between our algorithm and the one in Garey et al. [39]. In fact, since limited capacity batches have to be created, our algorithm uses the First Only Empty (FOE) empty rule defined in Ikura and Gimple [50] instead of scheduling each job contiguously backwards from the due dates. Later in this section, we prove that the use of such rule is correct.

Procedure $BS([\alpha,\beta],\mathscr{F},N_{\alpha\beta})$ 'Backscheduling with Forbidden Regions'
// $N_{\alpha\beta}$ is a set of jobs to be scheduled in $[\alpha,\beta]$ ignoring their release dates and deadlines
Let s denote the starting time of the previously scheduled batch, initially $s = \beta$;

WHILE there are unscheduled jobs DO

// Form batches one by one starting from the last batch:

Define the starting time of the current batch as $s := s - p$;

IF s falls in a forbidden interval $(\lambda, \mu) \in \mathcal{F}$ THEN

Set $s := \lambda$;

Assign b unscheduled jobs from $N_{\alpha,\beta}$ to the current batch; if there are less than b unscheduled jobs, assign all of them to that batch.

END WHILE

RETURN s;

After procedure FR has called subroutine BS for all possible d_j with $d_j \geq d_i$, it analyses the earliest starting time

$$s_i := \min\{s_{ij} \mid j \text{ is a job with } d_j \geq d_i\}.$$

If s_i appears to be smaller than r_i, then there exists a subset of jobs N_{r_i,d_j} which cannot be processed in the interval $[r_i, d_j]$ and FR declares failure: no feasible schedule exists meeting the deadlines. Otherwise, i.e. if $s_i \geq r_i$, every subset of jobs N_{r_i,d_j} can be scheduled in the corresponding interval $[r_i, d_j]$. In this case each earlier batch should complete before time s_i. This means that if $s_i < r_i + p$, then the interval $(s_i - p,\ r_i)$ has to be declared forbidden and added to the set $\mathcal{F}$.

Implementation details of procedures FR and BS are discussed in the proof of Theorem 4. Notice that whenever a new forbidden interval is created, it is merged with an existing one, if their intersection is not empty. Therefore we may assume that all generated forbidden intervals are disjoint.

Suppose now that procedure FR has been called for all release dates, has not declared failure and produced forbidden regions $\mathcal{F}$, which are merged if their intersection is not empty. Then procedure $FS(\mathcal{F})$ is called to construct a feasible schedule with the set of forbidden regions $\mathcal{F}$ found.

Procedure $FS(\mathcal{F})$ schedules the jobs into batches from left to right respecting forbidden regions. At each decision point, it selects up to b available jobs giving priority to those with smallest deadlines and assigns them to one batch. A decision point corresponds to a job release date or a batch completion time that does not fall into a forbidden region. Observe that the right-end of any forbidden interval is given by a release date of some job. Formally procedure $FS(\mathcal{F})$ can be described as follows.

Procedure $FS(\mathscr{F})$ 'Forward Scheduling with Forbidden Regions $\mathscr{F}$'
$q := 1$; // current batch number
$B_q := \emptyset$; // B_q is the set of jobs of the batch q
WHILE there are unscheduled jobs DO
  IF $q = 1$ THEN
    $t_1 := 0$;
  ELSE
    Let t_1 be the completion time of batch B_{q-1};
  $t_2 := \min\{r_j \mid j \text{ is an unscheduled job}\}$;
  $t := \max\{t_1, t_2\}$;
  IF t belongs to some forbidden interval $(\lambda, \mu) \in \mathscr{F}$ THEN
    Set $t := \mu$
  Add to batch B_q no more than b jobs available at time t with the smallest deadlines, break ties in accordance with the job numbering;
  Remove the jobs assigned to B_q from the set of unscheduled jobs;
  $q := q + 1$;
END WHILE

The correctness of the formulated algorithm is proved in Theorems 1 and 3. Prior to presenting them, we introduce an illustrative example.

Example Consider an instance with $n = 12$ jobs which should be processed in batches of the maximum capacity $b = 3$ and the batch processing time is $p = 4$. Job release dates and deadlines are given in Table 2.1. Observe that the jobs are numbered in non-decreasing order of their release dates.

j	1	2	3	4	5	6	7	8	9	10	11	12
r_j	0	4	5	6	6	9	12	13	13	14	23	24
d_j	5	14	13	10	11	15	22	18	19	20	31	29

Table 2.1: Input data for an instance of problem $1|\text{p-batch}, b < n, r_j, p_j = p, c_j \leq d_j|-$

Starting with an empty set of forbidden intervals $\mathscr{F}$, procedure $FR(r_i, \mathscr{F})$ updates $\mathscr{F}$ as follows.

$FR(r_{12},\mathcal{F})$ considers $N_{r_{12},d_{11}} = \{12\}$ and finds $s_{12,11} = 27$,
$N_{r_{12},d_{12}} = \{12\}$ and finds $s_{12,12} = 25$.
This results in $s_{12} = 25$ and $\mathcal{F} = \mathcal{F} \cup \{(21,24)\}$.

$FR(r_{11},\mathcal{F})$ considers $N_{r_{11},d_{11}} = \{11,12\}$ and finds $s_{11,11} = 27$.
This results in $s_{11} = 27$ and no changes in $\mathcal{F}$.

$FR(r_{10},\mathcal{F})$ considers $N_{r_{10},d_{11}} = \{10,11,12\}$ and finds $s_{10,11} = 27$,
$N_{r_{10},d_{12}} = \{10,12\}$ and finds $s_{10,12} = 25$,
$N_{r_{10},d_7} = \{10\}$ and finds $s_{10,7} = 18$,
$N_{r_{10},d_{10}} = \{10\}$ and finds $s_{10,10} = 16$.
This results in $s_{10} = 16$ and $\mathcal{F} = \mathcal{F} \cup \{(12,14)\}$.

The subsequent calls of procedure $FR(r_i,\mathcal{F})$ lead to the following updates of $\mathcal{F}$:
$FR(r_9,\mathcal{F})$ finds $s_9 = 15$ and $\mathcal{F} = \mathcal{F} \cup \{(11,13)\}$.
$FR(r_8,\mathcal{F})$ finds $s_8 = 14$ and $\mathcal{F} = \mathcal{F} \cup \{(10,13)\}$.
$FR(r_7,\mathcal{F})$ finds $s_7 = 14$ and $\mathcal{F} = \mathcal{F} \cup \{(10,12)\}$.
$FR(r_6,\mathcal{F})$ finds $s_6 = 10$ and $\mathcal{F} = \mathcal{F} \cup \{(6,9)\}$.
$FR(r_5,\mathcal{F})$ finds $s_5 = 6$ and $\mathcal{F} = \mathcal{F} \cup \{(2,6)\}$.
$FR(r_4,\mathcal{F})$ finds $s_4 = 6$ and $\mathcal{F} = \mathcal{F} \cup \{(2,6)\}$.
$FR(r_3,\mathcal{F})$ finds $s_3 = 6$ and $\mathcal{F} = \mathcal{F} \cup \{(2,5)\}$.
$FR(r_2,\mathcal{F})$ finds $s_2 = 6$ and $\mathcal{F} = \mathcal{F} \cup \{(2,4)\}$.
$FR(r_1,\mathcal{F})$ finds $s_1 = 1$ and $\mathcal{F} = \mathcal{F} \cup \{(-3,0)\}$.

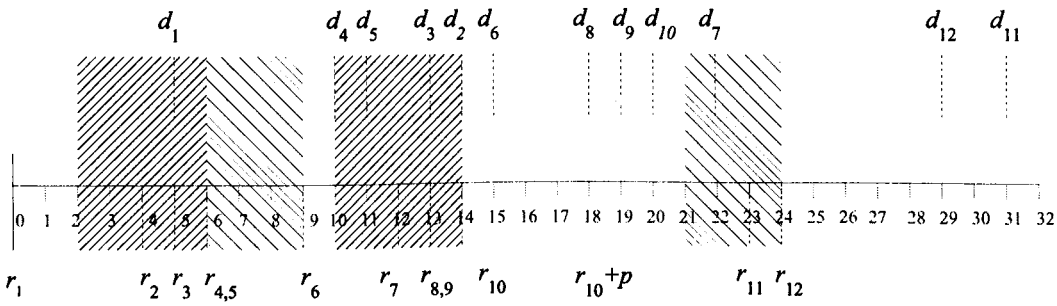

Figure 2.1: Forbidden regions $\mathcal{F} = \{(2,6),(6,9),(10,14),(21,24)\}$

The found forbidden regions are shown in Fig. 2.1. Observe that forbidden intervals $(2,6)$ and $(6,9)$ do not overlap, so that the forward scheduling procedure FS can select 6 as a starting time of a batch. This procedure generates a feasible schedule as follows.

- The first batch B_1 starts at time $t_1 = 0$ and it contains the only available job 1.

- The second batch B_2 starts at time $t = 6$ since it cannot start inside the forbidden interval $(2,6)$; procedure FS allocates jobs $\{3,4,5\}$ selected from the four available jobs $\{2,3,4,5\}$ based on the smallest deadlines.

- Next batch B_3 starts at time $t = 10$, which corresponds to the completion time of the previous batch and does not belong to $\mathscr{F}$; the batch contains two available jobs $\{2,6\}$.

- Next batch B_4 starts at time $t = 14$, which corresponds to the completion time of the previous batch and does not belong to $\mathscr{F}$; procedure FS allocates jobs $\{8,9,10\}$ selected from the four available jobs $\{7,8,9,10\}$ based on the smallest deadlines.

- Next batch B_5 starts at time $t = 18$, which corresponds to the completion time of the previous batch and does not belong to $\mathscr{F}$; the batch contains the only available job 7.

- The last batch B_6 starts at time $t = 24$ since it cannot start inside the forbidden interval $(21,24)$ and the the batch contains two available jobs $\{11,12\}$.

The Gantt chart of the resulting schedule is shown in Fig. 2.2.

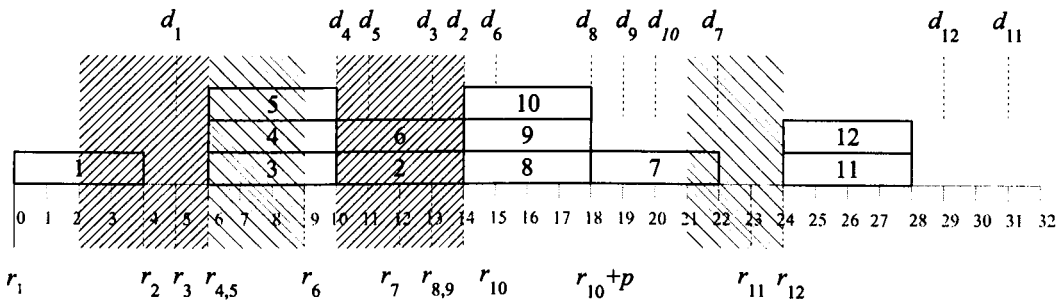

Figure 2.2: The schedule satisfying release dates, deadlines and forbidden regions $\mathscr{F}$

Now we turn to proving the correctness of algorithm 'Batch-Scheduling'. We start with a number of auxiliary statements first concluding with the main result formulated in Theorem 3.

First we show that the value s found by procedure BS is indeed the latest starting time for a subset $N_{\alpha,\beta}$.

Lemma 1. *Let s be the time calculated by procedure BS using the jobs in $N_{\alpha,\beta}$. If all jobs in $N_{\alpha,\beta}$ are grouped in batches which start strictly after time s, then at least one job from $N_{\alpha,\beta}$ is completed after time β.*

Proof. The backscheduling procedure *BS* arranges the jobs into batches in accordance with the First-Only-Empty policy [50] avoiding the given forbidden regions. It is known that this rule generates the minimum number of batches $\lceil |N_{\alpha,\beta}|/b \rceil$ necessary for a given set of jobs. Therefore the arguments of Lemma 3 from [39], which justify the correctness of algorithm *BS* for the single-machine case without batching, are applicable to the batching case. □

Before we proceed with procedure *FS*, we prove that procedure *FR* finds 'correct' forbidden regions. A forbidden interval $(\lambda,\mu) \in \mathscr{F}$ is *correct*, if in all feasible schedules no job can start at any time $s \in (\lambda,\mu)$.

Lemma 2. *Each forbidden interval found by* $FR(r_i,\mathscr{F})$ *is correct.*

Proof. To simplify the notation, we use d_j instead of d_{j_q} in this proof.

Let s be the minimum time s_{ij} found by procedure $BS\left(\left[r_i,d_j\right],\mathscr{F},N_{r_i,d_j}\right)$. Suppose $FR(r_i,\mathscr{F})$ finds a forbidden region $(s-p,r_i)$, but there exists a feasible schedule with all jobs meeting their deadlines and a job $k \notin N_{r_i,d_j}$ starting at time $s_k \in (s-p,r_i)$.

Consider the jobs in N_{r_i,d_j}. The release dates and deadlines of all jobs from N_{r_i,d_j} belong to $\left[r_i,d_j\right]$. Even if we relax individual job constraints and assume that all jobs from N_{r_i,d_j} have the same release date equal to r_i and the same deadline equal to d_j, then due to Lemma 1 they cannot be scheduled after job k without exceeding d_j since $c_k = s_k + p > s$. □

The next two theorems prove that algorithm 'Batch-Scheduling' is correct.

Theorem 1. *If procedure* $FR(r_i,\mathscr{F})$ *declares failure, then no feasible schedule with all jobs meeting their deadlines exists.*

Proof. Similar to the previous proof, we use d_j instead of d_{j_q}.

Suppose *FR* declares failure because subroutine $BS(\left[r_i,d_j\right],\mathscr{F},N_{r_i,d_j})$ finds the latest time s_{ij}, $s_{ij} < r_i$, when the jobs N_{r_i,d_j} should start in order to fit into interval $\left[r_i,d_j\right]$. Since procedure *BS* groups the jobs from N_{r_i,d_j} in the minimum number of batches relaxing individual job constraints on their release dates and deadlines, no feasible schedule exists for a subproblem defined by a subset of jobs N_{r_i,d_j}. □

Now we prove that in a schedule constructed by procedure *FS* all jobs meet their deadlines.

Theorem 2. *Procedure* $FS(\mathscr{F})$ *respect all release dates and all forbidden regions.*

Proof. This holds since the algorithm schedule a job strictly after its release date and, if the starting time fall in a forbidden regions, the processing of the job is postponed to first available time not in a forbidden region. □

Theorem 3. *Procedure $FS(\mathscr{F})$ finds a feasible schedule if one exists.*

Proof. We assume that $r_j \leq d_j - p$ for each job $j \in N$, since otherwise procedure $BS\left(\left[r_j, d_j\right], \mathscr{F}, N_{r_j,d_j}\right)$ declares failure when applied to interval $\left[r_j, d_j\right]$ with $j \in N_{r_j,d_j}$. Suppose an instance exists for which procedure $FS(\mathscr{F})$ generates a schedule $\mathscr{S}$ with some jobs completing after their deadlines. In this proof we show that in such a schedule either

(i) procedure FR would have declared failure, or

(ii) a batch would start in a forbidden region,

which are contradictions to the theorem 1 and the theorem 2 respectively.

Let us indicate with j the first late job in such an instance. In addition we assume that all idle intervals which appear after the smallest release date r_1 and before the starting time of job j are caused only by forbidden regions. Otherwise, if there is an idle interval $[\tau_1, \tau_2]$ before job j caused by a release date of some job and not by a forbidden region, then each job scheduled at or after τ_2 has a release date no smaller than τ_2 due to the properties of Algorithm FS. Therefore we can create a minimal counterexample by removing all jobs completed before τ_1. In the resulting counterexample the smallest release date is τ_2 and all idle intervals in-between τ_2 and the starting time of job j are caused only by forbidden regions.

Denote the batches that precede job j by $B_1, \ldots, B_v$ and the batch that contains job j by B_{v+1}. We distinguish two cases.

Case 1: *all jobs processed before j in the current schedule $\mathscr{S}$ have deadlines no larger than d_j and at least one of the batches processed before job j is non-full.*

Let B_z be the last non-full batch preceding job j and all batches $B_{z+1}, \ldots, B_v$ are full, see Fig. 2.3. Define job k as the one with the smallest release date among the jobs from $B_{z+1}, \ldots, B_v, B_{v+1}$. Since the jobs in $\mathscr{S}$ are scheduled by procedure FS, the release date of job k should satisfy

$$s(B_z) < r_k \leq s(B_{z+1}). \tag{2.1}$$

Indeed, if (2.1) does not hold, then there is at least one job (job k or another job with a smaller deadline) that should have been scheduled by FS in batch B_z. By definition of job k and by the assumption of Case 1, all jobs processed in the batches $B_{z+1}, \ldots, B_v$ together

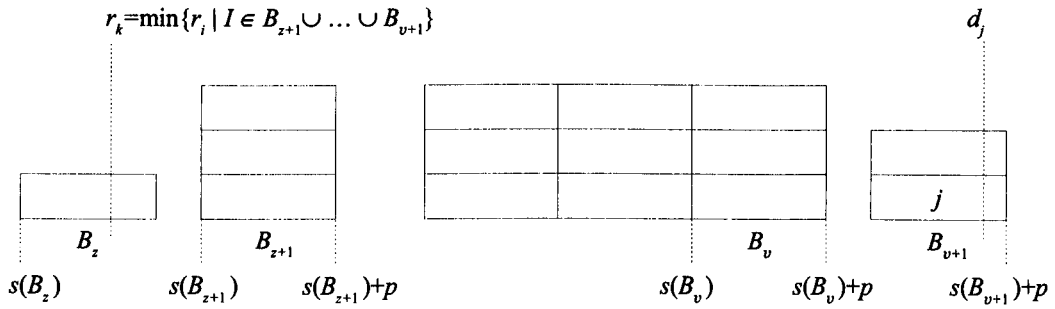

Figure 2.3: Schedule $\mathscr{S}$ considered in Case 1

with job j belong to the set N_{r_k,d_j}. Moreover, since the batches $B_{z+1},\ldots,B_v$ are full, the jobs in N_{r_k,d_j} require at least $v-z+1$ batches.

In the rest of the proof we will use the following auxiliary lemma.

Lemma 3. *If procedure* $BS([r_k,d_j],\mathscr{F},N_{r_k,d_j})$ *is called for the set* N_{r_k,d_j} *including all jobs from* $B_{z+1},\ldots,B_{v+1}$, *then it creates* $v-z+1$ *batches* $B'_{z+1},\ldots,B'_{v+1}$ *with starting times strictly less than those in the original schedule* $\mathscr{S}$*:*

$$s(B'_i) < s(B_i), \quad \textit{for all } i \textit{ with } z+1 \le i \le v+1.$$

Proof. Remind that procedure *BS* schedule jobs from right to left respecting the set of forbidden regions $\mathscr{F}$ which may be not empty. The proof is done by induction on the number of batches. Clearly, $s(B'_{v+1}) < s(B_{v+1})$ since $s(B'_{v+1}) = d_j - p$ is assigned to batch B'_{v+1} by procedure *BS*.

For the induction step assume that the statement holds for $B'_l,\ldots,B'_{v+1}$ with some l satisfying $z+1 < l \le v+1$, i.e.,

$$s(B'_i) < s(B_i), \quad \text{for all } i \text{ with } l \le i \le v+1. \tag{2.2}$$

Consider procedure *BS* scheduling the batch B'_{l-1}. If in the initial schedule $\mathscr{S}$ there is no idle time in-between batches B_{l-1} and B_l, then the fact that B'_l starts earlier implies that B'_{l-1} should start earlier than B_{l-1}. On the other hand, if B_{l-1} is separated from B_l by an idle time belonging to a forbidden region (λ,μ) in schedule $\mathscr{S}$, then batch B_{l-1} straddles λ, i.e., $s(B_{l-1}) \le \lambda < s(B_{l-1})+p$. Observe that the situation $\lambda = s(B_{l-1})+p$ is not possible, since in that case batch B_l would have been scheduled at time λ.

Since procedure *BS* respects forbidden regions and (2.2) holds for batch B'_l, then *BS* schedules batch B'_l at time λ or earlier so that B'_{l-1} completes at time λ or earlier. This completes the induction proof for Lemma 3. □

We now proceed with the proof of Theorem 3 showing that if procedure *BS* schedules the batch B'_{z+1} at an earlier time than B_{z+1} in schedule $\mathscr{S}$, then one of the contradictions (i) or (ii) holds.

Let s be the latest starting time calculated by procedure $BS([r_k,d_j],\mathscr{F},N_{r_k,d_j})$,

$$s = s(B'_{z+1}) < s(B_{z+1}).$$

Notice that at the time $BS([r_k,d_j],\mathscr{F},N_{r_k,d_j})$ is called, the set $\mathscr{F}$ contains all the correct forbidden regions with left end strictly larger than $r_k - p$. Since *BS* respects the forbidden regions and batch B_{z+1} starts earlier than in the original schedule $\mathscr{S}$, we have $s < s(B_z) + p$. Using condition (2.1) we obtain $s < r_k + p$. Thus, one of the following two cases happens:

- $s < r_k$ implies that jobs in $B_{z+1},\ldots,B_{v+1}$ cannot be completely processed in $[r_k,d_j]$. Then procedure *FR* should have declared failure, which is the contradiction (i);
- $r_k \le s < r_k + p$ implies that a forbidden region $(s-p,r_k)$ is created. Then batch B_z starts in that forbidden region in schedule $\mathscr{S}$, which is the contradiction (ii).

In the case all batches before B_{v+1} are full, then a similar argument can be applied. In particular, renumber the jobs in non-decreasing order of their release dates, if required, so that $k = 1$ and $B_{z+1} = 1$. Consider the application of procedure *BS* with the jobs N_{r_1,d_j}. Clearly the set N_{r_1,d_j} includes all jobs of the batches $B_1,\ldots,B_v$ together with job j. Using the same arguments as in the proof of Lemma 3 it is easy to verify that procedure *BS* schedules each batch strictly earlier than in the original schedule $\mathscr{S}$. Thus, since $s = s(B'_1)$ and $s < r_1$, we conclude that procedure *FR* would have declared failure, which is the contradiction (i).

Case 2: *at least one job processed before j has a larger deadline.*

Let i be the last job processed before job j with $d_i > d_j$, see Fig. 2.4. Denote the batch containing job i by B_x.

If there is a non-full batch processed in-between jobs i and j, then denote the latest batch with this property by B_z. All jobs of batches $B_z,\ldots,B_{v+1}$ have deadlines no larger than d_j and we arrive at Case 1 with batches $B_z,\ldots,B_{v+1}$.

If all batches $B_{x+1},\ldots,B_v$ processed in-between jobs i and j are full, then the fragment of the schedule from batch B_{x+1} until job j contains the jobs with deadlines no larger than d_j. If we consider Case 1 with z replaced by x, then following the same arguments we can define job k as the one with the smallest release date among the jobs from

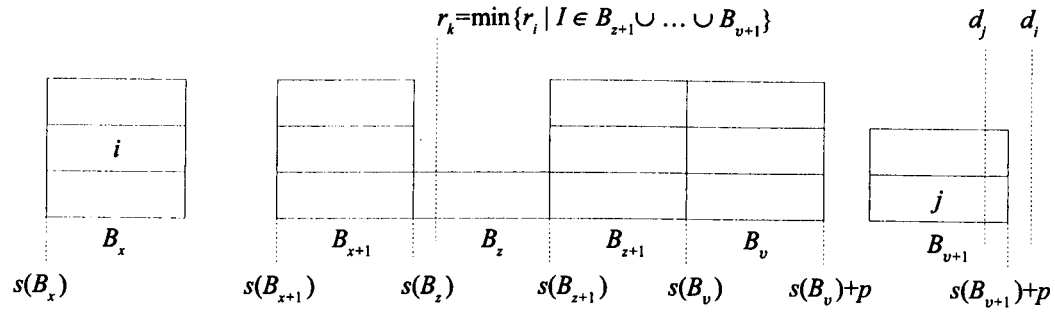

Figure 2.4: Schedule $\mathscr{S}$ for Case 3 with a non-full batch in-between i and j

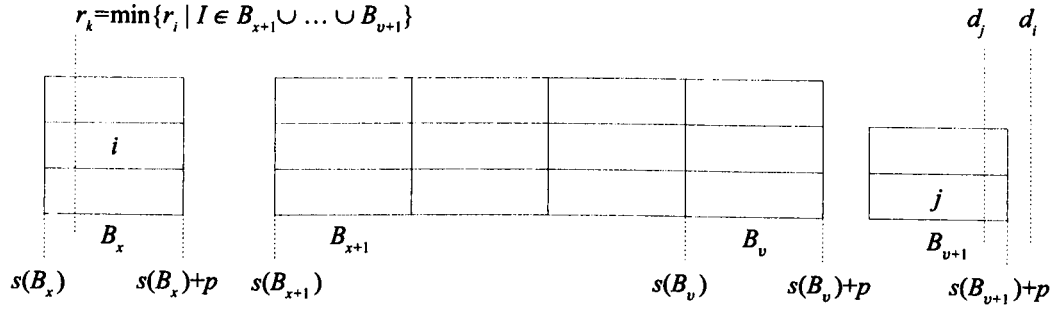

Figure 2.5: Schedule $\mathscr{S}$ for Case 3 with full batches in-between i and j

$B_{x+1}, \ldots, B_v, B_{v+1}$. Since the jobs in $\mathscr{S}$ are scheduled by procedure FS, the release date of job k should satisfy

$$s(B_x) < r_k \leq s(B_{x+1}); \tag{2.3}$$

otherwise job k should have been scheduled in batch B_x instead of job i due to $d_k < d_i$. Observe that condition (2.3) is the analogue of (2.1). The arguments used in the remaining part of the proof of Case 1 are applicable now, so that Lemma 3 with z replaced by x is also valid in Case 3 and therefore one of the contradictions (i) or (ii) holds. □

Theorem 4. *Problem* $1|p\text{-}batch, b < n, r_j, p_j = p, c_j \leq d_j|-$ *is solvable in* $O(n^2)$ *time.*

Proof. The correctness of algorithm 'Batch-Scheduling' follows from Theorems 1 and 3. Procedure $FR(r_i, \mathscr{F})$ is called no more than n times for the different values r_i. Each call of $FR(r_i, \mathscr{F})$ incurs $O(n)$ calculations of the sets $N_{r_i,d_{j_q}}$ and $O(n)$ calls of procedure BS.

We first estimate the time needed to find the sets $N_{r_i,d_{j_q}}$. The first call of procedure BS considers the largest possible time interval $[r_i, d_{j_n}]$ and selects the relevant jobs $N_{r_i,d_{j_n}}$ in $O(n)$ time. For the further calls with time intervals $[r_i, d_{j_{q-1}}]$, $q \leq n$, some jobs from $N_{r_i,d_{j_q}}$ should be removed. Due to the numbering $d_{j_1} \leq d_{j_2} \leq \cdots \leq d_{j_n}$, this can be done in no more than $|N_{r_i,d_{j_n}}|$ steps for all calls of FR with the fixed r_i. Thus all calculations of the sets $N_{r_i,d_{j_q}}$ for all values $r_i, i = n, n-1, \ldots, 2, 1$ can be done in $O(n^2)$ time.

Consider now the time complexity of procedure *BS*. Every initial call of this procedure with the largest release date r_n and the set of jobs $N_{r_n,d_{j_n}}$, $N_{r_n,d_{j_{n-1}}},\ldots,N_{r_n,d_{j_q}}$ can be implemented in $O(1)$ time since the set of forbidden intervals is empty at that stage. In order to implement the subsequent calls of procedure *BS* efficiently, one can keep records of the earlier calls. Note that each call to procedure $FR(r_i,\mathscr{F})$ creates at most one forbidden region $(s_i-\tau,r_i)$. Since the inclusions

$$[r_i,d_{j_q}]\subseteq[r_{i-1},d_{j_q}] \quad \text{and} \quad N_{r_i,d_{j_q}}\subseteq N_{r_{i-1},d_{j_q}} \tag{2.4}$$

hold, at most one new forbidden region has to be considered when scheduling the jobs from $N_{r_{i-1},d_{j_q}}$. Therefore, procedure $BS([r_{i-1},d_{j_q}],\mathscr{F},N_{r_{i-1},d_{j_q}})$ can use the partial schedule found earlier by procedure $BS([r_i,d_{j_q}],\mathscr{F},N_{r_i,d_{j_q}})$ to add the jobs from $N_{r_{i-1},d_{j_q}}\setminus N_{r_i,d_{j_q}}$. Thus, all calls of procedure *BS* with the fixed value d_{j_q} and different values r_n, $r_{n-1},\ldots,r_1$ can be implemented in no more than $|N_{r_1,d_{j_q}}|$ steps, so that the overall time complexity of all calls of procedure *BS* is $O(n^2)$.

Finally notice that, since the forbidden regions are created sequentially from right to left, overlapping forbidden regions can be detected and merged at the time of their creation with a constant time complexity. This completes the proof. □

An interesting property of a feasible schedule constructed by the algorithm with forbidden regions is its optimality with respect to two other criteria: the makespan $C_{\max}=\max_{j\in N}\{c_j\}$ and the sum of completion times $\sum_{j\in N}c_j$. This can be shown following the same arguments as in [39].

In addition, it is easy to verify a useful property which holds for a pair of jobs with agreeable release dates and deadlines.

Proposition 1. *If two jobs $i,j\in N$ with $i<j$ have agreeable release dates and deadlines, i.e., $r_i\le r_j$, $d_i\le d_j$, then algorithm 'Batch-Scheduling' finds a solution with job j starting not earlier than job i, i.e. $s_i\le s_j$.*

The correctness of the above proposition immediately follows from the behavior of procedure $FS(\mathscr{F})$ which schedules available jobs in non-decreasing order of deadlines.

2.2.2 Feasibility Problem with Precedence Constraints

In this subsection we study the p-batching problem with two types of precedence relations: completion-start and start-start relations defined as follows. If job i precedes job j in terms of the completion-start relation, denoted by *prec*, then j can start only after i is completed,

i.e. $s_j \geq c_i$. If job i precedes job j in terms of the start-start relation, denoted by $prec'$, then j cannot start earlier than i, i.e. $s_j \geq s_i$.

Completion-start relations are the most popular precedence constraints and are well-studied in the scheduling literature. However, start-start relations are of importance when several jobs can be processed in parallel either by a batching machine or by multiple machines. Such relations are especially useful in connection with algorithms which use list scheduling techniques. As we show in this subsection, the complexity status of the two versions of our batching problem with completion-start and start-start relations is different.

Proposition 2. *Problem* $1|prec, r_j, p_j = p, c_j \leq d_j|-$ *with a single non-batching machine and completion-start precedence relations is solvable in* $O(n \log n)$ *time.*

The above result is due to Garey et al. [39]. Observe that Proposition 2 is valid for the general case of the problem when jobs are released at different times and have non-equal deadlines.

Proposition 3. *Problem* $1|p\text{-}batch, b < n, prec, p_j = 1, c_j \leq d|-$ *with a single p-batching machine and completion-start precedence relations is strongly NP-hard even if all jobs are released simultaneously, have unit processing times and a common deadline* d.

The correctness of the above proposition follows from the equivalence of the p-batching problem $1|\text{p-batch}, b < n, prec, p_j = 1, c_j \leq d|-$ and problem $P|prec, p_j = 1, c_j \leq d|-$ with b identical parallel (non-batching) machines and unit-time jobs which start at integer times. The NP-hardness in the strong sense of the latter problem was established by Ullman [97].

We now turn to the start-start precedence relations and describe an algorithm 'Batch-Scheduling with Start-Start Relations' which solves problem $1|\text{p-batch}, b < n, prec', r_j, p_j = p, c_j \leq d_j|-$ in two steps. In the first step, a preprocessing is done by modifying the release dates and deadlines of the jobs in accordance with the given precedence relations; in the second step algorithm 'Batch-Scheduling' finds a feasible schedule which respects the precedence relations and the given deadlines, if such a schedule exists.

The preprocessing procedure is typical for problems with precedence relations, see, e.g., [19, 56]. The main idea is to modify release dates and deadlines of a problem instance so that the algorithm 'Batch-Scheduling' applied to the modified instance returns a solution which respects the precedence constraints. In particular, if job i should precede job j (denoted by $i \overset{prec'}{\rightarrow} j$), we can alter release dates and deadlines of i and j in order to exploit the property of Proposition 1, replacing r_j by $\max\{r_i, r_j\}$ and d_i by $\min\{d_i, d_j\}$.

Scanning all jobs in a topological order (i.e., such that $i < j$ for $i \overset{prec'}{\rightarrow} j$), modified release dates and deadlines can be calculated in $O(n^2)$.

Proposition 4. *Algorithm 'Batch-Scheduling with Start-Start Relations' solves problem* $1|p\text{-}batch, b < n, prec', r_j, p_j = p, c_j \leq d|_j-$ *with a single p-batching machine and start-start precedence relations in* $O(n^2)$ *time.*

Proof. For all $j \in N$ denote the modified release dates and deadlines by r'_j and d'_j, respectively. We show that

(a) if algorithm 'Batch-Scheduling with Start-Start Relations' does not construct a feasible schedule, then none exists,

(b) if a feasible schedule which respects r'_j and d'_j is found, then it respects r_j and d_j for all jobs $j \in N$, as well as the precedence constraints.

Claim (a) holds due to the fact that the constraints imposed by the modified release dates and deadlines are "weaker" than the precedence constraints: the adjustments do not allow the first job i from the pair $i \overset{prec'}{\rightarrow} j$ to be completed after the deadline d_j of the second job or the second job j to start before the release date r_i of the first job. Then in accordance with Theorem 1, if algorithm 'Batch-Scheduling', which is used at the second stage of 'Batch-Scheduling with Start-Start Relations', declares failure, then no schedule exists for the modified data r'_j and d'_j, and therefore no schedule exists for a "more constrained" problem with precedence relations.

To demonstrate that claim (b) holds, consider a schedule found by the algorithm for the modified data r'_j and d'_j. Since the preprocessing stage leads to agreeable release dates and deadlines with $r'_i \leq r'_j$ and $d'_i \leq d'_j$ for all $i \overset{prec'}{\rightarrow} j$, then due to Proposition 1, in the schedule found at the second stage by algorithm 'Batch-Scheduling' we have $s_i \leq s_j$ for all $i \overset{prec'}{\rightarrow} j$, i.e. each precedence constraint is observed.

The time complexity of algorithm 'Batch-Scheduling with Start-Start Relations' is $O(n^2)$ since the calculation of modified release dates and deadlines can be done in $O(n^2)$ time and algorithm 'Batch-Scheduling' of the second stage requires $O(n^2)$ time as well. □

2.2.3 Minimizing the Maximum Lateness

In this subsection we discuss how the $O(n^2)$-time algorithm 'Batch-Scheduling', which solves the feasibility problem $1|\text{p-batch}, b < n, r_j, p_j = p, c_j \leq d_j|-$, can be used for solving the $L_{\max}$-minimisation problem $1|\text{p-batch}, b < n, r_j, p_j = p|L_{\max}$. We consider a series

of feasibility problems $1|\text{p-batch}, b < n, r_j, p_j = p, c_j \leq d_j + \delta|-$ with due dates of all jobs increased by an offset δ; the smallest value δ^* for which a feasible schedule exists meeting all d_j-values defines the optimal $L_{\max}$-value. Since $L_{\max}$ can be as large as $r_{\max} + np$ in the worst case, where $r_{\max}$ is the largest release date, the straightforward application of a binary search approach would incur $O(\log(r_{\max} + np))$ calls to algorithm 'Batch-Scheduling' resulting in an algorithm with time complexity $O(n^2 \log(r_{\max} + np))$, which is polynomial but not strongly polynomial.

Using the property that all possible values of completion times belong to the set

$$\mathscr{C} := \{r_j + \alpha p \mid j \in N,\ \alpha \in \{1,2,\ldots,n\}\},$$

a strongly polynomial algorithm can be derived by considering trial values $\delta \in \mathscr{D}$, where

$$\mathscr{D} := \{r_j + \alpha p - d_i \mid i,j \in N,\ \alpha \in \{1,2,\ldots,n\}\}, \tag{2.5}$$

see Baptiste [10]. Since $|\mathscr{D}| = O(n^3)$, binary search over set $\mathscr{D}$ incurs $O(\log n)$ calls to algorithm 'Batch-Scheduling'. Thus, the overall time complexity is dominated by the generation and sorting of the set $\mathscr{D}$ requiring $O(n^3 \log n)$ time.

In what follows we describe a faster approach which considers only a subset of "appropriate" elements of $\mathscr{D}$, so that the overall time complexity of the resulting approach is $O(n^2 \log n)$. The main idea is to consider sequentially three 'layers' of $\mathscr{D}$-values with

- $\mathscr{D}'$ consisting of $O(n^2)$ elements $\{\delta'\}$ spreading over the whole range of $\mathscr{D}$-values,
- $\mathscr{D}''$ consisting of $O(n)$ elements $\{\delta'' \mid \delta'' \in (\delta'_l, \delta'_r]\} \cup \{\delta'_l, \delta'_r\}$, where δ'_l and δ'_r are two elements of $\mathscr{D}'$,
- $\mathscr{D}'''$ consisting of $O(n^2)$ elements $\{\delta''' \mid \delta''' \in (\delta''_u, \delta''_v]\}$, where δ''_u and δ''_v are two elements of $\mathscr{D}''$ and $(\delta''_u, \delta''_v] \subseteq (\delta'_l, \delta'_r]$.

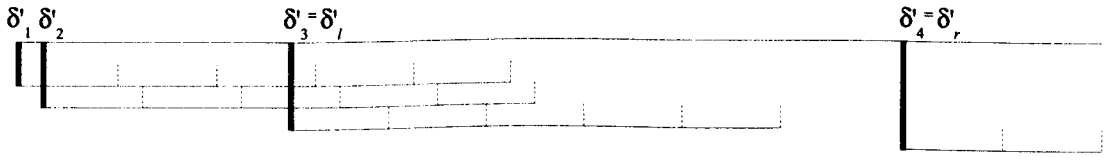

Figure 2.6: δ-values in the layer $\mathscr{D}'$

The first layer $\mathscr{D}' \subseteq \mathscr{D}$ is defined by relation (2.5) with $\alpha = 1$. In addition, we also

include the largest possible element from $\mathscr{D}$ defined as

$$\delta_{\max} = \max\{r_j \mid j \in N\} + np - \min\{d_j \mid j \in N\},$$

which is needed as the right boundary for binary search:

$$\mathscr{D}' = \{r_j + p - d_i \mid i, j \in N\} \cup \{\delta_{\max}\}, \text{ see Fig. 2.6.}$$

Clearly, such a set contains $O(n^2)$ distinct values which can be generated and sorted in $O(n^2 \log n)$ time. Using binary search combined with algorithm 'Batch-Scheduling', we find in $O(n^2 \log n)$ time the largest value δ_l' for which no feasible schedule exists and the smallest value δ_r' for which a feasible schedule exists. Since $\delta^* \in (\delta_l', \delta_r']$, the next layer is generated within these boundary values.

The appropriate values of $\mathscr{D}''$ are defined on the basis of the left boundary δ_l' by adding the offset αp; in addition set $\mathscr{D}''$ also includes δ_l' and δ_r':

$$\mathscr{D}'' = \{\delta'' = \delta_l' + \alpha p \mid \delta'' \in (\delta_l', \delta_r'],\ 1 \le \alpha < n\} \cup \{\delta_l', \delta_r'\}, \text{ see Fig. 2.7.}$$

Observe that δ_l' is of the form $r_j + p - d_i$ and, therefore, offset αp is defined for $1 \le \alpha < n$.

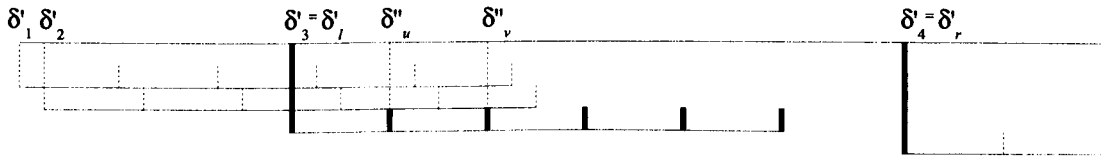

Figure 2.7: δ-values in the layer $\mathscr{D}''$

Clearly, the set $\mathscr{D}''$ contains at most $n+1$ values which can be generated and sorted in $O(n \log n)$ time. Using binary search combined with algorithm 'Batch-Scheduling', the appropriate boundary values δ_u'', δ_v'' for the second layer can be found in $O(n^2 \log n)$ time, the optimum value δ^* is guaranteed to satisfy

$$\delta^* \in (\delta_u'', \delta_v''] \subseteq (\delta_l', \delta_r']. \tag{2.6}$$

Notice that if $\delta_u'' = \delta_l' + (n-1)p$ as shown in Fig. 2.8, then using the fact that δ_l' and δ_r' are the two nearest values in $\mathscr{D}'$ we conclude that no value from $\mathscr{D}$ lies in-between δ_u'' and δ_v''. Therefore, due to (2.6) the optimal value of $L_{\max}$ corresponds to $\delta^* = \delta_v''$.

In all other cases we have

$$\delta_v'' - \delta_u'' \le p. \tag{2.7}$$

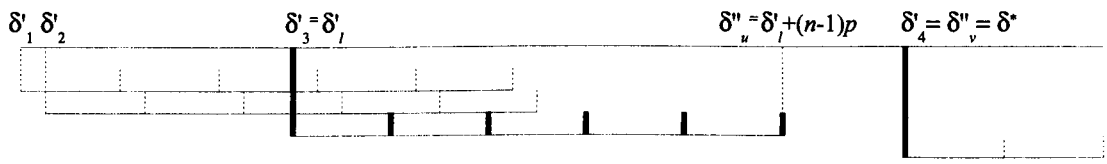

Figure 2.8: δ-values in the layer $\mathscr{D}'''$ when $\delta''_u = \delta'_l + (n-1)p$

Observe that the strict inequality may happen only if $\delta''_v = \delta'_r$; in all other cases δ''_u and δ''_v both belong to $\mathscr{D}''\backslash\{\delta'_r\}$ and the distance between two consecutive elements from this set is p.

For δ''_u and δ''_v satisfying (2.7), the search continues in the third layer $\mathscr{D}'''$ defined as

$$\mathscr{D}''' = \{\delta''' = \delta'_i + \alpha p \mid \delta'_i \in \mathscr{D}',\ \delta''' \in (\delta''_u, \delta''_v],\ 1 \le \alpha < n\},\ \text{see Fig. 2.9.}$$

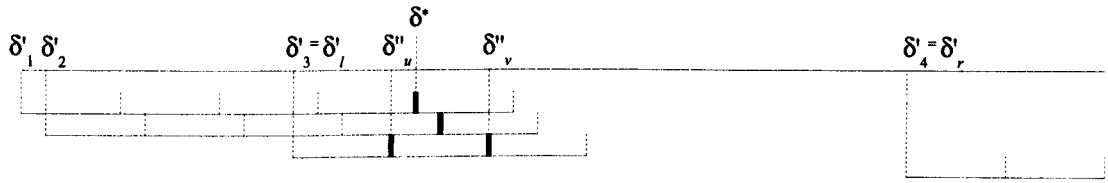

Figure 2.9: δ-values in the layer $\mathscr{D}'''$ when $\delta''_u < \delta'_l + (n-1)p$

Due to (2.7), each element $\delta'_i \in \mathscr{D}'$ can generate at most one element $\delta''' = \delta'_i + \alpha p$, which can be found in $O(1)$ time as the solution to the inequality

$$\delta''_u \le \delta'_i + \alpha p \le \delta''_v.$$

Therefore, there are $O(n^2)$ values δ'''_i which can be generated and sorted in $O(n^2 \log n)$ time. Using binary search combined with algorithm 'Batch-Scheduling', the optimum solution δ^* can be found among elements $\mathscr{D}'''$ in $O(n^2 \log n)$ time.

Thus, we have proved the following result.

Theorem 5. *Problem* $1|p\text{-batch}, b < n, r_j, p_j = p|L_{\max}$ *can be solved in* $O(n^2 \log n)$ *time by combining the feasibility algorithm 'Batch-Scheduling' and specially arranged binary search.*

2.3 Batch Scheduling with Parallel Machines

In this section we consider the problems $P|\text{p-batch}, b < n, r_j, p_j = p, c_j \le d_j|-$ and $P|\text{p-batch}, b < n, r_j, p_j = p|L_{\max}$ with parallel machines. We formulate two algorithms

with complexities $O(n^3 \log n)$ and $O(n^3 \log^2 n)$ based on the ideas of Simons [93] who developed an algorithm for solving the parallel machine problem $P|p_j = p, r_j, c_j \leq d_j|-$ in which batching is not allowed.

Using similar arguments as in Lemma 1 from [93] (see also Lemma 5.1 from [19]), it is easy to demonstrate that an optimal schedule can be found in the class of *cyclic list schedules* with the m earliest batches $B_1, \ldots, B_m$ assigned to machines $M_1, \ldots, M_m$, the next m batches $B_{m+1}, \ldots, B_{2m}$ assigned to machines $M_1, \ldots, M_m$, etc. If the batches are numbered in accordance with their starting times, then machine M_k processes batches B_k, B_{k+m}, B_{k+2m}, etc.

Suppose a partial schedule consisting of batches $B_1, B_2, \ldots, B_z, \ldots, B_{q-1}$ with $q - 1 \geq m$ has been constructed, and U is the set of remaining unscheduled jobs. In order to form the next batch B_q we determine its starting time t taking into account the fact that B_q cannot overlap with the previous batch B_{q-m} on the same machine and at least one unscheduled job should be released by that time. Then the algorithm tries to schedule at most b jobs at time t where preference is given to the available jobs with the smallest deadlines. If it is not possible to schedule a job before its deadline, a 'Crisis' subroutine is called. This happens if an available job i does not meet its deadline, i.e. $d_i < t + p$. Since an available job with the smallest deadline is always selected first, the crisis occurs when an attempt is made to schedule the first job in a new batch. The crisis subroutine backtracks over the current partial schedule searching for the latest batch B_z which satisfies at least one of the following two conditions:

(i) is incomplete (i.e. has less than b jobs),

(ii) contains a job u with a deadline larger than that of the crisis job i.

If no such batch exists, then the subroutine concludes that there is no feasible schedule and halts. Otherwise (i.e., if the required batch B_z is found) the subroutine initiates rescheduling of the jobs from batch B_z and the subsequent batches $B_{z+1}, \ldots, B_{q-1}$. As a result,

(i) the earlier incomplete batch B_z accommodates more jobs, or

(ii) job u is moved to a later batch leaving room for a job with a smaller deadline.

To perform re-scheduling, the crisis subroutine creates an additional restriction called *barrier*(B_z, r) that prohibits batch B_z to start earlier than time r, where the value of r is set to a value larger than the previous starting time of batch B_z (it equals the smallest release date of a job from $B_{z+1} \cup \cdots \cup B_{q-1} \cup \{i\}$). The jobs from $B_z, B_{z+1}, \ldots, B_{q-1}$ are claimed

unscheduled by adding them to the set of unscheduled jobs U and the algorithm continues with the partial schedule $B_1, B_2, \ldots, B_{z-1}$.

In what follows we provide a formal description of the algorithm 'Batch Scheduling with Barriers' together with the subroutines it uses.

Algorithm 'Batch-Scheduling with Barriers'

$q := 1$; // current batch number
$B_q := \emptyset$; // set of jobs of the current batch B_q
barrierlist$:= \emptyset$; // a set of barriers (B_z, r) containing batches and restrictions on their
// starting times (batch B_z is not allowed to start earlier than time r)
WHILE there are unscheduled jobs in U DO
  IF $q \leq m$ THEN
    Select machine M_q;
    $t_1 := 0$;
  ELSE
    Select machine $M_{q \bmod m}$; // where $M_0 = M_m$
    $t_1 := s(B_{q-m}) + p$;
  ENDIF
  $t_2 := \min\{r_j \mid j \in U \text{ is an unscheduled job}\}$;
  $t_3 := \max(\{r \mid (B_v, r) \text{ is a barrier; } 1 \leq v \leq q\} \cup \{0\})$;
  $t := \max\{t_1, t_2, t_3\}$;
  Select at most b jobs with the smallest deadlines available at time t;
  IF a job i with the smallest deadline satisfies $d_i \geq t + p$ THEN
    Schedule all selected jobs in batch B_q on the selected machine, i.e. start
    them at time $s(B_q) := t$ and remove them from the set of unscheduled jobs U;
    $q := q + 1$;
  ELSE
    Call Crisis(i);
END WHILE

Procedure Crisis(i)
Find the latest incomplete batch B_λ, $\lambda < q$, with less than b jobs;
Find the latest batch B_μ, $\mu < q$, with a job u such that $d_u > d_i$ (B_μ can be complete);
IF no batch B_λ or B_μ exists THEN
  Terminate declaring failure: no feasible schedule exists
ELSE
  $z := \max\{\lambda, \mu\}$; // Batch B_z and subsequent batches should be re-scheduled
  Determine $r = \min\{\{r_j \mid j \in B_{z+1}, \ldots, B_{q-1}\} \cup r_i\}$;
  Add (B_z, r) to *barrierlist*;
  Add the jobs from $B_z \cup \cdots \cup B_{q-1}$ to the set of unscheduled jobs U;
  Set $B_z = B_{z+1} = \ldots = B_{q-1} := \emptyset$;
  $q := z$;
ENDIF

We illustrate the performance of the algorithm 'Batch Scheduling with Barriers' by the following example.

Example Consider an instance with $n = 12$ jobs which should be processed in batches of maximum capacity $b = 2$ with processing time $p = 7$ on $m = 2$ machines. Job release dates and deadlines are given in Table 2.2. Observe that the jobs are numbered in non-decreasing order of their release dates.

j	1	2	3	4	5	6	7	8	9	10	11	12
r_j	0	0	0	5	6	6	6	6	6	15	18	18
d_j	14	14	14	13	24	24	24	29	27	25	32	32

Table 2.2: Input data for an instance of problem $P|\text{p-batch}, b < n, r_j, p_j = p, c_j \leq d_j|-$

A trace of the algorithm *'Batch-Scheduling with Barriers'* is given below.

$q = 1$: batch $B_1 = \{1,2\}$ is allocated to machine M_1 at time $t = 0$.
$q = 2$: batch $B_2 = \{3\}$ is allocated to machine M_2 at time $t = 0$.
$q = 3$: batch $B_3 = \{4\}$ is allocated to machine M_1 at time $t = t_1 = s(B_1) + p = 7$.

Job 4 is late. Procedure *Crisis*(4) is called and barrier $(B_2, 5)$ is added to *barrierlist*.

$q = 2$: batch $B_2 = \{3,4\}$ is allocated to machine M_2 at time $t = t_3 = 5$.

$q = 3$: batch $B_3 = \{5,6\}$ is allocated to machine M_1 at time $t = t_1 = s(B_1) + p = 7$.
$q = 4$: batch $B_4 = \{7,8\}$ is allocated to machine M_2 at time $t = t_1 = s(B_2) + p = 12$.
$q = 5$: batch $B_5 = \{9\}$ is allocated to machine M_1 at time $t = t_1 = s(B_3) + p = 14$.
$q = 6$: batch $B_6 = \{10\}$ is allocated to machine M_2 at time $t = t_1 = s(B_4) + p = 19$.

Job 10 is late. Procedure *Crisis*(10) is called and barrier $(B_5, 15)$ is added to *barrierlist*.

$q = 5$: batch $B_5 = \{9,10\}$ is allocated to machine M_1 at time $t = t_3 = 15$.
$q = 6$: batch $B_6 = \{11,12\}$ is allocated to machine M_2 at time $t = t_1 = s(B_4) + p = 19$.

The Gantt chart of the resulting feasible schedule is shown in Fig. 2.10.

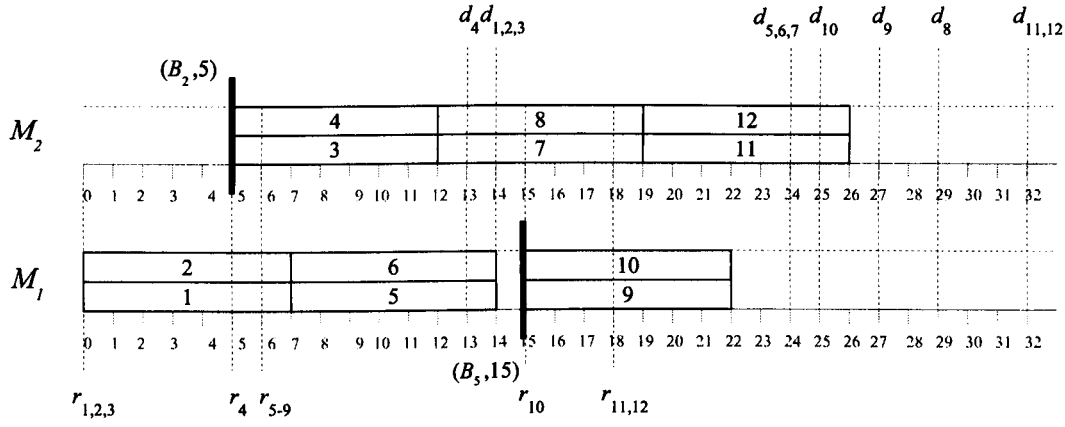

Figure 2.10: A feasible schedule satisfying release dates and deadlines for the p-batching problem with $m = 2$ parallel machines

We now prove the correctness of the algorithm 'Batch Scheduling with Barriers' by demonstrating that each barrier is correct and that if the algorithm does not provide a feasible schedule meeting all deadlines, then none exists. A barrier (B_z, r) is *correct* if in all feasible schedules batch B_z cannot start before time r.

Let $\mathscr{B}$ be a barrier list. A feasible $\mathscr{B}$-schedule is a feasible schedule for problem $P|\text{p-batch}, b < n, p_j = p, r_j, c_j \leq d_j|-$ with the property that if $(B_z, r) \in \mathscr{B}$, then batch B_z does not start before time r.

We introduce the algorithm 'Batch-Scheduling with Barriers($\mathscr{B}$)' by replacing the initialization statement '*barrierlist*:= $\emptyset$' by '*barrierlist*:= $\mathscr{B}$'.

Suppose that at some stage of the algorithm 'Batch-Scheduling with Barriers' we have constructed a partial schedule of batches $B_1, \ldots, B_{q-1}$ consisting of jobs $J \subset N$ and

the corresponding set of barriers is $\mathscr{B}$. Then we have the following property: If 'Batch-Scheduling with Barriers($\mathscr{B}$)' is applied to the set of jobs J, then it constructs the same schedule as algorithm 'Batch-Scheduling with Barriers' without any call of the crisis subroutine.

Lemma 4. *Let $\mathscr{B}$ be a set of barriers. Assume that algorithm 'Batch-Scheduling with Barriers($\mathscr{B}$)' has scheduled jobs J and calculated time t as a possible starting time for the next batch B_q. Then*

(i) there is no feasible $\mathscr{B}$-schedule of jobs J in which any of the batches B_ν ($\nu = 1,\ldots,q-1$) is scheduled before time $s(B_\nu)$;

(ii) there is no feasible $\mathscr{B}$-schedule in which B_q starts before time t.

Proof. We prove this lemma by induction on q. If (ii) holds for all $\nu \leq q-1$, then (i) holds. Thus, it remains to prove (ii) for $\nu = q$. Let t be the current time after jobs J are scheduled.

If $t = t_1$, then either $t = 0$ or $t = s\left(B_{q-1}\right) + p$. In the former case, B_q trivially cannot start earlier. In the latter case, by assumption, $s\left(B_{q-1}\right) + p$ is the earliest possible time at which batch B_{q-1} can finish and, therefore, the earliest possible starting time of B_q.

If $t = t_2$, then B_q is started at the minimum release date of the unscheduled jobs. Thus, t_2 is indeed the earliest possible starting time of B_q in any feasible schedule.

If $t = t_3$, then a barrier constraint is preventing B_q from starting earlier. □

Lemma 5. *Each barrier created by algorithm 'Batch-Scheduling with Barriers' is correct.*

Proof. Assume that the set $\mathscr{B}$ consists of $h-1$ barriers and that they are correct, the algorithm has constructed a partial schedule consisting of batches $B_1,\ldots,B_{q-1}$ at the time of the h-th crisis and that (B_z, r) is the h-th barrier to be created. Let t be the current time after the first $q-1$ batches are scheduled with respect to the first $h-1$ correct barriers and let ℓ be the crisis job so that $d_\ell < t + p$. Furthermore, the following two properties hold:

(a) for each job j assigned earlier to $B_{z+1} \cup \cdots \cup B_{q-1}$ we have $d_j \leq d_\ell < t + p$;

(b) each of the batches $B_{z+1},\ldots,B_{q-1}$ has exactly b jobs.

Suppose to the contrary that there is a feasible schedule in which batch B_z is scheduled before time r. Since r is the minimum release date of all jobs assigned earlier to $B_{z+1} \cup \cdots \cup B_{q-1} \cup \{\ell\}$, they cannot be assigned to batch B_z. Due to (b) and taking into account

job ℓ, after the first z batches we have to schedule $(q-z-1)b+1$ jobs which require at least $q-z$ batches. Thus, at least one job should be scheduled in batch B_q.

By applying Lemma 4 using the previously defined set $\mathscr{B}$ we conclude that batch B_q cannot start earlier than time t, so that the job in that batch is late. Thus, a feasible schedule in which batch B_z is scheduled before time r cannot exist. □

Lemma 6. *If the crisis subroutine declares failure, then no feasible schedule with all jobs meeting their deadlines exists.*

Proof. Subroutine Crisis(i) halts while considering job i only if the preceding batches $B_1,\ldots,B_{q-1}$, which have already been scheduled, are full and no job from those batches has a deadline larger than d_i. Since t is the earliest possible time when job i can start if the batches $B_1,\ldots,B_{q-1}$ start as early as possible and $d_i < t+p$, then clearly no feasible schedule exists. □

Lemma 7. *If the crisis subroutine does not declare failure at any stage of the algorithm 'Batch-Scheduling with Barriers', then the algorithm finds a feasible schedule with all jobs meeting their deadlines.*

The proof immediately follows from the fact that every job i is scheduled at the earliest possible time t for which condition $d_i \geq t+p$ is always satisfied.

Theorem 6. *If a feasible schedule for problem $P|p\text{-batch},b<n,p_j=p,r_j,c_j\leq d_j|-$ exists, then algorithm 'Batch-Scheduling with Barriers' produces a feasible schedule in at most $O(n^3\log n)$ time.*

Proof. The correctness of the algorithm follows from the preceding lemmas. Since there are at most n distinct release dates and at most n batches, the algorithm creates no more than n^2 barriers. A job can be scheduled in $O(\log n)$ time if a priority queue is used or in $O(\log\log n)$ time if a stratified binary tree is used. Since no more than n jobs are scheduled before a new barrier is created, the overall time complexity is $O\left(n^3\log n\right)$. □

Using the same arguments as in [19,93], one can demonstrate that the suggested algorithm constructs a schedule for problem $P|\text{p-batch},b<n,p_j=p,r_j,c_j\leq d_j|-$ such that all jobs meet their deadlines and in addition the values of $C_{\max}$ and $\sum c_j$ are minimum.

Finally we discuss the versions of the problem with precedence constraints or $L_{\max}$-criterion.

Since the problem with completion-start relations is NP-hard even for the single machine case, we consider only start-start precedence relations. It is easy to make sure that the arguments from Section 2.2.2 can be applied to the case of multiple machines and the

same preprocessing stage described as Stage 1 of the algorithm 'Batch-Scheduling with Start-Start Relations' followed by the algorithm 'Batch-Scheduling with Barriers' finds an optimal solution or concludes that none exists.

As far as $L_{\max}$-minimisation is concerned, the arguments from Section 2.2.3 hold so that the solution can be found using binary search over various values $\delta \in \mathscr{D}$, where $\mathscr{D}$ is defined by (2.5). As shown in Section 2.2.3, $|\mathscr{D}| = O(n^3)$ and this set can be generated and sorted in $O(n^3 \log n)$ time without constructing three layers $\mathscr{D}'$, $\mathscr{D}''$ and $\mathscr{D}'''$. The time complexity of solving the feasibility problem is also $O(n^3 \log n)$. Thus, using binary search over the set $\mathscr{D}$ combined with the $O(n^3 \log n)$-time feasibility algorithm results in an $O(n^3 \log^2 n)$-time algorithm for solving the $L_{\max}$-minimisation problem.

2.4 Discussion

In this chapter we presented several algorithms for solving p-batching problems with equal-length jobs outperforming the known algorithms published during the last 20 years. The developed algorithms generalized the concepts of forbidden regions and barriers formulated by Garey et al. [39] and Simons [94] for the classical problem where batching is not allowed. The results are summarised in two tables: Table 2.3 for the versions of the problem without precedence constraints (feasibility and $L_{\max}$-minimisation) and Table 2.4 for the feasibility version of the problem with precedence constraints. For comparison purposes, both tables include the results for the classical single and parallel machine problems where batching is not allowed.

Our results show how the elegant techniques developed for the classical single and parallel machine problems with equal processing times can be generalized for the p-batching problems resulting in faster algorithms. In addition, the results for the problems with start-start precedence constraints provide a powerful algorithmic tool to solve more complex problems in which a p-batching model appears as a subproblem.

Comparing the two types of the algorithms based on the concepts of forbidden regions and barriers, the algorithms which use forbidden regions appear to be faster than those using barriers, see results [93] and [94] in Table 2.3. A possible subject for future research could be exploring the idea of forbidden regions further and generalising it for the problem with parallel p-batching machines, although this might be a technically challenging task.

It is important to notice that, while in [39] a lot of work has been done on improving the implementation of the forbidden region algorithm in order to reduce its time complexity, the same work was not necessary in our case. In fact, since our objective is to minimise the maximum lateness, any time-complexity reduction of the decision version

of the problem would be neutralised by the time-complexity of sorting the values used the binary search.

Another direction of future research is related to the application of the developed algorithms to more complex problems which involve batching decisions. For example, supply chain management problems require coordination of transportation and production decisions. A transportation sub-problem associated with a segment of the supply network is usually characterized by equal delivery times and therefore it can be reduced to the batch-scheduling model studied in this chapter. Then batch-scheduling algorithms may be used as subroutines for solving transportation subproblems, for calculating lower bounds in branch-and-bound methods, or in local search procedures.

	Feasibility problem		$L_{\max}$-minimization	
$m = 1$, no batching	$O(n \log n)$	[39]*	$O(n^3 \log\log n)$	[94]**
			$O(n^2 \log n)$	[39]* combined with Section 2.2.3
$m = 1$, batching	$d_j = d$		$d_j = d$	
	$O(n \log n)$	[50]	$O(n \log n)$	[50]
	agreeable r_j and d_j		agreeable r_j and d_j	
	$O(n^2)$	[50]		
	$O(nb)$	[58]	$O(nb \log(np))$	[58]
	$O(n^8)$	[10]	$O(n^8 \log n)$	[10]
	$O(n^2)$	Section 2.2.1*	$O(n^2 \log n)$	Section 2.2.3*
$m > 1$, no batching	$O(n^3 \log n)$	[93]**	$O((d_{\max} \log n + n)nm)$, where $d_{\max} = \max_{j \in N}\{d_j\}$	[98]
	$O(n^2 m)$	[94]*		
$m > 1$, batching	$O(n^3 \log n)$	Section 2.3**	$O(n^3 \log^2 n)$	Section 2.3**

* Algorithms based on the concept of forbidden regions
** Algorithms based on the concept of barriers

Table 2.3: Summary of the results for problems without precedence constraints

	completion-start precedence relations *prec*	start-start precedence relations *prec'*
$m = 1$, no batching	$O(n \log n)$ [39]	$O(n \log n)$ [39]
$m = 1$, batching	strongly NP-hard [97]	$O(n^2)$ Section 2.2.2
$m > 1$, batching/ no batching	strongly NP-hard [97]	$O(n^3 \log n)$ Section 2.3

Table 2.4: Summary of the results for the feasibility problems with precedence constraints

Chapter 3

Coupled-operation scheduling

3.1 Overview

In this chapter we study a theoretical model for scheduling coupled-operations in a single-machine environment. We remind the reader that coupled-operation scheduling is a special class of scheduling problems with time-lags constraints where each job is made of two operations to be processed. The differences between the starting time of operations of the same jobs are defined by the time-lags values given in the problem instance. The problem consists on scheduling all operations on a single machine such that the maximum completion time (makespan) is minimised.

Coupled-operation problem arises in various applications: radar controllers send and receive signals after a delay since the pulse emission [32, 33, 88, 104]; some health care treatments must follow delivery patterns with strictly defined time-lags (see Chapter 4) ; in distributed computing the master processor organises data transmission and the time-lags in-between transmissions of input and output files correspond to the execution stage by a slave processor [62, 64].

Formally, in the model under consideration each job j of the set $N = \{1, 2, \ldots, n\}$ consists of a pair of operations a_j and b_j which should be processed without preemption by a single machine. Job j is characterized by a triple p_{a_j}, L_j, p_{b_j}, where p_{a_j} and p_{b_j} are processing times of a_j and b_j and L_j is the duration of a given time-lag. If the first

operation a_j starts at time s_{a_j}, then the second operation b_j should start exactly at time

$$s_{b_j} = s_{a_j} + p_{a_j} + L_j. \tag{3.1}$$

The machine can be used for processing other jobs in-between a_j and b_j, but the time-lag of duration L_j between completing a_j and starting b_j should be observed. The completion time of the second operation determines the completion time c_j of job j,

$$c_j = s_{b_j} + p_j.$$

The objective is to sequence all $2n$ operations of jobs N so that the machine processes at most one operation at a time and the makespan $C_{\max} = \max_{j \in N} \{c_j\}$ is minimum. Extending the notation from [42] and [74] we denote the problem by $1 \mid a_j, b_j, L_j \mid C_{\max}$.

The introduced model is characterized by fixed time-lags: in accordance with (3.1), the difference $s_{b_j} - (s_{a_j} + p_{a_j})$ between the starting time of b_j and completion time of a_j should be exactly equal to L_j. There are several related models of couple-operation scheduling in which time-lags are treated differently. In the model with flexible time-lags, denoted by $1 \mid a_j, b_j, \ell_j, u_j \mid C_{\max}$, the difference between the starting time of b_j and completion time of a_j should be within given boundaries,

$$\ell_j \leq s_{b_j} - (s_{a_j} + p_{a_j}) \leq u_j. \tag{3.2}$$

In the model with minimum time-lags, denoted by $1 \mid a_j, b_j, \ell_j, u_j = \infty \mid C_{\max}$, the upper bound is unlimited so that the delay between the first and the second operations can be indefinitely large.

All formulated versions of the coupled-operation problem are strongly NP-hard, see [74], [102] and [109]. Polynomial-time algorithms are known only for special cases and most of the results for the problem with fixed time-lags are formulated in [74] where a detailed classification of NP-hard and polynomially solvable cases is given. The long-standing open question on the complexity of problem $1 \mid a_j = b_j = p, L_j = L \mid C_{\max}$ has been recently resolved in [11], where the $O(\log n)$-time dynamic programming algorithm is proposed improving the preliminary results from [6]. The most recent research is mainly focused on various special cases of the problem with equal job parameters (e.g., equal processing times), but in a more general setting: coupled-operation jobs are replaced by chains of several operations, see [21, 68].

As far as approximation algorithms are concerned, the problem with fixed time-lags $1 \mid a_j, b_j, L_j \mid C_{\max}$ is approximable within a factor of $7/4$ in the case of unit-time operations

$a_j = b_j = 1$ [3], within a factor of $5/2$ in the case of equal-length operations $a_j = b_j$ and within a factor of $7/2$ in the general case (arbitrary a_j and b_j) [4]; it is also shown in [4] that no polynomial-time algorithm exists with an approximation ratio $2 - \varepsilon$ unless $P = NP$.

Another stream of research considers coupled-operation problems with precedence constraints, see, e.g., the summary and the main results in [13]. Notice that for the problem with flexible time-lags $1 \mid a_j, b_j, \ell_j, u_j, \pi \mid C_{\max}$, finding optimum starting times minimizing the makespan is a non-trivial task even if precedence constraints define a complete sequence π of operations. The algorithm involves longest path calculation in a disjunctive graph having positive- and negative-weight arcs (see [49, 81]). We are not aware of any algorithms for solving the coupled-operation problem with a given sequence π_a of first operations or with a given sequence π_b of second operations.

To the best of our knowledge, there is only one publication [81] which discusses heuristic algorithms for the coupled-operation problem. It studies the version of the problem with flexible time-lags $1 \mid a_j, b_j, \ell_j, u_j \mid C_{\max}$ comparing different constructive heuristics and local search algorithms. Interestingly, the proposed local search algorithms appear to be less efficient than the most successful constructive heuristics and that inefficiency of local search is explained by the high cost of generating and evaluating infeasible solutions. Indeed, the algorithms developed in [81] for flexible time-lags are applicable to the version with fixed time-lags $1 \mid a_j, b_j, L_j \mid C_{\max}$. However, due to the importance of the latter problem for real-world applications, it is particularly desirable to design efficient problem-specific algorithms for it. As we show in this thesis, this can be achieved by exploiting special properties of the problem.

Our main objective is to design a successful local search method. We start this study with the analysis of possible representations of feasible solutions and the neighbourhood structure. The local search approach from [81] developed for the problem $1 \mid a_j, b_j, \ell_j, u_j \mid C_{\max}$ with flexible time-lags represents feasible solutions as permutations π of all $2n$ operations and generates neighbours by removing one operation from π and inserting it elsewhere. The quality of a new solution is evaluated via the $O(n^2)$ longest path algorithm which either fixes the starting times of all operations observing given bounds ℓ_j, u_j for time-lags or identifies that no feasible solution exists. As noted in [81], the resulting local search approach performs poorly in comparison with constructive heuristics since infeasible solutions prevail around local optima. It is likely that such an approach would perform even worse for problem $1 \mid a_j, b_j, L_j \mid C_{\max}$ with fixed time-lags as infeasible neighbours would probably occur more often as time-lags cannot be adjusted in that problem. Due to this reason, in our study we pay special attention to

alternative representations of feasible solutions and alternative neighbourhood structures.

For solution representation, one natural approach is based on considering permutation π_a of a-operations or permutation π_b of b-operations. Notice that due to the problem symmetry for the makespan objective, a problem with a fixed permutation π_a can be reformulated as the problem with a fixed permutation π_b. Using a single permutation π_a (or permutation π_b) may seem attractive as this representation is compact and leads to simple strategies for neighbour generation. Such a representation induces an important subproblem in which the permutation π_a (or π_b) is fixed and the objective is to produce a complete schedule, i.e., to specify complete permutation π of all operations $2n$ and their starting times so that the makespan $C_{\max}$ is minimum. We perform complexity analysis of that subproblem in Section 3.2 and demonstrate that it is NP-hard in the strong sense even in the case of unit processing time. This negative result suggests that using representation π_a (or, equivalently, π_b) is less preferable in comparison with the full permutation π.

As far as neighbour generation strategy is concerned, we take into account the conclusions from [81] on inefficiency of re-insertion of a single operation in a given permutation π of $2n$ operations as it often leads to infeasible permutations. The alternative neighbour generating strategy we suggest optimally re-inserts the whole job consisting of two operations. Although the difference between our strategy and the one from [81] may look insignificant, it in facts results in an efficient search procedure: our strategy always generates feasible solutions during the search and the neighbours are obtained as solutions to a specially defined optimization problem. Enumerating neighbours of good quality is perhaps one of the reasons of good performance of our method.

The formulated ideas are elaborated in the tabu search algorithm presented in Section 3.4. We suggest two enhancements that improve the search:

- maintaining the pool of solutions ranked in accordance with the estimates of possible improvements that can be achieved if the neighbour is generated;
- creating the tabu list which keeps the main characteristics of the eliminated solutions in the format of critical paths.

The performance of the tabu search algorithm is evaluated empirically comparing it with the winning method from [81] - the random constructive heuristic. Its adaptation for the model with fixed time-lags is described in Section 3.5 followed by the summary of computational experiments in Section 3.6. Conclusions are the drawn in Section 3.7.

3.2 NP-hardness of the Problem with a Given Sequence of First Operations

It is known that if the permutation π of $2n$ operations is given, then their optimum starting times can be found in $O(n^2)$ time as a solution to a specially defined longest path problem [81]. In this section we consider the related problem in which the permutation π_a of a-operations is given while permutation of b-operations is not fixed. Without loss of generality we assume that the jobs are numbered in accordance with the permutation of a-operations so that $\pi_a = (1,2,\ldots,n)$. The objective is to find the starting times of all operations and complete permutation π of all operations that minimises the makespan so that the makespan $C_{\max}$ is minimum.

Our main result is the proof that the coupled-operation problem with a given permutation π_a of a-operations is NP-hard in the strong sense even if all operations have unit processing times. The decision version of this problem consists in verifying whether there exists a feasible solution with the makespan not-exceeding a given threshold value T. The latter problem is denoted by COED(π_a, T).

We reduce the following coupled-operation problem with minimum delay and a given makespan threshold t, denoted by COMD(t), to problem COED(π_a, T).
COMD(t): given a set of jobs $Q = \{1,2,\ldots,q\}$, each job $j \in Q$ consisting of two unit-time operations separated by a time-lag of duration no less than ℓ_j, does there exist a feasible schedule with the makespan no larger than t, so that no two operations are processed simultaneously and the second operation of each job j starts after at least ℓ_j time units elapses the first operation of that job is completed. It is known that COMD(t) is NP-complete in the strong sense, see [109].

The reduction described in this section is based on the following observations:

1. Consider two unit-time coupled-operation jobs i and j with exact time lags such that $l_i \leq l_j$ and the a-operation of the job i starts to be processed before the a-operation of the job j. Then, the difference between the starting times of the b-operation of jobs j and i is always greater than $l_j - l_i$. In addition, it is possible to arbitrarily set the job starting times at which a-operation of job j and b-operation of job i swap. This is achieved by increasing both time-lags values of an arbitrary constant.

2. Given a set of jobs and a fixed sequence of a-operations, there exists a assignment of values to the jobs time-lags such that all permutations of b-operation can be constructed respecting the initial sequence of a-operation.

The idea of the reduction is to model the operations of each job for the problem COMD

by the b-operations of two jobs of COED (later called *A-proxy* and *B-proxy* jobs). The b-operations of the proxy jobs reproduce a solution of the COMD problem in a fixed time interval in COED schedule. In order to guarantee that the time-lags between b-operations of proxy jobs is no less than a minimum values, time-lags of pairs of proxy jobs are tuned according to observation 1. Then, since the b-operations of the proxy jobs should have the possibility to be scheduled according to any permutation, the time-lag differences between different proxy job pair are set according to observation 2. Finally, in order to force the b-operations of the proxy jobs to be scheduled within a given time-interval, special *filler* jobs are scheduled in the first part of the schedule such that:

- the last a-operation of filler jobs is scheduled just one unit before all their b-operations;
- a-operations of the proxy jobs are scheduled in between b-operations of the filler jobs.
- the part of the schedule where b-operations of filler jobs and a-operations of proxy jobs are processed is tight, i.e. no idle time appears.

An example of COED schedule results of a reduction of a COMD problem instance can be seen in Figure 3.1.

More formally, given an instance $\mathscr{I}(t)$ of problem COMD we construct an instance $\mathscr{I}'(\pi_a, T)$ of problem COED(π_a, T) as follows. The set of jobs N in instance $\mathscr{I}'(\pi_a, T)$ consists of $n = 3tq + 1$ coupled-operation jobs which fixed time-lags L_j are defined in accordance with the job type.

- There are q sets $F_1, \ldots, F_q$ of the so-called *filler jobs*, each set $F_h = \{(3t-2)(h-1)+1, \ldots, (3t-2)h\}$, $1 \leq h \leq q$, consisting of $3t-2$ jobs with time-lags
$$L_j = 3tq, \quad j \in F_h.$$
- There is one set R consisting of a single *sentinel job*, $R = \{(3t-2)q+1\}$, with the time-lag
$$L_j = t(3q+1), \quad j \in R.$$
- There are q sets $U_1, \ldots, U_q$ of *A-proxy* jobs, each set $U_h = \{(3t-2)q+2h\}$, $1 \leq h \leq q$, consisting of a single job with the time-lag
$$L_j = 3t(q-h)+2t-1, \quad j \in U_h.$$

- There are q sets $V_1,\ldots,V_q$ of *B-proxy* jobs, each set $V_h=\{(3t-2)q+2h+1\}$, $1\le h\le q$, consisting of a single job with the time-lag

$$L_j = L_{j-1}+\ell_h = 3t(q-h)+2t-1+\ell_h, \quad j\in V_h.$$

Observe that

$$L_j \ge 2t-1 \text{ for } j\in U_h\cup V_h. \tag{3.3}$$

The makespan threshold value is $T=6tq+t+2$ and the sequence of a-operations is $\pi_a=(1,2,\ldots,3tq+1)$:

$$\pi_a = (\underbrace{1,\ldots,3t-2}_{F_1},\underbrace{3t-1,\ldots,2(3t-2)}_{F_2},\ldots,\underbrace{(3t-2)(q-1)+1,\ldots,(3t-2)q}_{F_q},$$
$$\underbrace{(3t-2)q+1}_{R},$$
$$\underbrace{(3t-2)q+2}_{U_1},\underbrace{(3t-2)q+3}_{V_1},\underbrace{(3t-2)q+4}_{U_2},\underbrace{(3t-2)q+5}_{V_2},\ldots,\underbrace{3tq}_{U_q},\underbrace{3tq+1}_{V_q}). \tag{3.4}$$

In what follows, we use the following notation for single-element sets:

$$\begin{aligned} R &= \{r\},\\ U_h &= \{u(h)\},\\ V_h &= \{v(h)\}. \end{aligned}$$

Theorem 7. *If there exists a feasible schedule for instance $\mathcal{I}(t)$ of problem* COMD(t), *then there exists a feasible schedule for instance $\mathcal{I}'(\pi_a,T)$ of problem* COED(π_a,T).

Proof: Let a feasible solution S for instance $\mathcal{I}(t)$ be given by starting times s_{a_h} and s_{b_h}, $h\in Q$, of its a- and b-operations. We construct a feasible schedule S' for instance $\mathcal{I}'(\pi_a,T)$ with permutation π_a given by (3.4). First we describe the structure of that schedule and then specify the starting times s'_{a_j} and s'_{b_j}, $j\in N$, for all operations.

There are five types of time intervals:

- q time intervals $\lambda_1,\lambda_2,\ldots,\lambda_q$, each of length $3t$, such that in interval

$$\lambda_h=[3t(h-1),\ 3th],\ \ 1\le h\le q,$$

all first operations of the jobs from F_h are processed; since there are $3t-2$ jobs in

F_h and the length of interval λ_h is $3t$, there are two idle time intervals of unit length in each interval λ_h;

- one unit-length time interval

$$\xi = [3tq,\ 3tq+1]$$

for processing the first operation of the sentinel job from R;

- q time intervals $\mu_1, \mu_2, \ldots, \mu_q$, each of length $3t$, such that in interval

$$\mu_h = [3t(q+h-1)+1,\ 3t(q+h)+1],\ \ 1 \le h \le q,$$

all second operations of the jobs from F_h are processed together with the first operations of the jobs from U_h and V_h; since there are $3t-2$ jobs in F_h, one job in U_h and one job in V_h, there are no idle time slots in μ_h;

- one interval τ of length t defined as

$$\tau = [6tq+1,\ 6tq+1+t]$$

for processing the second operations of the jobs from $\bigcup_{h=1}^{q} U_h$ and $\bigcup_{h=1}^{q} V_h$;

- one unit-length time interval

$$\eta = [6tq+t+1,\ 6tq+t+2]$$

for processing the second operation of the sentinel job from R.

To give a full description of the schedule, we specify the starting times $s'_{b_1}, s'_{b_2}, \ldots, s'_{b_n}$, of the b-operations of jobs N; the starting times of the corresponding a-operations are then derived as $s'_{a_j} = s'_{b_j} - L_j - 1$.

The job completing at time T in the schedule for instance $\mathscr{I}'(\pi_a, T)$ is the sentinel job $r \in R$ and its two operations are processed in time intervals ξ and η:

$$s'_{b_r} = T - 1 = 6tq+t+1,\ \ s'_{a_r} = (6tq+t+1) - t(3q+1) - 1 = 3tq.$$

Scanning the schedule for instance $\mathscr{I}'(\pi_a, T)$ from its right end backwards, we define the starting times of the jobs from $V_q, U_q, V_{q-1}, U_{q-1}, \ldots, V_1, U_1$. For each $1 \le h \le q$, the time slots for the second operations of $v(h) \in V_h$ and $u(h) \in U_h$ are allocated on the basis

of the starting times s_{a_h} and s_{b_h} of the two operations of job h in the schedule for instance $\mathscr{I}(t)$:

$$\begin{aligned} s'_{b_{v(h)}} = 6tq+1+s_{b_h}, \quad s'_{a_{v(h)}} &= \big(6tq+1+s_{b_h}\big)-(3t(q-h)+t+1+\ell_h)-1 \\ &= 3t(q+h)+s_{b_h}-t-1-\ell_h, \\ s'_{b_{u(h)}} = 6tq+1+s_{a_h}, \quad s'_{a_{u(h)}} &= (6tq+1+s_{a_h})-(3t(q-h)+t+1)-1 \\ &= 3t(q+h)+s_{a_h}-t-1. \end{aligned} \tag{3.5}$$

Observe that the difference in the starting times of second operations of jobs $v(h)$ and $u(h)$ in the schedule for instance for instance $\mathscr{I}'(\pi_a,t)$ is the same as the distance between two operations of job h in the schedule for instance $\mathscr{I}(t)$:

$$s'_{b_{v(h)}} - s'_{b_{u(h)}} = s_{b_h} - s_{a_h}.$$

In addition, in the schedule for instance for instance $\mathscr{I}'(\pi_a,t)$

(i) the first operation of job $v(h)$ starts after the first operation of job $u(h)$:

$$s'_{a_v} \geq s'_{a_u} + 1;$$

(ii) all second operations of the jobs $\bigcup_{h=1}^{q} U_h$ and $\bigcup_{h=1}^{q} V_h$ are fully processed in time interval τ;

(iii) for each fixed h, $1 \leq h \leq q$, the first operations of $u(h)$ and $v(h)$ are both processed in time interval μ_h.

Here observations (ii) and (iii) follow from the fact that in instance $\mathscr{I}(t)$, the starting times s_{a_h} and s_{b_h} are non-negative for any job $h \in Q$ and they are bounded by $t-2$.

We now define the time intervals for the filling jobs. Consider the set F_h for some h, $1 \leq h \leq q$. The $3t-2$ jobs in F_h have their second operations processed in time interval μ_h. The length of the latter interval is $3t$ and two unit-time slots within μ_h are already fixed for $u(h)$ and $v(h)$, see (3.5). Thus there is a unique way to allocate the jobs from F_h in time interval μ_h keeping them in the order of their numbering and avoiding clashes with $u(h)$ and $v(h)$.

Having allocated the second operations of the jobs from F_h in interval μ_h, their first operations are automatically allocated in interval λ_h. Since the time-lags for the jobs in F_h are equal, the order of the first operations is the same as the order of the second operations and it satisfies the given permutation π_a. Taking into account observation (i), the order given by π_a is satisfied for all jobs. ∎

An example of schedule generated by the reduction shown in Theorem 7 is shown in Figure 3.1.

In the remaining part we prove that if there exists a feasible schedule for instance $\mathcal{I}'(\pi_a, T)$ of problem COED(π_a, T), then there exists a feasible schedule for the related instance $\mathcal{I}(t)$ of problem COMD(t). We assume that t satisfies the following two conditions:

$$t \geq \ell_j + 2, \quad j \in Q, \tag{3.6}$$

$$t \geq 2q; \tag{3.7}$$

otherwise COMD(t) does not have a feasible solution since the makespan of the coupled-operation schedule cannot be smaller either of the values, the length of a single job or the combined length of q jobs consisting of two unit-time operations.

We start with a lemma which characterises the structure of a feasible solution for instance $\mathcal{I}'(\pi_a, T)$.

Lemma 8. *In any feasible solution S' for instance $\mathcal{I}'(\pi_a, T)$ of problem* COED(π_a, T) *the following properties hold:*

1) *the a-operations of all jobs from N are processed in one of the time intervals $\lambda_1, \ldots, \lambda_q, \xi, \mu_1, \ldots, \mu_q$, i.e.,*
   $s'_{a_j} \leq 6tq, \qquad j \in N;$

2) *the a-operation of the sentinel job $r \in R$ is processed in time interval ξ or earlier, i.e.,*
   $s'_{a_r} \leq 3tq;$

3) *the a-operations of all jobs from $\bigcup_{h=1}^{q} F_h$ are processed in time intervals $\lambda_1, \ldots, \lambda_q$, i.e.,*
   $s'_{a_j} \leq 3tq - 1, \qquad j \in F_h;$

4) *the b-operations of all jobs from F_h, $1 \leq h \leq q$, are processed in time intervals $\mu_1, \ldots, \mu_q$, i.e.,*
   $3tq + 1 \leq s'_{b_j} \leq 6tq, \qquad j \in F_h;$

5) *each unit time slot of intervals $\mu_1, \ldots, \mu_q$ has an operation allocated to it; the set of operations allocated to intervals $\mu_1, \ldots, \mu_q$ consists of all b-operations of the jobs from $\bigcup_{h=1}^{q} F_h$ and exactly one operation for each job from $\bigcup_{h=1}^{q} (U_h \cup V_h)$.*

6) *job $j = (3t-2)q$, which is the last job of the set F_q, starts at time $3tq - 1$;*

7) *the a-operations of all jobs from* $\bigcup_{h=1}^{q}(U_h \cup V_h)$ *are processed in time intervals* $\mu_1, \ldots, \mu_q$, *i.e.*
$s'_{a_j} \geq 3tq+1, \qquad j \in \bigcup_{h=1}^{q}(U_h \cup V_h)$;

8) *the b-operations of all jobs from* $\bigcup_{h=1}^{q}(U_h \cup V_h)$ *are processed in time interval* τ, *i.e.*
$6tq+1 \leq s'_{b_j} \leq 6tq+t, \qquad j \in \bigcup_{h=1}^{q}(U_h \cup V_h)$.

Proof:
1) For any job j from $\bigcup_{h=1}^{q}(U_h \cup V_h)$, its time-lag satisfies (3.3), while all other jobs have even larger time-lags. For any job $j \in N$, its completion time should not be larger than $T = 6tq+t+2$, which implies

$$s'_j \leq T - L_j - 2 = (6tq+t+2) - (2t-1) - 2 = 6tq - t + 1 \leq 6tq.$$

2) To be completed within the threshold makespan value T, the starting time of the a-operation of job $r \in R$ should satisfy

$$s'_{a_r} \leq T - L_r - 2 = (6tq+t+2) - t(3q+1) - 2 = 3tq.$$

3) In accordance with sequence π_a, any filler job from $\bigcup_{h=1}^{q} F_h$ must start before the sentinel job from R, i.e., at time $3tq-1$ or earlier.
4) By the previous property, all filler jobs from $\bigcup_{h=1}^{q} F_h$ have their starting times within $[0, 3tq-1]$. Since all of them have the same time-lag of $3tq$, the b-operations start within time interval $[3tq+1, 6tq]$.
5) Due to the property 4), there are $(3t-2)q$ b-operations of the jobs from $\bigcup_{h=1}^{q} F_h$ which have their starting times in $[3tq+1, 6tq]$, leaving room for at most $2q$ additional unit-time operations in $\mu_1, \ldots, \mu_q$. We show that exactly $2q$ operations of the jobs from $\bigcup_{h=1}^{q} U_h$ and $\bigcup_{h=1}^{q} V_h$, are scheduled in $\mu_1, \ldots, \mu_q$, one operation for each job.

Consider f job $j \in \bigcup_{h=1}^{q}(U_h \cup V_h)$. If

$$s'_{a_j} \geq 3tq+1, \tag{3.8}$$

then due to property 1), the a-operation of job h is processed within in $\mu_1, \ldots, \mu_q$.

Otherwise, i.e., if

$$s'_{a_j} \leq 3tq, \tag{3.9}$$

we show that the corresponding b-operation is processed within in $\mu_1, \ldots, \mu_q$.

To prove that the left boundary $3tq$ of μ_1 is observed, we derive lower bounds on s'_{a_j} and s'_{b_j}. Since the set of jobs which precede j in permutation π_a includes $(3t-2)q$ operations of filler jobs from $\bigcup_{h=1}^{q} F_h$ and one sentinel job from R,

$$s'_{a_j} \geq (3t-2)q+1.$$

Taking into account condition (3.3), we obtain:

$$s'_{b_j} = s'_{a_j} + L_j + 1 \geq ((3t-2)q+1) + (2t-1) + 1 = 3tq + 2(t-q) + 1 \geq 3tq + 2q + 1,$$

where the last inequality is due to assumption (3.7).

To prove that the right boundary $6tq$ of μ_q is observed, we derive the upper bound on s'_{b_j} using condition (3.9) and the fact that the time-lag L_j is bounded by the maximum time-lag in the set $\bigcup_{h=1}^{q}(U_h \cup V_h)$, which is $L_{v(1)} = 3t(q-1) + 2t - 1 + \ell_1$. We conclude:

$$s'_{b_j} = s'_{a_j} + L_j + 1 \leq 3tq + (3t(q-1) + 2t - 1 + \ell_1) + 1 = 6tq - t + \ell_1 \leq 6tq - 2,$$

where the last inequality is due to assumption (3.6).

Thus in any case, (3.8) or (3.9), at least one operation of a job $j \in \bigcup_{h=1}^{q}(U_h \cup V_h)$ is processed within intervals $\mu_1, \ldots, \mu_q$. Taking into account that there are $2q$ unit time intervals left after allocation of the b-operations of $\bigcup_{h=1}^{q} F_h$, we conclude that exactly one operation of every job j is processed within intervals $\mu_1, \ldots, \mu_q$.

6) Consider job $j = (3t-2)q$, which is the last one in the set F_q. By condition 3),

$$s'_{a_j} \leq 3tq - 1.$$

To prove that

$$s'_{a_j} \geq 3tq - 1$$

we show that

$$s'_{b_j} \geq 6tq.$$

Suppose that $s'_{b_j} < 6tq$. By property 5), there is a job $j \in \bigcup_{h=1}^{q}(U_h \cup V_h)$ allocated to time interval $[6tq, 6tq+1]$. If it is the first operation of j, i.e., $s'_{a_j} = 6tq$, then the deadline T is violated:

$$s'_{b_j} = s'_{a_j} + L_j + 2 \geq 6tq + (2t-1) + 2 > T,$$

where the inequality is due to (3.3). If it is the second operation of j, i.e.,

$$s'_{b_j} = 6tq,$$

then job j would have two operation processed within $\mu_1, \ldots, \mu_q$ since its time-lag satisfies

$$L_j \leq 3t(q-1) + 2t - 1 + \ell_h \leq 3tq - 3,$$

which implies

$$s'_{a_j} = s'_{b_j} - L_j - 1 \geq 6tq - (3tq-3) - 1 = 3tq + 2,$$

a contradiction to property 5).

7) In accordance with permutation π_a, the starting time of the last job of the set F_q with index $f = (3t-2)q$ is smaller than that of the sentinel job $r \in R$ which in its turn is smaller than the starting time of any job $j \in \bigcup_{h=1}^{q} (U_h \cup V_h)$:

$$s'_{a_f} < s'_{a_r} < s'_{a_j},$$

so that

$$s'_{a_j} \geq s'_{a_f} + 2.$$

The statement now follows from property 6).

8) The total length of time intervals $\mu_1, \ldots, \mu_q$ is $3tq$ and in accordance with properties 5) and 7) there are exactly $3tq$ operations allocated there: $(3t-2)q$ operations from $\bigcup_{h=1}^{q} F_h$ and $3q$ a-operations from $\bigcup_{h=1}^{q} (U_h \cup V_h)$. Therefore all b-operations of the jobs from $\bigcup_{h=1}^{q} (U_h \cup V_h)$ are processed after time intervals $\mu_1, \ldots, \mu_q$.

It remains to show that no b-operation from $\bigcup_{h=1}^{q} (U_h \cup V_h)$ can be processed in time interval ξ. If it is a case, then for the sentinel job $r \in R$,

$$\begin{aligned} s'_{b_r} &\leq 6tq + t, \\ s'_{a_r} &= s'_{b_r} - L_r - 1 = s'_{b_r} - t(3q+1) - 1 \leq 3tq - 1. \end{aligned}$$

The latter condition cannot happen since due to property 6) the a-operation of the last job $f = (3t-2)q$ from F_q is assigned to time slot $[3tq-1, 3tq]$ and it should precede the a-operation of the sentinel job. ∎

Based on the properties formulated in Lemma 8, we now complete the proof of the NP-hardness result.

Theorem 8. *If there exists a feasible schedule for instance $\mathscr{I}'(\pi_a, T)$ of problem*

COED(π_a, T), *then there exists a feasible schedule for instance* $\mathscr{I}(t)$ *of problem* COMD(t).

Proof: Consider a feasible schedule for instance $\mathscr{I}'(\pi_a, T)$. By construction of instances $\mathscr{I}(t)$ and $\mathscr{I}'(\pi_a, T)$, for any job h of instance $\mathscr{I}(t)$, its two operation a_h and b_h correspond to the following two operations of instance $\mathscr{I}'(\pi_a, T)$: the b-operation of job $u(h) \in U_h$ and the b-operation of job $v(h) \in V_h$.

Due to property 8) of Lemma 8, all b-operations of the jobs from $\bigcup_{h=1}^{q}(U_h \cup V_h)$ are processed in time interval τ of length t. We define a solution to instance $\mathscr{I}(t)$ via the partial solution to instance $\mathscr{I}'(\pi_a, T)$ in the time interval τ. For any job h, $1 \leq h \leq q$, of instance $\mathscr{I}(t)$, the starting times of its two operations are defined as

$$\begin{aligned} s_{a_h} &:= s'_{b_{u(h)}} - (6tq+1), \quad u(h) \in U_h, \\ s_{b_h} &:= s'_{b_{v(h)}} - (6tq+1), \quad v(h) \in V_h. \end{aligned}$$

It remains to show that in instance $\mathscr{I}(t)$

$$s_{b_h} \geq s_{a_h} + 1 + \ell_j,$$

or equivalently

$$s'_{b_{v(h)}} \geq s'_{b_{u(h)}} + 1 + \ell_j.$$

Indeed, due to permutation π_a which is observed in the solution to instance $\mathscr{I}'(\pi_a, T)$,

$$s'_{a_{v(h)}} \geq s'_{a_{u(h)}} + 1.$$

Taking into account that

$$\begin{aligned} L_{a_{u(h)}} &= 3t(q-h) + 2t - 1, \\ L_{a_{v(h)}} &= 3t(q-h) + 2t - 1 + \ell_h, \end{aligned}$$

we conclude:

$$\begin{aligned} s'_{b_{v(h)}} - s'_{b_{u(h)}} &= \left[s'_{a_{v(h)}} + L_{a_{v(h)}} + 1\right] - \left[s'_{a_{u(h)}} + L_{a_{u(h)}} + 1\right] \\ &= \left[s'_{a_{u(h)}} + 3t(q-h) + 2t - 1 + \ell_h\right] - \left[s'_{a_{u(h)}} + (3t(q-h) + 2t - 1)\right] \\ &\geq \ell_h. \end{aligned}$$

■

The results obtained in this section hold also for the symmetric problem in which a permutation π_b of b-operations is fixed, i.e. $1|a_j = b_j = 1, L_j, \pi_b|C_{\max}$. The equivalence of the two problems is proved by the fact that a solution for the problem $1|a_j = b_j = 1, L_j, \pi_a|C_{\max}$ can be easily transformed into a solution of the problem $1|a_j = b_j = 1, L_j, \pi_b|C_{\max}$ where $\pi_a = \pi_b$. In particular, consider the operation starting times $s_{a_1}, s_{a_2}, \ldots, s_{a_n}, s_{b_1}, s_{b_2}, \ldots, s_{b_n}$ of a solution of the problem $1|a_j = b_j = 1, L_j, \pi_a|C_{\max}$, the operation starting times $s'_{a_1}, s'_{a_2}, \ldots, s'_{a_n}, s'_{b_1}, s'_{b_2}, \ldots, s'_{b_n}$ for the problem $1|a_j = b_j = 1, L_j, \pi_b|C_{\max}$ can be calculated by the formula:

$$s'_i = C_{\max} - s_i \qquad \forall i \in \{a_j, b_j \mid j \in N\} \tag{3.10}$$

where $C_{\max}$ is the optimal makespan value obtained in the first problem. It is trivial to see that an optimal solution for the first problem is transformed in an optimal solution of the second and vice versa.

The problem $1|a_j = b_j = 1, L_j, \pi_a|C_{\max}$ considered in this section is a special case of the problem with equal processing times $1|a_j = b_j = p, L_j, \pi_a|C_{\max}$ and of the more general problem $1|a_j, b_j, L_j, \pi_a|C_{\max}$. This fact implies that both problems are also NP-hard in the strong sense. The hierarchy of the complexity results in this section is shown in Figure 3.2.

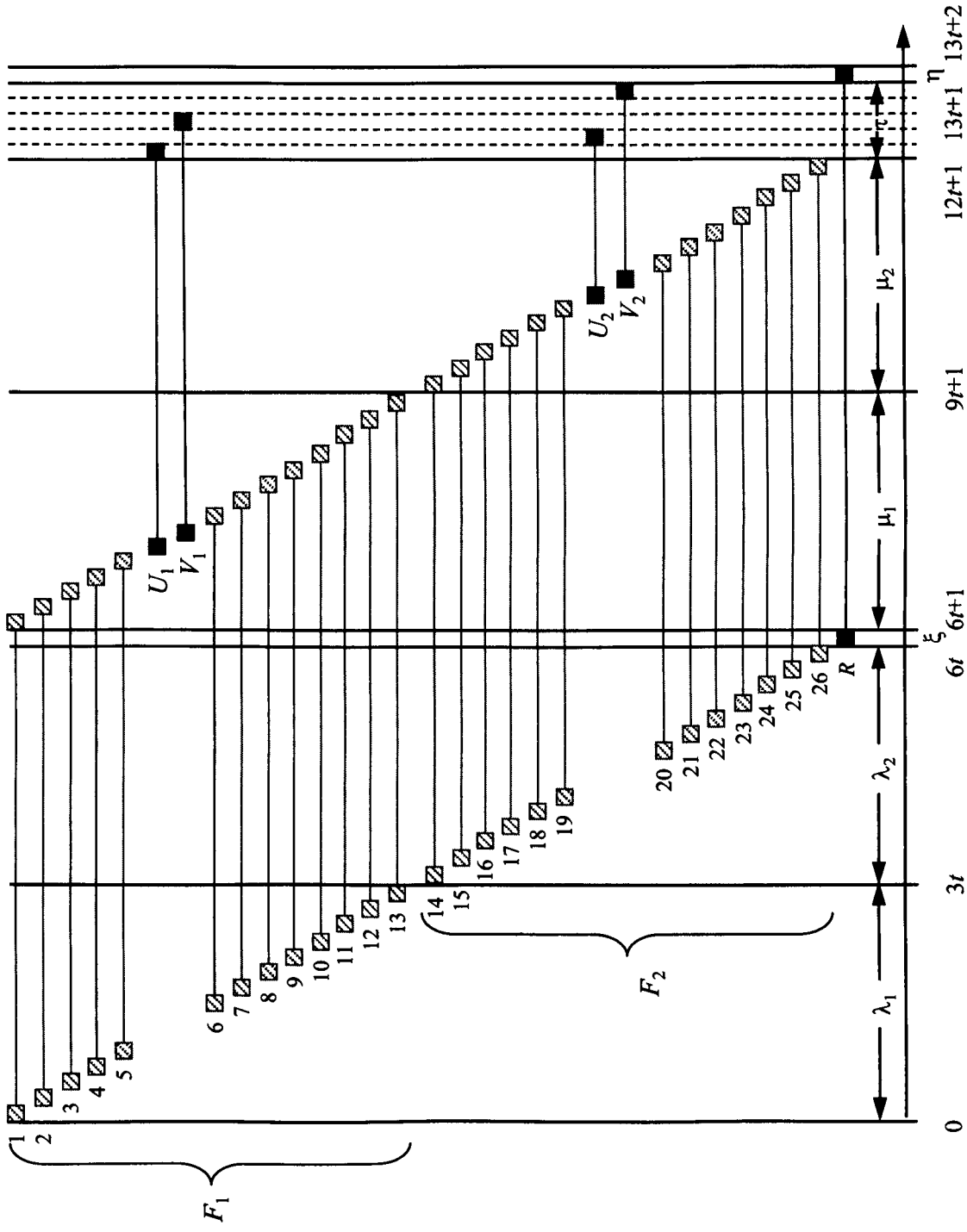

Figure 3.1: Example of feasible schedule for instance COED(π_a,67) generated from a feasible schedule for the problem COMD(5) with two coupled-operation jobs with minimum time-lags $\ell_1 = 1$ and $\ell_2 = 2$

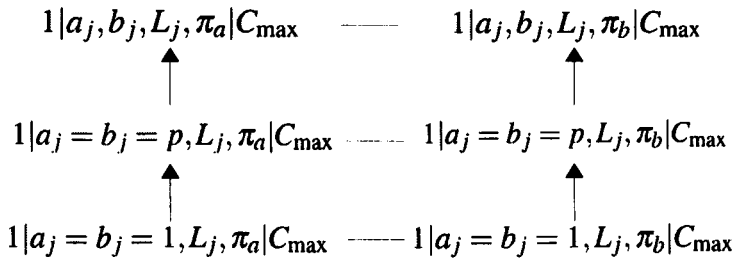

Figure 3.2: Graph of strongly NP-hard coupled-operation scheduling problems

3.3 Integer Linear Program Formulation

The coupled-operation scheduling problem with fixed time-lags can be modelled by many integer linear program formulations. These are useful to express the constraints of the problem in an unambiguous way and to provide an initial ready-to-use formulation to be run on general solvers. In this section we describe two possible integer linear program formulations.

In what follow we denote by O_j the set of operations to be scheduled for each job $j \in N$, and by A and B the sets of all a and b operations respectively. In addition we assume that operations are numbered such that $u < v$ for all $u \in A$ and $v \in B$.

The first formulation ILP_1 uses two sets of decision variables: *starting time* variables and *precedence* variables. There is a starting time variable $x_u \in \mathbb{N}$ for each operation $u \in O_j$ of a job $j \in N$. There is a precedence variable $y_{u,v}$ between all operations $u \in O_j$ and $v \in O_i$ belonging to different jobs $j, i \in N$ and such that $u < v$. Their values are assigned as follows:

$$y_{u,v} = \begin{cases} 1, & \text{if operation } u \text{ precedes operation } v \text{ in the final schedule ,} \\ 0, & \text{otherwise.} \end{cases}$$

In addition, an auxiliary variable $c \in \mathbb{N}$ is used to represent the makespan value of the schedule by the following set of constraints:

$$x_u + p_u \leq c \qquad u \in B \cap O_j, \quad j \in N$$

A second set of constraints models the fact that the processing of the operations cannot overlap in time :

$$x_v + p_v - y_{u,v} M \leq x_u \qquad u \in O_j, v \in O_i, \quad i, j \in N, \quad i \neq j.$$

The constant M is an upper bound of the minimum makespan of the schedule such that the completion time of any job j does exceed M. The time-lag between operations of the same job is modelled as follows:

$$x_u + p_u + L_j = x_v \qquad u \in A \cap O_j, \quad v \in B \cap O_j, \quad j \in N.$$

Thus the complete formulation is the following:

$$
\begin{aligned}
ILP_1: \quad & \min \quad c \\
& \text{s.t.} \quad x_u + p_u \leq c && u \in B \cap O_j, \quad j \in N, \\
& \qquad x_v + p_v - y_{u,v} M \leq x_u, && u \in O_j, v \in O_i, \quad i,j \in N, \quad i \neq j, \\
& \qquad x_u + p_u + L_j = x_v, && u \in A \cap O_j, \quad v \in B \cap O_j, \quad j \in N, \\
& \qquad y_{u,v} \in \{0,1\}, && u,v \in O_j, \quad u < v. \\
& \qquad x_u \in \mathbb{N}, && u \in O_j.
\end{aligned}
$$

This formulation has been tested on different solvers which failed to find feasible solutions even for small size instances (i.e. 10 jobs) in a reasonable amount of time. The cause of such inefficiency is the use of the so called BigM technique which models disjunctive constraints in the integer linear program and yields to weak bounds in the linear relaxation of the problem.

Better results have been obtained using a second integer linear program formulation ILP_2. Differently from the previous one, this formulation uses a discrete time horizon which is which is divided in contiguous unit-time slots. Consequently all instance parameters such as processing times and time-lags are expressed by positive integer values.

Let us denote by S_j the set of all possible times of the first operation of a job $j \in N$. Clearly, since the number of elements in S_j is finite and limited by an upper bound M on the maximum makespan. In formulation ILP_2, a variable $x_{j,s}$ is defined for each job $j \in N$ and each possible starting time $s \in S_j$ such that:

$$
x_{j,s} = \begin{cases} 1, & \text{if job } j \text{ starts to be processed at time } s \text{ in a schedule}, \\ 0, & \text{otherwise.} \end{cases}
$$

Similar to ILP_1, a variable c is used to model the makespan of the schedule.

Consider the set $K_{s,t}$ of jobs such that, if starting at time s, they have an operation processed at time t. Then the first set of constraints guarantees that no more than one operation is processed at each time slot t:

$$
\sum_{0 \leq s < M} \sum_{j \in K_{s,t}} x_{j,s} \leq 1, \qquad 0 \leq t < M.
$$

A second set of constraints guarantees that only one time slot is assigned as starting time

of each job. This is done by the following constraint:

$$\sum_{s \in S_j} x_{j,s} = 1, \qquad j \in N.$$

The last set of constraints forces the variable c to be no smaller than the completion time of all operations in the schedule:

$$sx_{j,s} + p_{a_j} + L_j + p_{b_j} \leq c, \qquad j \in N, \quad s \in S_j.$$

Summarising the complete formulation is as follows:

$$\begin{aligned} ILP_2: \quad & \min \quad c \\ & \text{s.t.} \quad \sum_{0 \leq s < M} \sum_{j \in K_{s,t}} x_{j,s} \leq 1, && 0 \leq t < M, \\ & \qquad \sum_{s \in S_j} x_{j,s} = 1, && j \in N, \\ & \qquad sx_{j,s} + p_{a_j} + L_j + p_{b_j} \leq c, && s \in S_j, \quad j \in N, \\ & \qquad x_{j,s} \in \{0,1\}, && s \in S_j, \quad j \in N. \end{aligned}$$

In ILP_2, although the LP relaxation provides better bounds than the relaxation of ILP_1, the formulation is still inefficient for instances involving processing times, time-lags and makespan with large values. In fact ILP_2 involves $O(nt)$ number of constraints which is much higher that the $O(n^2)$ constraints in ILP_1. Clearly, this is a consequence of the fact that time-indexed formulations are pseudo-polynomial in the number of jobs to be scheduled. However, in chapter 4 we use time-indexed formulations since the problems to tackle involve processing times and time-lags values of few time-slots.

3.4 Tabu Search Algorithm

Due to the NP-hardness of the coupled-operation problem with the fixed permutation of a-operations (or b-operations), we use the alternative representation given by permutation π of all $2n$ operations. We suggest a neighbour generation strategy based on removing both operations of a selected job j and inserting them elsewhere so that the makespan is minimised and the sequence of the remaining $2(n-1)$ operations remains unchanged. An efficient insertion algorithm for finding the optimum re-allocation of job j is based on the disjunctive graph model (see, e.g., [19]) and the short cycle property formulated in [43].

We start with describing the disjunctive graph model and the insertion algorithm (Sec-

tion 3.4.1). Then we describe the neighbourhood structure (Section 3.4.2). Finally, we conclude with the general description of the tabu search algorithm (Section 3.4.3).

3.4.1 Disjunctive Graph Model and the Insertion Algorithm

In this section we present an efficient procedure for solving the problem of inserting two operations of a given job j into a partial schedule with a fixed sequence of operations of the jobs from $N \setminus \{j\}$. We assume that an upper bound $C_{\max}^{UB}$ on the makespan value is given and our task is to find a feasible schedule for that upper bound with job j inserted, if one exists.

The algorithm is based on the disjunctive graph model which can be described as follows. A disjunctive graph $G = (V, A, E, \mathscr{E}, c)$ is a directed graph with the vertex-set V, the set of conjunctive arcs A, the set of disjunctive arcs E, the family $\mathscr{E} \subseteq 2^E$ of disjunctive sets and the arc cost function c.

Set V contains vertices v_i, $i = 1, 2, \ldots, 2n$, representing $2n$ operations of the coupled-operation problem and two additional dummy nodes: origin v_0 and terminal v_*. If v_i represents the a-operation (b-operation) of job j, notation a_j (b_j, respectively) is used in line with v_i.

Conjunctive arcs A characterise the minimum difference in the starting times of the operations they connect: if there is an arc (v_i, v_j), then the starting times s_{v_i} and s_{v_j} should satisfy

$$s_{v_j} \geq s_{v_i} + c_{v_i v_j}.$$

There are four conjunctive arcs for each job $j \in N$:

- two arcs (a_j, b_j) and (b_j, a_j) represent precedence constraints between operations of the same job and their costs $c_{a_j b_j} = p_{a_j} + L_j$ and $c_{b_j a_j} = -(p_{a_j} + L_j)$ specify the fixed distance between the starting times of a_j and b_j;
- one arc (v_0, a_j) from origin v_0 to the a-operation of job j with cost $c_{v_0 a_j} = 0$;
- one arc (b_j, v_*) from the b-operation of job j to the terminal node v_* with cost $c_{b_j v_*} = p_{b_j}$.

One additional conjunctive arc (v_*, v_0) connects the terminal node and the origin; its cost is based on the given value of the makespan upper bound: $c_{v_* v_0} = -C_{\max}^{UB}$, which implies

$$s_{v_0} \geq s_{v_*} - C_{\max}^{UB}$$

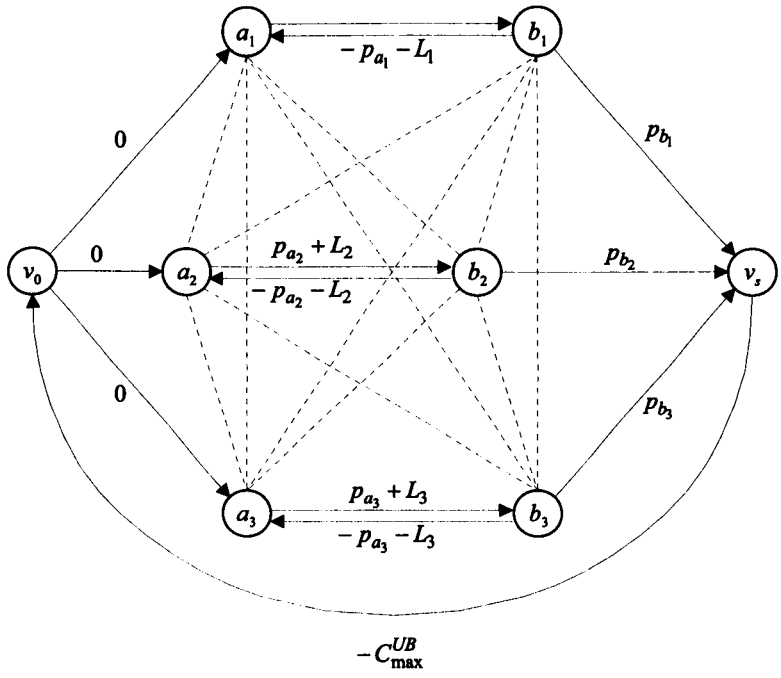

Figure 3.3: Disjunctive graph for three coupled-operation jobs (pairs of disjunctive arcs are represented by dashed lines)

or equivalently

$$C_{\max} = s_{v_*} \leq s_{v_0} + C_{\max}^{UB}.$$

Set E contains pairs of disjunctive arcs $e = (v_i, v_j)$ and $\bar{e} = (v_j, v_i)$ which connect two operations of different jobs. We call $\bar{e}$ the mate of e and vice versa. In a complete solution, one of the disjunctive arcs of the pair is selected specifying the order between the two operations while another arc is dropped. The cost of a disjunctive arc (v_i, v_j) is defined as the processing time of the operation corresponding to node v_i.

The family of disjunctive sets is $\mathscr{E} \subseteq 2^E$ has the meaning that for each pair $e = (v_i, v_j)$ and $\bar{e} = (v_j, v_i)$ at least one of the disjunctive constraints should be satisfied:

$$s_{v_j} \geq s_{v_i} + c_{v_i v_j} \quad \text{or} \quad s_{v_i} \geq s_{v_j} + c_{v_j v_i}.$$

The scheduling decision consists in finding a *selection* of disjunctive arcs $\mathscr{F} \subseteq E$ containing no more than one mate from a pair. Selection $\mathscr{F}$ is *feasible* if it is *complete* (exactly one arc is selected from each pair of arcs e and $\bar{e}$) and *positive acyclic* (digraph $(V, A \cup \mathscr{F}, c)$ does not contain cycles of positive cost). A feasible selection $\mathscr{F}$ specifies a schedule $\mathscr{S}$ in which the starting time of every operation v_i is defined as the longest path from origin v_0 to the corresponding node v_i in digraph $(V, A \cup \mathscr{F}, c)$. The makespan of the

schedule is equal to the length of the *critical path*, i.e., the longest directed path from the origin v_0 to the terminal node v_*. Notice that for an infeasible selection the critical path does not exist since it would have an infinite cost; for a feasible selection a critical path can be found in $O(|V|^2)$ time.

We can now introduce the neighbour generation problem for the coupled-operation problem. It is defined for the current partial solution $\mathscr{S}$ corresponding to selection $\mathscr{F}$ and job $j \in N$ to be inserted in $\mathscr{S}$. We assume that $\mathscr{F}$ is complete and positive acyclic for the subproblem defined by the nodes $V \setminus \{a_j, b_j\}$; otherwise inserting two operations of job j cannot lead to a feasible solution. For the insertion set $J = \{a_j, b_j\}$ we define the set of disjunctive arcs $E(J) \subseteq E$ and the set of conjunctive arcs $A(J) \subseteq A$ which connect nodes in J and the remaining nodes $V \setminus J$. Notice that conjunctive arcs (a_j, b_j) and (b_j, a_j) are not included in $A(J)$.

Definition Given a job j with operations $J = \{a_j, b_j\}$ and a selection $\mathscr{F}$ feasible for the nodes $V \setminus J$ and not containing any disjunctive arcs from $E(J)$, the *job insertion problem* consists in finding a selection $\mathscr{F}^J \subset E(J)$ such that:

- $|\mathscr{F}^J| = \frac{1}{2}|E(J)|$,
- the resulting graph $G = (V, A \cup \mathscr{F} \cup \mathscr{F}^J, c)$ is positive acyclic.

The associated *insertion graph* is denoted by $G^J = (V, A \cup \mathscr{F}, E(J), \mathscr{E}(E(J)), c)$, where $\mathscr{E}(E(J)) \subset \mathscr{E}$ is the family of disjunctive sets defined over $E(J)$.

A feasible selection can be found efficiently by the procedure from [43] if insertion graph G^J has a so called *short cycle property* which states that for any selection $\mathscr{F}^J$, if the corresponding digraph $(V, A \cup \mathscr{F} \cup \mathscr{F}^J, c)$ contains a positive cycle, then it contains a short positive cycle visiting J has exactly once. We first demonstrate that this property is satisfied for the insertion graph of the coupled-operation problem and then describe how the algorithm from [43] can be applied to solving our insertion problem.

Proposition 5. *An insertion graph G^J of the coupled-operation problem has the short-cycle property.*

Proof. Suppose $\mathscr{F}^J$ is a positive cyclic selection with more than two arcs from $E(J)$. We show how a short positive cycle visiting J exactly once can be constructed.

Let Z be a positive cycle with arcs from $A \cup A(J) \cup \mathscr{F}^J$ which visits J twice or more. The possible components of cycle Z are arcs (a_j, b_j), (b_j, a_j) which may appear multiple times in Z and the fragments of the form $a_j P a_j$, $b_j Q b_j$, $a_j R b_j$ and $b_j T a_j$, where P, Q, R and T are the paths consisting of nodes different from $\{a_j, b_j\}$.

If Z has a fragment of the form a_jPa_j (or b_jQb_j) and such a fragment is of positive length, then it is a required short positive cycle; if that fragment has non-positive length, it can be removed from Z so that the remaining cycle is positive and it has less visits to J.

If Z has a fragment of the form a_jQb_j and its length $c(R)$ satisfies

$$c(R) > c_{a_jb_j}, \tag{3.11}$$

then we construct a short cycle $a_jRb_ja_j$: it visits J only once and its length $c(R)+c_{b_ja_j}$ is positive since $c_{b_ja_j} = -c_{a_jb_j}$. Alternatively, if condition (3.11) does not hold, we replace a fragment of type a_jRb_j by a single arc (a_j,b_j) obtaining a new cycle with less visits to J and which length is positive since $c(R) \leq c_{a_jb_j}$.

It is easy to make sure that a similar transformation can be done for fragment b_jRa_j: one has to swap a_j and b_j in the above arguments and replace condition (3.11) by $c(Q) > c_{b_ja_j}$.

Thus considering the fragments of Z with end nodes in $\{a_j,b_j\}$ we either construct a short positive cycle based on that fragment or eliminate a part of cycle Z constructing a new positive cycle with less arcs and less visits to J. Applying this procedure we eventually produce a short positive cycle with exactly one visit to J. ∎

The short cycle property implies that a selection is feasible if and only if it does not include disjunctive arcs of the following two types:

(i) arc $e \in E(J)$ which incurs a positive cycle in the digraph $(V,A \cup \mathscr{F} \cup \{e\},c)$;

(ii) a combination of two arcs $u,v \in E(J)$ which incur a positive cycle in the digraph $(V,A \cup \mathscr{F} \cup \{u,v\},c)$.

This restriction on the choice of disjunctive arcs can be modelled naturally as a *conflict graph* [43]. Formally, a conflict graph $H_{G^J} = (X,U)$ has nodes $X = E(J)$ corresponding to disjunctive arcs of the insertion graph G^J and undirected arcs U of two types:

- loops $(e,e) \in U$ for disjunctive arcs e of type (i),

- undirected arcs $(u,v) \in U$ for every pair of disjunctive arcs u,v of type (ii).

Then any positive acyclic selection corresponds to a stable set in conflict graph H_{G^J}, and the required feasible selection $\mathscr{F}^J$ for graph G^J is the stable set of maximal cardinality in H_{G^J}, $|\mathscr{F}^J| = \frac{1}{2}|E(J)|$. If the maximum cardinality of a stable set is less than $\frac{1}{2}|E(J)|$, then no feasible solution exists.

Due to Proposition 5, H_{G^J} is bipartite with node partition $X = E^+(J) \cup E^-(J)$, where $E^+(J)$ are the arcs from $E(J)$ outgoing from J and $E^-(J)$ are the arcs incoming to J. Clearly, if the insertion graph G^J does not have disjunctive arcs of type (i), then a stable set of cardinality $\frac{1}{2}|E(J)|$ can be selected as $E^+(J)$ or $E^-(J)$. The presence of disjunctive arcs of type (i) introduces additional restrictions; still it is possible to find a stable set efficiently, as suggested in [43], without employing a standard approach based on the minimum cut problem.

Disjunctive arcs of type (i) dictate some limitations on the choice of selection $\mathscr{F}^J$: arc e of type (i) cannot be included in $\mathscr{F}^J$ or equivalently the corresponding node e of the conflict graph cannot be included in the stable set. Hence for each e of type (i) we should include its mate $\overline{e}$ in the stable set. Having selected $\overline{e}$, we need to prohibit all $f \in E(J)$ connected with $\overline{e}$ by arc $(\overline{e}, f)$ in the conflict graph since arcs $\overline{e}$, f are of type (ii). Prohibiting f implies that its mate $\overline{f}$ should be selected and the procedure should be continued for $\overline{f}$.

Having performed all "necessary" selections dictated by disjunctive arcs of type (i), the selected disjunctive arcs and their mates can be excluded from further consideration. The resulting subgraph $\widehat{H}_{G^J} \subseteq H_{G^J}$ is bipartite and it contains only the nodes of type (ii), and a stable set for $\widehat{H}_{G^J}$ can be selected as one of the two the bipartition set.

The correctness of the described approach for the insertion problem with the short cycle property is rigorously proved in [43] and it can be formally presented as the following procedure.

Procedure 'Insert(G^J,J)'

1. Construct the bipartite conflict graph $H_{G^J} = (X, U)$; identify all arcs e of type (i) and include their mates $Q = \{\overline{e} \in E : (e,e) \in U\}$ in $\mathscr{F}^J$.

2. For every $u \in Q$ find all nodes $\{\overline{f}_1, \overline{f}_2, \ldots, \overline{f}_k\}$ reachable in H_{G^J} from u through alternating paths of the form $(u, f_1, \overline{f}_1, f_2, \overline{f}_2, \ldots, f_k, \overline{f}_k)$; denote the set of nodes reachable from Q by Q^*, $Q \subseteq Q^*$, and extend $\mathscr{F}^J$ by including Q^*.

3. If Q^* is not stable, then terminate: no feasible selection exists.

4. Otherwise, consider a subgraph $\widehat{H}_{G^J} = (\widehat{X}, \widehat{U})$ of H_{G^J} by eliminating all nodes Q^*, their mates and associated arcs; define a stable set T in $\widehat{H}_{G^J}$ as $\widehat{X} \cap E^-(J)$ or $\widehat{X} \cap E^-(J)$.
   Extend $\mathscr{F}^J$ by including T.

In the case of the coupled-operation scheduling problem, conflict graph H_{G^J} has $|X| = 8(n-1)$ nodes corresponding to $2(n-1)$ pairs of disjunctive arcs connecting a_j with the remaining $2(n-1)$ nodes and similarly $2(n-1)$ pairs of disjunctive arcs incident to b_j. Step 1 should involve the all-pairs longest path algorithm (e.g., the Floyd-Warshall algorithm, see [7]) for preprocessing and then it checks each disjunctive arc of G^J for property (i) and each combination of two disjunctive arcs for property (ii), which requires $O(n^2)$ time. Steps 2 and 3 can be implemented in $O(n^2)$ time, while Step 4 requires $O(n)$ time. Thus the overall time complexity of the described procedure is $O(n^3)$.

Observe that having introduced the arc (v_*, v_0) of weight $-C^{UB}_{\max}$ in the disjunctive graph model, we guarantee that the 'Insert' procedure finds a feasible schedule (if one exists) with the the makespan no larger than $C^{UB}_{\max}$. In order to find a feasible insertion minimizing the makespan, one can apply the above technique in combination with binary search, considering the makespan values in the interval $[C^{LB}_{\max}, C^{UB}_{\max}]$ defined by the the lower and upper bound of the makespan. One algorithm for calculating lower bounds is discussed in Section 3.6.2.

3.4.2 Neighbourhood Structure

Given a current complete schedule $\mathscr{S}$ specified by a feasible selection $\mathscr{F}$, we define the neighbourhood of $\mathscr{S}$ via removing a critical job from that solution and inserting it optimally without altering the order of other operations. A job is *critical* if one of its operations a_j or b_j or both operations belong to the critical path in the disjunctive graph.

Unlike the earlier approach from [81], instead of re-inserting a chosen critical operation, we re-insert a critical job consisting of two operations ensuring that the resulting solution is feasible and has the minimum makespan. In order to guarantee that the neighbour differs from $\mathscr{S}$, we force one of the disjunctive arcs e belonging to the critical path in $\mathscr{S}$ to be replaced by its mate $\bar{e}$ oriented in the opposite direction. We call e a *disjunctive critical arc* and we select e to be incident to the operation of the critical job chosen for re-insertion. Formally, the neighbour generating procedure can be described as follows.

Procedure 'Neighbour($\mathscr{S}, j, e$)'

1. Define the insertion set $J = \{a_j, b_j\}$ and the insertion graph G^J by removing all disjunctive arcs $E(J)$ incident to operations a_j and b_j from $\mathscr{F}$:

$$G^J = (V, A \cup (\mathscr{F} \setminus E(J)),\ E(J), \mathscr{E}(E(J)), c).$$

2. Include the mate $\overline{e}$ of the given disjunctive critical arc e into the selection and apply procedure 'Insert(G^J,J)' to find a new feasible selection $(\mathscr{F} \setminus E(J)) \cup \{\overline{e}\} \cup \mathscr{F}^J$ for which the makespan value is minimum. Denote the resulting schedule by $\mathscr{S}'_{j,e}$.

3. Return schedule $\mathscr{S}'_{j,e}$ as the neighbour of $\mathscr{S}$.

Observe that procedure 'Insert(G^J,J)' presented in the previous section does not guarantee that the required arc $\overline{e}$ is selected. However, this can be easily ensured if a loop $(e,e) \in U$ is included in the conflict graph H_{G^J} so that Procedure 'Insert' is forced to select $\overline{e}$.

In what follows we are mainly interested in the best possible neighbour $\mathscr{S}'_j$ of the current solution $\mathscr{S}$ obtained via re-inserting job j. In order to find $\mathscr{S}'_j$, we need to identify all disjunctive critical arcs $E' \subset \mathscr{F}$ incident to j, apply procedure 'Neighbour($\mathscr{S}$, j, e)' for each such arc and select the neighbour $\mathscr{S}'_j$ with the smallest makespan value:

$$C_{\max}\left(\mathscr{S}'_j\right) = \min_{e \in E'} \left\{C_{\max}\left(\mathscr{S}'_{j,e}\right)\right\}.$$

Notice that the set E' contains two disjunctive arcs, if job j has one operation on the critical path, or four disjunctive arcs, if both operations of job j belong to the critical path.

Consider now the problem of finding a neighbour of the current solution $\mathscr{S}$ having the smallest possible makespan. The straightforward approach consists in generating neighbour schedules $\mathscr{S}'_j$ for all critical jobs j and finding the required neighbour with the smallest value $C_{\max}\left(\mathscr{S}'_j\right)$. This, however, is computationally expensive due to the high computational cost of the relatively slow procedure 'Insert'.

A possible alternative approach is to calculate some estimates $\sigma\left(\mathscr{S}'_j\right)$ of the makespan value of each neighbour schedule $\mathscr{S}'_j$ assuming that a potentially good neighbour has the lowest estimate. For a fast method, we consider only Step 1 of Procedure 'Neighbour' without taking into account the selected disjunctive critical arc e and the subsequent 'Insert' procedure. The resulting estimate can be used as a quality measure of the insertion graph G^J. The most accurate characteristic of G^J is its makespan, which can be found in $O\left(n^2\right)$ time via the longest path calculation:

$$\sigma_1\left(\mathscr{S}'_j\right) = C_{\max}\left(G^J\right). \tag{3.12}$$

A faster but less accurate characterisation of G^J can be found in $O(1)$ time as

$$\sigma_2\left(\mathscr{S}'_j\right) = C_{\max}\left(\mathscr{S}\right) - \Delta, \tag{3.13}$$

where the decrease value Δ which might be achieved if job j is removed from $(\mathscr{S})$ can take the following values:

$$\Delta = \begin{cases} p_{a_j} + L_j + p_{b_j}, & \text{if operations } a_j \text{ and } b_j \text{ appear in the critical path in the order } a_j, b_j, \\ p_{b_j}, & \text{if operations } a_j \text{ and } b_j \text{ appear in the critical path in the order } b_j, a_j, \\ p_{a_j}, & \text{if only one operation } a_j \text{ appears in the critical path,} \\ p_{b_j}, & \text{if only one operation } b_j \text{ appears in the critical path.} \end{cases}$$

Notice that

$$C_{\max}\left(G^J\right) \leq C_{\max}\left(\mathscr{S}'_j\right)$$

since inserting job j in G^J can only increase $C_{\max}\left(G^J\right)$. Therefore estimate (3.12) can be considered as the lower bound on the value of $C_{\max}\left(\mathscr{S}'_{j,e}\right)$,

$$\sigma_1\left(\mathscr{S}'_j\right) \leq C_{\max}\left(\mathscr{S}'_j\right).$$

However, estimate (3.13) may satisfy

$$\sigma_2\left(\mathscr{S}'_j\right) \leq C_{\max}\left(\mathscr{S}'_j\right) \text{ or } \sigma_2\left(\mathscr{S}'_j\right) \geq C_{\max}\left(\mathscr{S}'_j\right)$$

since either condition

$$\sigma_2\left(\mathscr{S}'_j\right) \leq C_{\max}\left(G^J\right) \text{ or } \sigma_2\left(\mathscr{S}''_j\right) \geq C_{\max}\left(G^J\right)$$

may hold.

Empirical evaluation of the described techniques has demonstrated that generating all neighbour schedules of the current schedule $\mathscr{S}$ and calculating their makespan values is not efficient since it slows down the search procedure. Calculating estimates σ_2 is much faster in comparison with σ_1 and leads to better results as a broader range of the solution space is examined fast enough. Therefore the approach adopted in the implementation and used in computational experiments generates only one potentially good neighbour on the basis of estimate σ_2.

3.4.3 Description of the Tabu Search Implementation

In this section we describe implementation details of our tabu search algorithm. The algorithm maintains *tabu list* $\mathscr{T}$ of the characteristics of the most recent visited solutions to prevent reproducing those solutions and to avoid generating solutions with similar characteristics, see [47] for the general description of the tabu search algorithm.

The visited solutions are represented in the tabu list by their critical paths. For a critical path recorded in the list, we keep two additional characteristics:

- a *hit counter* which specifies how many solutions with that critical path were visited,
- a *serial number* of the last visited solution with that critical path.

Once a new neighbour is generated, its critical path is added to the tabu list, if it does not duplicate an existing entry in the tabu list; otherwise a hit counter of the existing entry is incremented and a serial number is updated.

Whenever the number of entries in the tabu list reaches a given threshold on the list size, the list is halved by removing the entries with the smallest hit counter values (i.e., solutions visited less often); in case of ties, the entries with the smallest serial numbers (i.e., the older solutions) are eliminated first. In the updated list, the hit counters are set to zero for all entries. This elimination strategy guarantees that the tabu list contains critical paths of the solutions which are visited most recently and, in addition, generated most frequently during the search.

While a traditional tabu search algorithm considers one current solution at a time and performs moves from one current solution to its neighbour, in our implementation we maintain a pool $\mathcal{Q}$ of current solutions. Initially, that pool is filled in with solutions generated by the constructive heuristics described in Section 3.5. At the neighbour generation stage, we find estimates of all non-visited neighbours of all current solutions of the pool $\mathcal{Q}$ and generate using procedure 'Insert' one neighbour which has the smallest estimate. As discussed in Section 3.4.2, we use estimate σ_2 to examine all neighbours fast enough. The newly generated neighbour is either rejected, if its critical path is in the tabu list, or, otherwise, it is accepted and included in the pool $\mathcal{Q}$.

Since it is important to have a fast procedure to recognise whether a neighbour has been examined or not, we maintain for each schedule $\mathcal{S} \in Q$ the list $\chi(\mathcal{S})$ of critical jobs of that schedule which re-insertion results in an unvisited neighbour: if $j \in \chi(\mathcal{S})$, then the corresponding neighbour $\mathcal{S}'_j$ of schedule $\mathcal{S}$ has not been considered yet. Formally our implementation of the tabu search algorithm can be described as follows.

Algorithm 'Tabu-Search'

1. Initialize an empty tabu list $\mathcal{T}$ and the pool $\mathcal{Q}$ of current solutions containing a given number of solutions generated by constructive heuristics.
   For each schedule $\mathcal{S} \in \mathcal{Q}$, find its critical path and initiate the list $\chi(\mathcal{S})$ of critical jobs for neighbour generation.

2. Repeat steps 3-4 while the computation time is less than the pre-specified time limit.

3. Use (3.13) to find estimates $\sigma_2\left(\mathscr{S}'_j\right)$ of all non-visited neighbours of the solutions belonging to the pool $\mathscr{Q}$. Find solution $\mathscr{S} \in \mathscr{Q}$ which neighbour $\mathscr{S}'_j$ has the smallest estimate σ_2,

$$\sigma_2\left(\mathscr{S}'_j\right) = \min_{\mathscr{S}\in\mathscr{Q}} \min_{k\in\chi(\mathscr{S})} \left\{\sigma_2\left(\mathscr{S}'_k\right) \mid \mathscr{S}'_k \text{ is a obtained from } \mathscr{S} \text{ by re-inserting } k\right\}.$$

   Generate $\mathscr{S}'_j$ using procedure 'Insert' and remove job j from list $\chi(\mathscr{S})$.
   Replace the record of the best solution found so far, if $\mathscr{S}'_j$ beats it.
   If $\mathscr{S}'_j$ is the last non-visited neighbour of $\mathscr{S}$, remove $\mathscr{S}$ from the pool $\mathscr{Q}$.

4. If the critical path of $\mathscr{S}'_j$ belongs to tabu list $\mathscr{T}$, update the hit counter and the serial number of the corresponding entry in $\mathscr{T}$, mark neighbour $\mathscr{S}'_j$ as visited and eliminate it from further consideration.
   Otherwise perform Steps 4.1-4.2.

   4.1. Add the critical path of $\mathscr{S}'_j$ to $\mathscr{T}$ setting the hit counter to 1 and serial number to the current number of $\mathscr{S}'_j$. If the size of $\mathscr{T}$ reaches a given threshold value, remove half of the entries which have the smallest hit counter values; in case of ties eliminate the entries with the smallest serial numbers.

   4.2. Add $\mathscr{S}'_j$ to the pool of current solutions $\mathscr{Q}$. If the size of $\mathscr{Q}$ reaches a given threshold value, $\mathscr{Q}$ is emptied and refilled with a given number of schedules generated by constructive heuristics.

In order to perform the search operation and adding new records to the tabu list fast enough, we implement it as a radix tree (Patricia trie) with alphabet $\{1,2,\ldots,2n\}$ needed to specify critical operations and their sequences.

3.5 Constructive Heuristics

To the best of our knowledge, there are no heuristics developed specifically for the coupled-operation scheduling problem with fixed time-lags. The most relevant algorithms presented in [81] are aimed at solving a more general version of the problem in which time-lags are flexible, i.e., can vary within given limits (see Section 3.2). Computational experiments show that for the problem with flexible time-lags constructive random

heuristic outperforms local-search algorithms. In this section we adapt the most successful constructive heuristic JOIN-DECOMPOSE [81] to our problem and, in addition, we develop a simple but efficient EARLIEST-FIT dispatching rule. Both heuristics are used to evaluate the performance of tabu search in the computational experiments of Section 3.6.

JOIN-DECOMPOSE (JD) from [81] is a constructive heuristic based on the idea of composite jobs which are created and break down in two stages: the *join* and the *decompose* stage respectively. The algorithm maintains a pool of composite jobs which is initially defined as the set of the individual input jobs. At the join stage, two individual jobs in the pool are selected and replaced by their combination: the new composed job. The procedure is repeated until the jobs in the queue cannot be combined any more and therefore they are placed into the schedule one after another without interleaving. At the decompose stage, combined jobs are broken down into a schedule of individual jobs. In [81], the authors describe 12 rules generating combined jobs and additional rules for selecting the best combined job for the problem with flexible time-lags. The same rules apply also to the case with fixed time-lags by simply setting the flexibility parameter f to zero. In our experiments, JD heuristic is run multiple times inserting some randomness in the selection of the rules to be applied and halting when a given time limit is reached. Repeated application of the algorithm generates many feasible schedules from which the best one is selected.

EARLIEST-FIT (EF) is based on a greedy dispatching rule which schedule jobs one by one. At each iteration, such a rule randomly selects one unscheduled job and includes it into the current partial schedule at the earliest starting time available so that no job already scheduled is postponed and all time-lags are observed. Formally, for a coupled-operation job j, starting time t is feasible if the machine is idle in time intervals $[t, t+p_{a_j}]$ and $[t+p_{a_j}+L_j, t+p_{a_j}+L_j+p_{b_j}]$ where operations a_j and b_j can be processed. In our experiments, EF heuristic is run multiple times in a similar fashion of JD heuristic.

Both algorithms JOIN-DECOMPOSE and EARLIEST-FIT can be implemented in $O(n^2)$ time and are easy to code.

3.6 Computational Experiments

In order to evaluate the performance of our tabu search algorithm, we compare it with the two constructive heuristics JD and EF presented in Section 3.5. JD is a winning algorithm for the coupled-operation problem with flexible time-lags [81] and hence it can be considered as a benchmark. For this reason we mainly repeat the experimental design

from [81] with some additions.

Notice that the computational experiments in [81] investigate the performances of the algorithms for coupled-operation scheduling problem with flexible time-lags. Thus, the time-lags of the problem instances are generated using two parameters: a parameter α to determine the minimum length of the time-lag and a parameter β that establishes the maximum increment the actual time-lag can take over the minimum length (flexibility). For this reason, in the rest of this section, whenever a comparison is made between our results and the ones in [81], we consider only experiments in [81] performed with instances generated setting the flexibility parameter β to zero.

3.6.1 Instance Generation

Computational experiments have been performed on two classes of instances: SIM and MIX. In the SIM class, all jobs have similar time-lags; in the MIX class, job time-lags differ significantly.

In all instances processing times p_{a_j} and p_{b_j} are integers drawn from the uniform distribution,

$$p_{a_j}, p_{b_j} \in [1, 100] \text{ for each } j \in N.$$

In what follows we use the symbol p_{avg} to indicate the average processing time in the instance calculated as

$$p_{avg} = \frac{1}{2n} \sum_{j \in N} \left(p_{a_j} + p_{b_j} \right).$$

The time-lags L_j are also integer values sampled from an uniform distribution depending on a parameter α. For the SIM class the following formula is used:

- $L_j \in [0.9\alpha p_{avg}, \ldots, 1.1\alpha p_{avg}]$

For MIX class, we distinguish between two subclasses of instances, MIX_1 and MIX_2, depending on the time-lag calculation formula used:

- $L_j \in [1, \ldots, \alpha]$ is used for the MIX_1 class (from [81]),
- $L_j \in [1, \ldots, \alpha p_{avg}]$ is used for the MIX_2 class.

Notice that, MIX_2 subclass is not present in computational experiments in [81]. It has been added in our work since instances in MIX_1 appear to be biased by the fact the α-values used in the [81] (i.e. 5,10,25,50) generate time-lags too small in comparison to average processing time (50). As a consequence, the optimal solutions of such instances present a few interleaving jobs and do not represent the complexity of the problem. The

subclass MIX_1 has been kept in our experiments for completeness of the analysis and to allow a one-to-one comparison with results in [81].

For each class SIM, MIX_1 and MIX_2 we generate 10 instances for each combination of parameters $n \in \{20,30,50,100\}$ and α where:

- $\alpha \in \{5,10,25,50\}$ for MIX_1 class.
- $\alpha \in \left\{1, \frac{n}{5}, \frac{2n}{5}\right\}$ for SIM and MIX_2 class.

All algorithms have been coded in C and run on one core of a PC with processor Intel Quad Core 2.5 Ghz with 3 GB of RAM. The parameters of the tabu search algorithm have been tuned after several preliminary experiments: we have set 500 as the maximum length of the tabu list $\mathscr{T}$ and 200 as the maximum length of pool $\mathscr{Q}$. Each time the pool is emptied, 50 random solutions are generated using constructive heuristics from Section 3.5.

The experiments involve two constructive heuristics JD and EF and two versions of tabu search: TS-JD, which uses heuristic JOIN-DECOMPOSE to generate new solutions whenever $\mathscr{Q}$ is empty, and TS-EF, which uses heuristic EARLIEST-FIT to generate new solutions whenever $\mathscr{Q}$ is empty.

3.6.2 Results

The performance of an algorithm on a given instance is measured in terms of the relative deviation ρ from the lower bound:

$$\rho = \frac{C_{\max}^{Best} - LB}{LB}$$

where $C_{\max}^{Best}$ is the smallest makespan found by the algorithm and LB is the lower bound. The performances of an algorithm on a set of instances is then measured as the average value $\overline{\rho}$ of the relative deviation over all the instance considered. Where necessary, we indicate the average time T_{avg} that the algorithm takes to generate the best solutions found over all the instances of the set considered.

For lower bound calculation we use the approach described in [81]. It starts with identifying a subset of jobs $N_1 \subseteq N$ which cannot be interleaved with any other job. Job j is included in this set if none of the sequences (a_k,a_j,b_j,b_k), (a_k,a_j,b_k,b_j), (a_j,a_k,b_j,b_k) or (a_j,a_k,b_k,b_j) is feasible for any $k \in N \setminus \{j\}$. Then the contribution of all jobs from N_1 is

$$LB(N_1) = \sum_{j \in N_1} \left(p_{a_j} + L_j + p_{b_j}\right).$$

The contribution of the remaining jobs $N_2 = N \setminus N_1$, which can be interleaved, is no less than their total processing requirement and it is also no less than the length of the longest job in that set:

$$LB(N_2) = \max\left\{\sum_{j\in N_2}(p_{a_j}+p_{b_j}), \max_{j\in N_2}\{p_{a_j}+L_j+p_{b_j}\}\right\}.$$

The overall lower bound is then

$$LB = LB(N_1) + LB(N_2).$$

In the experiments, we set up a time limit of 600 seconds for tabu search and for multiple runs of each of the two heuristics. Each time a new solution is generated, its makespan is compared with the lower bound *LB* to check if the optimum is found and the program may terminate immediately. This happened frequently only in our experiments with MIX_1 instances.

Table 3.1 summarises the average accuracy of the two versions of the tabu search algorithm TS-EF and TS-JD and the two constructive heuristics EF and JD on the MIX and SIM classes of instances. Algorithms are listed in the order of overall performance measured in terms of the average value $\overline{\rho}$ of the relative ratio ρ calculated over 10 test instances. The second characteristic of each algorithm is the average time (in seconds) of finding the best solution.

Algorithm	Overall		MIX_1		MIX_2		SIM	
	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}
TS-EF	0.084	200	0.056	121	0.100	277	0.120	357
TS-JD	0.090	150	0.057	83	0.120	322	0.128	349
EF	0.198	186	0.071	115	0.250	272	0.336	281
JD	0.292	131	0.062	2	0.547	126	0.607	145

Table 3.1: A summary of the results for all instances

Tabu search algorithms outperform constructive heuristics in terms of the solution quality. JD constructive heuristic tends to generate the best solution early and then continue running until the 600-second limit is reached without any improvement. Each version of tabu search continues improving the best solution for a longer period of time in comparison with the corresponding constructive heuristic. Comparing the behaviour of algorithms on MIX and SIM instances we observe that, as a rule, all algorithms find so-

	Algorithm	MIX ($\beta = 0$)
Best random heuristics	RC	0.073
	C	0.077
Best local search	SMD	0.074
	RMD	0.086
	LHTS	0.093

Table 3.2: Value of $\overline{\rho}$ for MIX instances obtained by the experiment in [81] with parameter $\beta = 0$

lutions closer to the lower bound for MIX instances.

Table 3.2 reports the performances of the best constructive heuristics and local search algorithms presented in [81] evaluated on instances comparable to the MIX_1 class in this chapter ($\beta = 0$). The algorithms presented has been designed for the solution of coupled-operation scheduling problem with flexible time-lags. The heuristics C and RC are the counterparts of our JD heuristic where additional rules are put in place to consider job flexibility parameter during the selection process at the join stage. While in RC multiple schedules are generated applying selection rules randomly, in C heuristic only one schedule is generated using only one rule. The heuristics SMD, RMD and LHTS are local search algorithms based on a reinsertion neighbourhood structure where, give a permutation of the operations, an operation from the critical path is remove and re-inserted in another random position. Differently from our neighbourhood, infeasible permutations can be generated and later discarded. SMD and RMD use such neighbourhood structure repeatedly in descent algorithms starting the search with random solution in RMD and seeded solution in SMD. LTSH algorithm implements such neighbourhood structure in local search framework. It easy to see that, despite the instance data used seem to be biased (see previous comments above), our tabu-search algorithms TS-JD and TS-EF outperform all the best algorithms in [81]. Notice that local search algorithms SMD, RMD and LHT are often unable to find better solutions than those found by constructive heuristics RC and C.

Tables 3.3 and 3.4 illustrate the behavior of the algorithms on instances of different sizes. For both classes of instances, MIX and SIM, tabu search algorithms produce high quality solutions whichever the size of the instance is; the accuracy only slightly deteriorated for larger values of n. Constructive heuristics perform similarly on MIX_1 instances, while they show a substantial deterioration of the solution quality when n in-

Algorithm	SIM			
Input size n	20	30	50	100
TS-EF	0.99	0.112	0.116	0.151
TS-JD	0.101	0.114	0.119	0.178
EF	0.188	0.264	0.377	0.636
JD	0.240	0.287	0.483	1.419

Table 3.3: Value of $\overline{\rho}$ for SIM instances depending on instance size

Algorithm	MIX_1				MIX_2			
Input size n	20	30	50	100	20	30	50	100
TS-EF	0.050	0.059	0.057	0.060	0.076	0.089	0.094	0.140
TS-JD	0.051	0.059	0.059	0.060	0.077	0.090	0.095	0.218
EF	0.056	0.066	0.077	0.086	0.145	0.200	0.272	0.382
JD	0.055	0.066	0.064	0.064	0.175	0.255	0.470	1.287

Table 3.4: Value of $\overline{\rho}$ for MIX instances depending on instance size

creases for MIX_2 and SIM classes. Heuristic JD performs particularly poorly on instances with $n = 100$ jobs resulting in deviation values $\overline{\rho}$ being almost twice the deviation of EF heuristic. Notice that $\overline{\rho}$ values are homogeneous for all the instance sizes in MIX_1. This is an evidence of the bias in MIX_1 instances whose optimal solutions have few interleaving jobs and can be found early in the search.

Tables 3.5, 3.6 and 3.7 illustrate the behavior of the algorithms on instances with different values of α-parameter (see Section 3.6.1). Observe that $\overline{\rho}$-value for constructive heuristics is generally proportionate to the value of α-parameter. Differently, tabu search algorithms running on SIM and MIX_2 instances seem to perform better when $\alpha > 1$ in comparison with $\alpha = 1$. This suggests that our tabu search design is much more powerful than constructive heuristics to generate schedules of good quality even if there are many interleaving jobs. For the MIX_1 class all algorithms reach similar $\overline{\rho}$ due to the biased structure of the input instance mentioned above.

Summarising, computational experiments show that the proposed tabu search algorithms based on job re-insertion provide solutions of higher quality than the algorithms previously known. In particular, they compare favourably with constructive heuristics EF and JD. Notice that JD is the best published algorithm for the coupled-operation problem with flexible time-lags so far. JD performs poorly in comparison to our heuristic EF when

Algorithm	SIM					
α	1		$\frac{n}{5}$		$\frac{2n}{5}$	
	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}
TS-EF	0.183	295	0.082	388	0.094	389
TS-JD	0.188	260	0.085	400	0.110	388
EF	0.281	276	0.334	261	0.484	307
JD	0.212	64	0.533	185	1.076	188

Table 3.5: Value of $\overline{\rho}$ for SIM instances depending on α-value

Algorithm	MIX_1							
α	5		10		25		50	
	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}
TS-EF	0.006	60	0.028	59	0.081	128	0.111	273
TS-JD	0.006	15	0.028	17	0.083	130	0.113	172
JD	0.006	0	0.028	0	0.083	0	0.132	8
EF	0.006	2	0.028	59	0.089	128	0.162	273

Table 3.6: Value of $\overline{\rho}$ for MIX_1 instances depending on α-value

Algorithm	MIX_2					
α	1		$\frac{n}{5}$		$\frac{2n}{5}$	
	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}	$\overline{\rho}$	T_{avg}
TS-EF	0.113	157	0.080	318	0.106	358
TS-JD	0.113	184	0.082	399	0.165	383
EF	0.161	185	0.277	341	0.310	292
JD	0.129	2	0.447	182	1.065	195

Table 3.7: Value of $\overline{\rho}$ for MIX_2 instances depending on α-value

run for a long period of time since it generates solutions of a similar structure. Differently, heuristic EF explores broadly the solution space generating a wider variety of solutions in comparison with JD. However, JD heuristic results to be an appropriate choice when a reasonably good solution for special structured instances is needed within a few seconds.

3.7 Discussion

In this chapter we develop a tabu search algorithm for scheduling n coupled-operation jobs with fixed time-lags on a single machine. In order to explore possible solution representations, we study the special case of the problem with the fixed order of the first n operations and establish its NP-hardness even for the special case of unit-time operations. Due to this reason we select an alternative solution representation based on the permutation of all $2n$ operations.

For the neighbour generation strategy we adopt the insertion technique proposed in [43]. We show that an optimal insertion of a job in a partial schedule can be found in polynomial time. The algorithm uses the disjunctive graph model and exploits a so-called short cycle property.

The tabu search algorithm we propose differs from the traditional version by maintaining a pool of current solutions instead of a single solution. Due to this reason, it explores a broader part of the solution space. With a larger size of the neighbourhood, fast selection of a potentially good neighbour is particularly important. To achieve this we calculate makespan estimates for each of the possible neighbours and we select the candidate with the smallest estimate.

The following two enhancements are introduced in tabu list implementation: (1) solutions are represented by their critical paths and (2) the entries of the tabu list are prioritised depending on how often the corresponding solutions are generated; whenever there is a need to reduce the size of the tabu list, the entries with the lowest frequency are removed first.

Computational experiments demonstrate that the proposed tabu search is effective for solving the coupled-operation problem with fixed time-lags. It outperforms the earlier heuristics developed for the more general problem with flexible time-lags, in particular the join-and-decompose (JD) heuristic, which is considered to be the most successful among those studied in [81]. We believe, that the main reason of good performance of our tabu search algorithm is related to the neighbour generation strategy which always produces a feasible solution while neighbour generation strategy from [81] often results in infeasible solutions.

To summarise, the main outcomes of this chapter include the analysis of possible solution representations, an efficient neighbour generation strategy and two enhancements for tabu search implementation. It should be noted, that the improvements achieved for the coupled-operation problem with fixed time-lags cannot be immediately transferred to the more general problem with flexible time-lags. In particular, the insertion technique for neighbour generations is not applicable to the flexible time-lag version of the problem since the short cycle property is not satisfied for it.

The outcomes of this work open new research directions. One research direction concerns the study of the other compact representation for feasible schedules. In particular it would important to know the complexity of the problem of constructing a feasible schedule given a permutation π_a of a-operations and a permutation π_b of b-operations. The existence of a polynomial-time algorithm would allow the design of a solution algorithm visiting a reduced solution space. Another research direction would be a study aimed at determining whether the neighbourhood based on the job reinsertion problem presented in this section is opt-connected or not. This would be a stronger analytical argument to the efficiency of the algorithm proposed.

Chapter 4

Scheduling Patient Appointments via Multi-level Template

4.1 Overview and Problem Definition

In this chapter we study a multi-criteria optimization problem which appears in the context of scheduling chemotherapy appointments. The scenario we consider is typical for many chemotherapy outpatient clinics. In its current form, the problem was formulated by the Institute of Oncology at St. James's University Hospital in Leeds, U.K.

Chemotherapy is an important and widely used therapy to treat cancer. It consists of cyclic administrations of drug mixtures delivered to patients under rigid protocols called *regimens*. In an outpatient clinic, chemotherapy is mainly administered orally or through intravenous systems in a day case unit. The two main characteristics of a regimen are drug combinations and delivery patterns. Once a regimen is prescribed, a patient should visit the clinic at treatment days which are separated by a fixed number of rest days. During a treatment day, a patient undergoes treatment procedures which involve several nurse activities separated by time intervals of fixed length. Without loss of generality, in this chapter we assume that each nurse activity is a 15-min time slot in which a nurse is acting for the treatment of a patient.

A regimen is characterized by two types of patterns: a *multi-day pattern* given as a sequence of treatment days and rest days in-between and, for each treatment day, an

intra-day pattern given as a sequence of nurse activities and time-lags in-between, each of which consists of several 15-minute time slots. Typically, nurse activities are related to setting up an intravenous machine, its re-setting for a new drug mixture or a checkup of patient's state. The working day of the clinic is split into 15-minute time slots and this is the main time unit of an intra-day schedule.

Upon arrival to the clinic, a patient is allocated to a nurse for all treatment procedures of the current one-day session. It is important that the same nurse delivers all treatment procedures to an assigned patient during one treatment day; however, it is not required that the patient is treated by the same nurse at other treatment days.

A nurse may treat several patients during a day if the related activities do not overlap in time. If treatment procedures assigned to one nurse lead to a *clash* in the nurse's schedule with more than one activity planned for the same time slot, the nurse needs to speed up the normal operations, delay the starting time of patient appointment or request the cover of an additional nurse(s). It is required that each nurse has at least 30-minute break (two time slots) in the middle of the day.

In the rest of the chapter we use the term "*appointment*" for one treatment day of a patient. An appointment is scheduled if its date and starting time are fixed and a nurse is allocated to perform all associated treatment activities on that day. An appointment is partially scheduled if only some of these parameters (day, time, nurse allocation) are fixed.

In what follows we describe the problem considered in this chapter by the use of an example. All the requirements and constraints defined have been gathered from the clinic.

Example 1. Consider a multi-day schedule for nine patients P1, ..., P9 is which is shown in Fig. 4.1 (a) with treatment days marked by dark boxes. No treatments are planned for weekend days which are dashed. The fragments of schedules for three days and one nurse are shown in Fig. 4.1 (b). The first two intra-day schedules have clashing activities resulting in a solution of poor quality since clashing treatments cannot be delivered by a single nurse.

In order to characterise the effect of clashing activities, we use the following two measures of an intra-day schedule for day d and nurse n: the *number of clashing activities* $\Omega_{d,n}$ and the *maximum clash density* $\Delta_{d,n}$. The value of $\Omega_{d,n}$ is defined as the minimum number of activities which, if removed from the schedule of day d, result in a clash-free schedule for nurse n. The value of $\Delta_{d,n}$ is defined as the maximum number of activities assigned to one nurse in one time slot.

In the example shown in Fig. 4.1 (b), the schedule for the second day (d =08/07/2008) has clashing activities in time slots 9:15, 9:30, 12:30, 13:30 and 14:45. The schedule

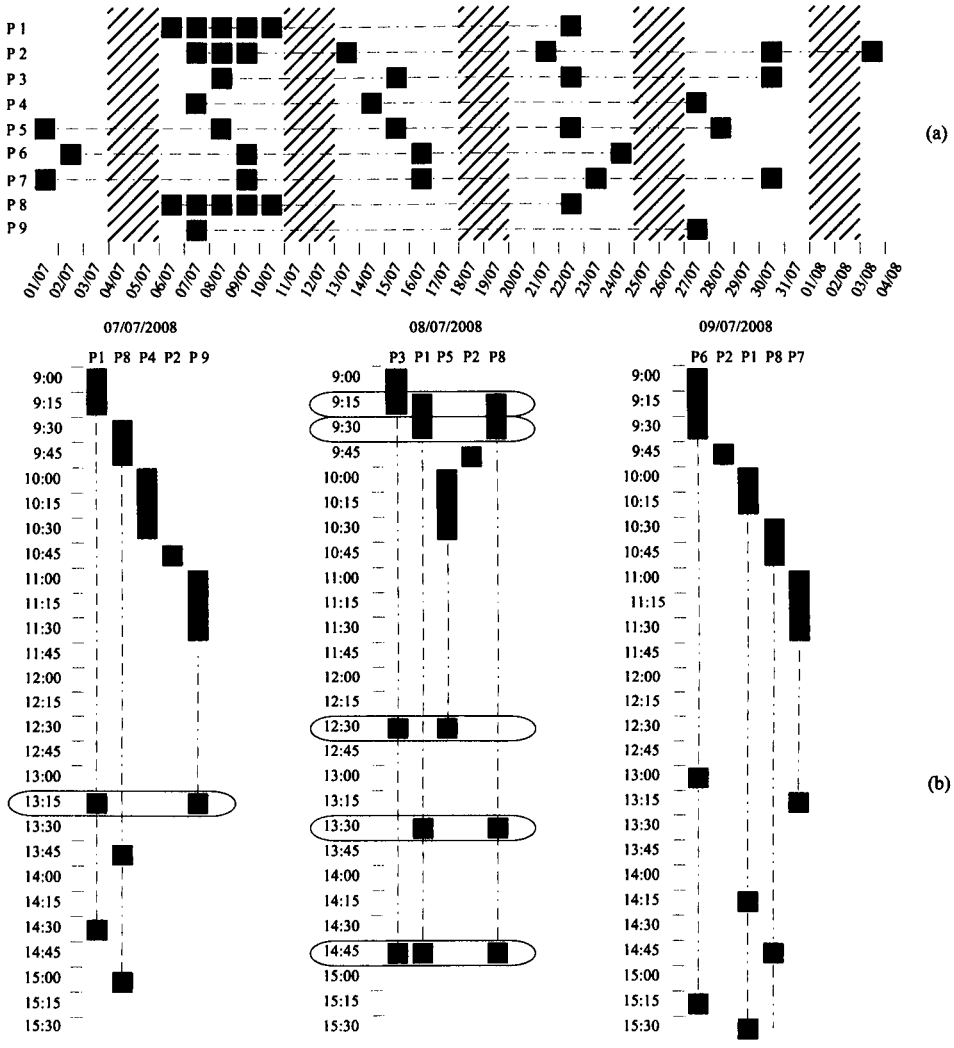

Figure 4.1: An example of a multi-day schedule and three intra-day schedules, each schedule for one nurse

becomes clash free if we remove, e.g., the following activities: four activities of patient P8 assigned to time slots 9:15, 9:30, 13:30, 14:45, two activities of patient P1 assigned to time slots 9:15, 14:45, and one activity of patient P5 assigned to time slot 12:30, which implies that $\Omega_{d,n} = 7$. The second characteristic $\Delta_{d,n} = 3$ reflects the fact that there are three clashing activities in time slots 9:15 and 14:45, and this is the maximum number of activities assigned to one time slot.

The main decisions associated with scheduling appointments for one patient are as follows:

D1: selecting the date of the first appointment, which in fact fixes the dates of all subsequent appointments of the associated multi-day pattern;

D2: selecting the starting time of each appointment; this fixes the time slots for all treatment activities needed for the patient on the visit day;

D3: allocating nurses, one nurse per patient per visit day, to perform all treatment procedures for that patient on the day.

The main outcome of the above decision making process is a *multi-day schedule* combined with a series of *intra-day schedules*, one for each day of the time horizon. These schedules fully describe all nurse activities in the clinic.

There are a number of metrics which characterise the quality of the combined schedule.

F1: The average number of waiting days for all patients, where the number of waiting days for a patient is measured as the delay from a target date to the day of the first appointment. In this metric, the target date of the first visit is predefined by patient's requirements or by the clinic waiting targets.

F2: The maximum clash density Δ is calculated as the maximum among $\Delta_{d,n}$-values for all nurses $n \in N$ and all days $d \in H$ of the selected time horizon H, i.e. $\Delta = \max_{d \in H, n \in N} \Delta_{d,n}$.

F3: The total number of clashes Ω for all nurses is calculated as the sum of the $\Omega_{d,n}$-values for all nurses over the time horizon H, i.e. $\Omega = \sum_{d \in H, n \in N} \Omega_{d,n}$.

One more quality measure is related to the pharmacy, which supplies drugs for chemotherapy treatments. While some drugs are available off-the-shelf, others need to be prepared in the hospital pharmacy. Since the shelf-life of drugs can be limited, it is often required to produce some drugs just before their administration. Due to this, a schedule can be efficient in terms of characteristics F1-F3, but might be poor in terms of the pharmacy constraints. Although in this study we mainly focus on metrics F1-F3, the approach we propose is very well suited for further enhancements which can take into account drug preparation.

With increasing number of cancer patients, chemotherapy departments are under pressure to provide more efficient and qualitatively better service. The World Health Organization [1] and the Department of Health, UK [72] suggest a redesign of the delivery of chemotherapy treatments and set up new targets in terms of patients' waiting times and improved quality of service. In this chapter we show that advanced scheduling algorithms can help in providing timely treatments to larger numbers of patients within shorter

time-frames, balancing workloads of medical staff and reducing the costs of treatments. Automated scheduling may also help in efficient rescheduling which is often needed due to unpredictable events such as changes in patients treatment plans and clinic resources.

Outpatient appointment scheduling in health care has been studied since the early fifties when Bailey [8] and Lin and Haley [65] published their first results on block appointment systems where a single queue model was used to minimise patient waiting times. Since then, the main stream of research focuses on finding appropriate rules for assigning patients' appointments to time intervals in order to minimise patients' waiting times, idle time and overtime of physicians or trade-offs between these objectives, see, e.g., Fetter and Thompson [34].

In more recent research, the models are extended by considering additional unexpected events such as no-shows, walk-ins and emergency treatments. A popular approach aimed at reducing effects of those unexpected events is related to overbooking, see, e.g. Vissers [100]. A comprehensive survey with a detailed classification of scheduling models for booking outpatient appointments can be found in Cayirli and Veral [25].

A new trend of research is related to appointment scheduling systems with specific time requirements and restrictive constraints on resource availability, see Gupta and Denton [46]. For example, there is an increasing number of publications on scheduling radiotherapy appointments. The special feature of radiotherapy treatments is a multi-day pattern consisting of several consecutive days; additional constraints are related to the limited number of available treatment machines, see Conforti et al. [29] and Petrovic et al. [79].

Similar to scheduling radiotherapy appointments, the problem of scheduling chemotherapy appointments also deals with multi-day treatment patterns and limited resources, but multi-day patterns usually have rest days in-between visit days. In addition, patient's treatment procedures during one visit day should follow a complex intra-day pattern. In spite of its importance, the problem of scheduling chemotherapy appointments has received less attention of OR researchers than scheduling radiotherapy appointments. The majority of publications appear in medical journals and they discuss mainly scheduling strategies adopted by practitioners.

As a rule, scheduling decisions are made by a qualified nurse who selects appointment dates and times based on the knowledge of treatment procedures, resource availability and personal experience. We refer to Turkcan et al. [96] who provides an overview of publications in oncology journals. Here we only mention the article by Dobish [31] which describes a scheduling strategy based on creating manually a weekly template schedule. The template is used as a basis for scheduling patients arriving over time. Positive feed-

back of practitioners discussed in [31] and the potential of using a template for scheduling patients in an uncertain environment have inspired our work. Notice that the strategy suggested in [31] is of empirical nature, while in this study the idea of template schedule is developed into a formal optimization model with advanced features.

We are aware of only one publication by Turkcan et al. [96] who consider optimization aspects of delivering chemotherapy treatments. There are a number of points of difference between the model from [96] and the one studied in this chapter: intra-day patterns are treated in [96] as contiguous blocks of time rather than sequences of nurse activities separated by idle intervals; nurse's workload is restricted in [96] by acuity levels of allocated patients while in our model nurse's workload is estimated taking into account actual intra-day patterns of allocated patients and possible clashes between them.

The solution approaches are also different. The approach proposed in this chapter constructs a dynamic plan for all anticipated appointments of the time horizon; such a plan is then frequently adjusted and tuned in accordance with changes in demand and patients' state of health. All appointment dates and times are scheduled and communicated to a patient at the time a patient is admitted to the clinic. In [96] scheduling is performed at regular times for intervals in-between scheduling sessions. The decisions are made for available patients only, which implies that patients arriving in-between scheduling sessions should always wait for at least next session or even longer to get their first appointment. Moreover, appointment times for subsequent visits may not be available all at once as each scheduling session considers a limited time interval until the next session.

In spite of all the differences, both studies, by [96] and ours, indicate that the problem of scheduling chemotherapy appointments has multiple constraints, complex objectives and requires further study.

There is a stream of research related to other aspects of chemotherapy treatments. In particular, OR techniques are used in the chemotherapy context for constructing effective treatment plans which define drug dosages and their administration protocols, see Agur et al. [5], Ochoa et al. [71] and Petrovic et al. [78]; the output of such models are intra-day and multi-day patterns tuned for individual patients rather than actual schedules for delivering those treatments.

This study can be considered as the first attempt to develop an optimization model for scheduling patients' appointments which should follow intra-day and multi-day patterns. We explore the benefits of using a template schedule to fix appointment dates and times for patients arriving over time, but unlike the earlier study by Dobish [31], the advanced features of our approach are based on mathematical analysis of the model rather than empirical observations. The *multilevel template schedule* we produce determines possible

dates and time slots at which future appointments can be scheduled. The appointments of the template are grouped into levels with a view of producing an efficient running schedule with even nurse workload and the smallest possible number of clashes. The *running schedule* is created by considering arriving patients one by one; for each new patient all required appointments are assigned to appropriate time slots of the template schedule matching the prescribed multi-day and intra-day patterns. An additional rescheduling procedure is used to reoptimized each intra-day schedule on a treatment day or shortly beforehand. The key stages of the planning process are modelled as integer linear programs and solved using CPLEX solver.

The chapter is organised as follows. In Section 4.2 we introduce the concepts of template schedule and running schedule and we give a general overview of our approach. In Section 4.3 we formulate two integer linear programs (ILPs) for generating a template schedule: one ILP to define appointment dates for future patients and another one to define appointment times for each day of the template schedule. In Section 4.4 we describe how the template schedule is used for creating a running schedule for arriving patients. Section 4.5 explains how daily rescheduling is done. In Section 4.7 we evaluate the proposed approach via computational experiments performed on two scenarios typical for a real-world clinic. Finally, conclusions are discussed in Section 4.8.

4.2 Notation and General Idea of the Solution Approach

We are given a time horizon H in which patients of a set P have to be treated in accordance with their multi-day and intra-day patterns. It is assumed that patients arrive during the first days H_a of time horizon H, which we call the *arrival period*, $H_a \subset H$. The overall length of H is sufficiently large so that if a new patient arriving within H_a is scheduled with the largest possible acceptable delay, all visits of the corresponding multi-day pattern can be scheduled within the remaining part $H \backslash H_a$ of the time horizon.

The approach we propose consists of four stages:

Stage 1. Forecasting the arrival rate of new patients for each regimen;

Stage 2. Creating the template schedule based on the forecast;

Stage 3. Producing a running schedule for actual patients arriving over time using the template;

Stage 4. Daily rescheduling to re-optimise nurse allocation and to take into account changes in patients' treatment plans.

Stages 1-2 are performed in advance, prior to the first day of the arrival period H_a, and the generated template covers the whole time horizon H. Stages 3-4 are run on a daily basis.

In order to ensure a smooth transition from one time horizon to the next one, we adopt the idea of rolling time horizon. First an initial template schedule is created for time horizon H. A running schedule is generated and maintained during a part of the arrival period H_a (say, for one week). After that the time horizon is shifted to start at the current date, the template schedule is re-calculated for the new time horizon and the same scheduling approach is applied to accommodate patients arriving during the new arrival period H_a keeping the appointments of the previously scheduled patients unchanged.

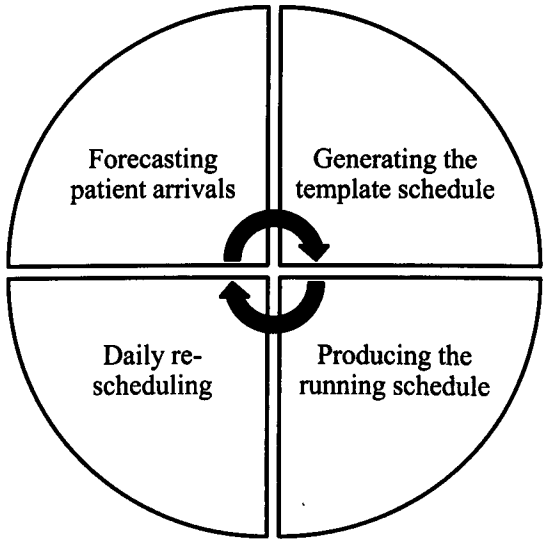

Figure 4.2: Representation of the 4-stage approach

Stage 1: Forecasting patient arrivals. The first stage is aimed at forecasting the arrival rate of patients at the clinic for the arrival period H_a. The output of this stage is the expected number of patients λ_r for each regimen $r \in R$ arriving within one week of the arrival period H_a and the distribution function characterising patient arrival over time.

In this chapter we focus mainly on scheduling procedures rather than on forecasting techniques. In line with the traditional approach often adopted in similar scenarios, we assume that the arrival of patients with a regimen r is a Poisson process with mean and variance equal to λ_r, see, e.g., Brahimi and Worthington [15], Cayirli at Veral [25], Ogulata et al. [73], Vermeulen et al. [99], Wijewickrama and Takakuwa [101] and Williams

et al. [103]. The value of λ_r is estimated using the recorded historical data under the assumption that the patient arrival rate for the new arrival period H_a does not substantially differ from the past.

Stage 2: Generating a template schedule. The second stage is aimed at producing the template schedule consisting of appointments for a set of *artificial patients* $P^{\mathscr{A}}$. In such a schedule the dates, the times and the nurse assignments for the appointments of artificial patients (or *artificial appointments*) are fixed. During the booking process in Stage 3, the dates, the times and the nurses assignments of artificial appointments become dates, times and the nurses assignments of arriving patients. Thus, artificial appointments are placeholders for appointments of real patients arriving over the time.

Stage 2 consists of two parts:

Stage 2A: Generating data describing arrival of artificial patients;

Stage 2B: Producing a multilevel template schedule as a solution to the problem of scheduling artificial patients.

Stage 2A is based on the output of the forecasting stage. It generates, for each regimen $r \in R$, the number of artificial patients with that regimen and, for each artificial patient, the target date of the first visit. Stage 2A deliberately produces the data for more artificial patients than the number of expected patients, keeping the proportion of artificial patients with different regimens the same as the patient proportion in the forecast. The main objective is to produce a template schedule in Stage 2B with an overestimated number of appointments, which should provide more flexibility for booking appointments for arriving patients in Stage 3 and for handling unexpected arrivals.

Stage 2B schedules appointments of artificial patients in terms of decision D1-D3 defined in Section 4.1 and produces a multilevel template schedule. The template can be seen as a combination of a multi-day schedule for time horizon H with a series of one-day schedules assigned to different nurses.

Due to the overestimated number of artificial patients, intra-day schedules of the template may have a large number of unavoidable clashing activities; still such a template leads to a successful running schedule at Stages 3-4. In fact, the multilevel feature of the template described below ensures that, when a running schedule is constructed, the number of clashing activities is as small as possible.

Appointments of the template schedule are grouped into so-called *density sets* according to the maximum clash density $\Delta_{d,n}$ of nurse activities scheduled for each day. Formally, the appointments of artificial patients of a day $d \in H$ assigned to the same nurse n

are grouped into density sets $\mathscr{L}_{d,n}^{(1)}, \mathscr{L}_{d,n}^{(2)}, \ldots, \mathscr{L}_{d,n}^{(K)}$, where the upper index k, $1 \leq k \leq K$, denotes a *density level* defined as the maximum clash density $\Delta_{d,n}$ of appointments $\mathscr{L}_{d,n}^{(k)}$. By definition the density sets for one intra-day schedule of one nurse are nested:

$$\mathscr{L}_{d,n}^{(1)} \subseteq \mathscr{L}_{d,n}^{(2)} \subseteq \cdots \subseteq \mathscr{L}_{d,n}^{(K)}.$$

In particular, if we keep only those appointments which belong to $\mathscr{L}_{d,n}^{(1)}$ removing all other appointments from the intra-day schedule of nurse n, then the resulting schedule does not have clashing activities, i.e., $\Delta_{d,n} = 1$. In general, for any k, $1 \leq k \leq K$, a partial schedule consisting of appointments from the set $\mathscr{L}_{d,n}^{(k)}$ may have clashes of the maximum density $\Delta_{d,n} = k$. Although the total number of density levels K can be as large as the number of artificial patients scheduled on one day, in our experiments on real world data the template schedule produced has no more than 5 density levels, so that $K \leq 5$ (see Section 4.7).

Notice that the density levels into which a schedule can be divided are not unique. For example consider a simple case in which two clashing appointments are divided into two density sets. The only appointment belonging to the first density set can be any of the two clashing appointments. The objective functions that are introduced later in the chapter will restrict the possible combinations of appointments which can be assigned to density classes.

In the example shown in Fig. 4.1, the second one-day schedule corresponding to day $d = 08/07/2008$, if considered as a template schedule for nurse n, can be split into density sets as follows:

$$\begin{aligned}
\mathscr{L}_{d,n}^{(1)} &= \{\text{P1}, \text{P2}, \text{P5}\}, \\
\mathscr{L}_{d,n}^{(2)} &= \mathscr{L}_{d,n}^{(1)} \cup \{\text{P8}\} = \{\text{P1}, \text{P2}, \text{P5}, \text{P8}\}, \\
\mathscr{L}_{d,n}^{(3)} &= \mathscr{L}_{d,n}^{(2)} \cup \{\text{P3}\} = \{\text{P1}, \text{P2}, \text{P5}, \text{P8}, \text{P3}\}.
\end{aligned}$$

To distinguish between the regimens, we denote the set of appointments of a regimen $r \in R$ belonging to a density set $\mathscr{L}_{d,n}^{(k)}$ by $\mathscr{L}_{r,d,n}^{(k)}$. In the above example, if patients P1 and P8 have the same regimen r_1, while the regimens of patients P2, P3, P5 are r_2, r_3, r_4, respectively, then each of the sets $\mathscr{L}_{r_i,d,n}^{(k)}$ is as follows:

$$\begin{array}{llllll}
\mathscr{L}_{d,n}^{(1)}: & \mathscr{L}_{r_1,d,n}^{(1)} = \{\text{P1}\}, & \mathscr{L}_{r_2,d,n}^{(1)} = \{\text{P2}\}, & \mathscr{L}_{r_3,d,n}^{(1)} = \emptyset, & \mathscr{L}_{r_4,d,n}^{(1)} = \{\text{P5}\}, \\
\mathscr{L}_{d,n}^{(2)}: & \mathscr{L}_{r_1,d,n}^{(2)} = \{\text{P1}, \text{P8}\}, & \mathscr{L}_{r_2,d,n}^{(2)} = \{\text{P2}\}, & \mathscr{L}_{r_3,d,n}^{(2)} = \emptyset, & \mathscr{L}_{r_4,d,n}^{(2)} = \{\text{P5}\}, \\
\mathscr{L}_{d,n}^{(3)}: & \mathscr{L}_{r_1,d,n}^{(3)} = \{\text{P1}, \text{P8}\}, & \mathscr{L}_{r_2,d,n}^{(3)} = \{\text{P2}\}, & \mathscr{L}_{r_3,d,n}^{(3)} = \{\text{P3}\}, & \mathscr{L}_{r_4,d,n}^{(3)} = \{\text{P5}\}.
\end{array}$$

As we show in Sections 4.3-4.4, density levels assigned to appointments of the template schedule in Stage 2, play a fundamental role in Stage 3.

Stage 3: Producing the running schedule. The third stage deals with daily arrivals of new patients P. The main objective is to fix appointment dates and times for arriving patients using the template schedule generated in Stage 2. For each newly arrived patient $p \in P$ with a regimen r, the algorithm finds a matching combination of pre-scheduled appointments of the same regimen in the template schedule such that the multi-day pattern of patient p matches the selected appointments. The selected matching appointments of the template do not necessarily belong to a single artificial patient; instead they may correspond to different parts of multi-day patterns of several artificial patients with the required regimen.

If the overestimation rate for the number of artificial patients is selected appropriately in Stage 2, the template schedule contains a sufficient number of artificial appointments for each regimen so that there always exist matching artificial patients scheduled in the template schedule after the arrival of actual patient p. Moreover, there is some flexibility in selecting a match for an actual patient, which allows to keep the number of clashes and patient waiting times at minimum. In particular, if a decision is adopted to schedule an actual patient on day $d \in H$, the algorithm gives preference in the template schedule to the pre-scheduled appointments belonging to the set

$$\mathscr{L}_{r,d}^{(k)} = \cup_{n \in N_d} \mathscr{L}_{r,d,n}^{(k)},$$

with the lowest density level k, where N_d is the set of nurses available on day d. For example, set $\mathscr{L}_{r,d}^{(2)}$ is explored only if there are no available pre-scheduled matching appointments in set $\mathscr{L}_{r,d}^{(1)}$. The next set $\mathscr{L}_{r,d}^{(3)}$ is considered only if there are no available pre-scheduled matching appointments in $\mathscr{L}_{r,d}^{(2)}$, etc. Additional priority rules are used if there are more than one matching pre-scheduled intra-day patterns in the template belonging to the same density set.

Stage 4: Daily rescheduling. Although the running schedule generated in Stage 3 fixes appointment dates and times and nurse allocation for all patients, the quality of the intra-day schedule can be further improved via re-optimization. At this stage, the number of clashing activities of the running schedule is reduced by changing allocation of patients to nurses and allowing small delays (within certain limits) in starting times of scheduled appointments.

4.3 Generating the Template Schedule

Stage 2 includes two parts: generating data for artificial patients (Stage 2A) and scheduling them in a template schedule (Stage 2B).

At Stage 2A we are given, for each regimen $r \in R$, the estimated patient arrival rate λ_r. It specifies the number of patients expected to arrive during one week of the arrival period $H_a \subset H$. The output data of Stage 2A consists of

- for each regimen $r \in R$, the set $P_r^{\mathscr{A}}$ of artificial patients that must be scheduled in the template schedule;
- for each artificial patient $p \in \cup_{r \in R} P_r^{\mathscr{A}}$, the target day t_p of the first appointment.

The above data provides the input for Stage 2B. In addition, it also includes nurse availability for each day $d \in H$. The latter is given as the set N_d of nurses available on that day, and working time for each nurse $n \in N_d$ given as a set of 15-minute time slots $H_{n,d}$ of day d when nurse n is available.

The output data of Stage 2B is the multilevel template schedule. It can be seen as a combination of

- a multi-day schedule which specifies for each day $d \in H$ the set $P_d^{\mathscr{A}}$ of artificial patients to be treated on that day; that set can be further split into subsets $P_{r,d}^{\mathscr{A}}$, $r \in R$, each of which includes only the patients with regimen r to be treated on that day;
- an intra-day schedule for each day $d \in H$, which specifies for each patient $p \in P_d^{\mathscr{A}}$ the allocated nurse $n_p \in N_d$ and the appointment starting time; it also specifies allocation of all appointments of artificial patients $P_{r,d}^{\mathscr{A}}$ into density sets $\mathscr{L}_{r,d,n}^{(k)}$, $r \in R$, $n \in N_d$, $k = 1, \ldots, K$, see Section 4.2 for the definition.

In what follows we describe how Stages 2A and 2B can be implemented.

4.3.1 Generating Data for Artificial Patients

As described in Section 4.2, in order to achieve a high quality running schedule, we produce a template schedule for a larger number of artificial patients than expected. In particular, the number of artificial patients $|P_r^{\mathscr{A}}|$ with regimen r for the arrival period consisting of $|H_a|$ working days is defined as

$$|P_r^{\mathscr{A}}| = \xi \lambda_r \times \frac{|H_a|}{5}, \tag{4.1}$$

where $\frac{|H_a|}{5}$ represents the number of weeks in H_a and $\xi \geq 1$ is an overestimation rate, whose value is determined empirically. The main purpose is to ensure that for each actual arriving patient considered in Stage 3 there can be found a sufficient number of matching artificial patients with the same regimen to select from, so that the incurred waiting time and the number of clashes are within acceptable limits.

An additional adjustment is made for those regimens which happen rarely. In particular, if $\xi\lambda_r < 1$ for a regimen r, then we assign $|P_r^{\mathscr{A}}|$ a higher value:

$$|P_r^{\mathscr{A}}| = \frac{|H_a|}{5}, \tag{4.2}$$

i.e., we force at least one artificial patient with regimen r to be scheduled each week. Such an adjustment ensures that at Stage 3, when an actual patient with a rare regimen r arrives, the waiting time will not be too large, whichever week the patient arrives.

Having defined the number of patients $|P_r^{\mathscr{A}}|$ for each regimen $r \in R$, we generate for each artificial patient $p \in P_r^{\mathscr{A}}$ a target day $t_p \in H_a$ of the first appointment. The values t_p, $p \in P_r$, are uniformly distributed in H. These values become the main input for generating the template schedule in Stage 2B. In the resulting schedule, the day d_p of the first appointment of patient p is selected as close as possible to t_p and either option $d_p \geq t_p$ or $d_p < t_p$ is acceptable.

4.3.2 Quality Metrics of the Template Schedule

The overall performance of our four-stage solution approach is measured in terms of the quality of the running schedule with respect to the objectives F1, F2, F3 introduced in Section 4.1. In order to achieve a successful running schedule, we propose a number of metrics for the template schedule: two metrics U and V characterise the template schedule at the multi-day level and three metrics X_d, Y_d and Z_d characterise the template schedule at the intra-day level for each day d of time horizon H. Each of the metric is represented by a well defined objective function defined below. In the creation of the template schedule the objective functions are optimized in the order defined by a hierarchy described later in this section.

The first metric U specifies the *maximum daily workload excess* of the clinic for the whole template schedule defined over the time horizon H. This metric considers for each day $d \in H$ the difference between the clinic workload W_d and its daily capacity C_d. The workload W_d is the total number of 15-minute nurse activities which appear in the intra-day schedule of the template. It can be calculated by counting for each nurse n the number $W_{n,d}$ of activities performed by that nurse on day d and summing up those values for all

nurses $n \in N_d$:

$$W_d = \sum_{n \in N_d} W_{n,d}.$$

The daily capacity C_d is the maximum number of activities which can be performed by available nurses:

$$C_d = \sum_{n \in N_d} |H_{n,d}|, \tag{4.3}$$

where $H_{n,d}$ specifies working time of nurse n on day d given as a set of 15-minute time slots when the nurse is available.

With an overestimated number of artificial patients and a limited number of nurses, some days of the template schedule may have activities which cannot be covered by available nurses, so that $W_d > C_d$. Then the workload excess of day d is $\max\{W_d - C_d, 0\}$. The maximum daily workload excess U for the whole template schedule is defined for time horizon H as

$$U = \max_{d \in H} \{\max\{W_d - C_d, 0\}\}. \tag{4.4}$$

It is desirable to keep the workload excess as small as possible distributing it evenly over time horizon H.

Stages 3 and 4 of our approach use two additional characteristics related to daily workload excess: the *relative workload* $\hat{W}_{n,d}$ of nurse n on day d, which quantifies the proportion of time the nurse delivers treatments to patients:

$$\hat{W}_{n,d} = \frac{W_{n,d}}{|H_{n,d}|}, \tag{4.5}$$

and the *average relative workload* $\hat{W}_d$ of all nurses working on day d:

$$\hat{W}_d = \frac{\sum_{n \in N_d} W_{n,d}}{\sum_{n \in N_d} |H_{n,d}|} = \frac{W_d}{C_d}. \tag{4.6}$$

Characteristic $\hat{W}_{n,d}$ is used at Stage 4 when daily rescheduling is done aimed at balancing the workloads of different nurses. Characteristic $\hat{W}_d$ is used at Stage 3 when the best possible options are selected from the template schedule to assign appointments of actual patients keeping the maximum relative daily workload as small as possible and evenly distributed over time horizon H.

To define metric V, we calculate for each artificial patient $p \in P^{\mathscr{A}}$ the deviation $|d_p - t_p|$ of the date of the first appointment d_p from the target day t_p, and take the average

of these values:

$$V = \frac{1}{|P^{\mathscr{A}}|} \sum_{p \in P^{\mathscr{A}}} |d_p - t_p| . \tag{4.7}$$

Observe that the definition of V includes the case in which the first visit day d_p of an artificial patient p precedes the target day t_p. When comparing two template schedules, the one with a smaller value of the metric U is preferred. In case of ties than metric V determines the best between the two template schedules.

We now turn to one-day metrics X_d, Y_d and Z_d. These metrics characterise the quality of an intra-day schedule of day d of the template and they are closely related to the notion of a density set introduced in Section 4.2. Each metric X_d, Y_d and Z_d is in fact a collection of metrics $X_d^{(k)}$, $Y_d^{(k)}$ and $Z_d^{(k)}$, respectively, defined for density levels $k = 1, 2, \ldots, K$.

Consider an intra-day schedule for day $d \in H$, with artificial patients $P_d^{\mathscr{A}}$ treated on that day, which are split into subsets $P_{r,d,n}^{\mathscr{A}}$ depending on regimen r and allocated nurse n. In the intra-day schedule of nurse n, appointments of patients $P_{r,d,n}^{\mathscr{A}}$ are scheduled and allocated to sets $\mathscr{L}_{r,d,n}^{(k)}$, $1 \le k \le K$, belonging to the density set $\mathscr{L}_{d,n}^{(k)}$, see Section 4.2 for the definition.

For the partial intra-day schedule consisting of appointments from $\mathscr{L}_{d,n}^{(k)}$ we define $\Omega_{d,n}^{(k)}$ as the total number of 15-minute activities, which, if removed from that partial schedule, result in a clash-free schedule.

Using the above notations, we formally introduce three metrics $X_d^{(k)}$, $Y_d^{(k)}$ and $Z_d^{(k)}$ for a density level k and then provide their interpretation and justification. The metrics are defined as

$$\begin{aligned} X_d^{(k)} &= \min_{r \in R, n \in N_d} \left\{ \frac{|\mathscr{L}_{r,d,n}^{(k)}|}{|P_{r,d,n}^{\mathscr{A}}|} \right\} && \text{(to be maximised)}, \\ Y_d^{(k)} &= \sum_{r \in R} \alpha_{r,d} \sum_{n \in N_d} \left| \mathscr{L}_{r,d,n}^{(k)} \right| && \text{(to be maximised)}, \\ Z_d^{(k)} &= \sum_{n \in N_d} \Omega_{d,n}^{(k)} && \text{(to be minimised)}, \end{aligned} \tag{4.8}$$

where parameter $\alpha_{r,d}$ is an additional weight characteristic defined empirically.

Metric $X_d^{(k)}$ measures the proportion of artificial appointments of different regimens allocated to $\mathscr{L}_{d,n}^{(k)}$. Maximising $X_d^{(k)}$ ensures a fair representation of appointments of regimens in density set $\mathscr{L}_{d,n}^{(k)}$, so that in the resulting template schedule every regimen has appointments in $\mathscr{L}_{d,n}^{(k)}$ and no regimen is overlooked in favour of another regimen.

Metric $Y_d^{(k)}$ counts the total number of appointments allocated to density set $\mathscr{L}_{d,n}^{(k)}$, in the case of unit weights $\alpha_{r,d}$, or the weighted number of appointments allocated to that density set, otherwise. In the weighted version of metric $Y_d^{(k)}$, the most frequent regimens r have higher weights, so that maximising $Y_d^{(k)}$ is aimed at allocating as many

appointments as possible to the density set of level k giving preference to the appointments of the most frequent regimens. In our experiments, we set $\alpha_{r,d} = \left|P^{\mathscr{A}}_{r,d,n}\right|$.

Finally, metric $Z_d^{(k)}$ measures the overall number of clashing activities determined by appointments in $\mathscr{L}_{d,n}^{(k)}$. Clearly, that value should be as small as possible.

If several intra-day schedules are compared in terms of metrics (4.8), then in the first place, we give preference to a schedule with the highest proportion $X_d^{(1)}$ of appointments assigned to density level $k = 1$ for each regimen $r \in R$ and each nurse $n \in N_d$. Among the schedules with the same value $X_d^{(1)}$, we give preference to those schedules having the largest total weighted number $Y_d^{(1)}$ of appointments allocated to density level $k = 1$. If there are still several schedules equivalent in terms of $X_d^{(1)}$ and $Y_d^{(1)}$, the preferred schedule has the smallest number $Z_d^{(1)}$ of clashing activities. The comparison then continues for density level $k = 2$, considering metrics $X_d^{(2)}$, $Y_d^{(2)}$ and $Z_d^{(2)}$ in this order. The similar approach is applied for higher density levels $k = 3, \ldots, K$.

The order of consideration of density levels starting from $k = 1$, proceeding to $k = 2$ and so on up to $k = K$, is in agreement with the procedure used in Stage 3. The latter procedure explores for each actual patient density level $k = 1$, then density level $k = 2$, etc., in order to ensure that matching appointments are found in the lowest possible density level of the template.

Example 2. Consider 9 artificial patients P1, P2, P3, P5, P8, P10, P11, P12 and P13 that should be treated on day d by nurse n. The nurse working day starts at 9:00, finishes at 15:45 and consists of 15-minute time intervals numbered from 1 to 27. Patients' regimens and intra-day patterns are given in Table 4.1. An intra-day pattern is specified as a sequence of time slots, numbered from 1, which require nurse actions. If an appointment is scheduled to start in time slot $t = 1$ (at 9:00), then the intra-day pattern incurs nurse activities in time slots listed in the pattern; if an appointment is scheduled to start at time $t > 1$, the intra-day pattern incurs nurse activities in time slots listed in the pattern incremented by $t - 1$.

The workload excess $\max\{W_d - C_d, 0\}$ of the selected day d is 2 since all activities require $W_d = 29$ time slots while the nurse can only cover $C_d = 27$ time slots.

Consider two template schedules S_1 and S_2 given by Table 4.2 and graphically represented as the first two schedules in Fig. 4.3. There are two density sets for each schedule $\mathscr{L}_{d,n}^{(1)}$ and $\mathscr{L}_{d,n}^{(2)}$ which are specified in Table 4.2 and marked in Fig. 4.3. The maximum clash density of both schedule S_1 and S_2 is $\Delta_{d,n} = 2$, while the number of clashing activities is $\Omega_{d,n} = 3$ for S_1 and $\Omega_{d,n} = 5$ for S_2. The values of all metrics for schedules S_1 and S_2 are shown in Table 4.3, where metric $Y_d^{(k)}$ is calculated with weights $\alpha_{r,d}$ set equal to $|P^{\mathscr{A}}_{r,d,n}|$.

Appointment	Regimen	Intra-day Pattern	$\lvert P_{r,d}^{\mathscr{A}} \rvert$
P1 P8	r_1	1,2,18,23	2
P2	r_2	1	1
P3	r_3	1,2,15,24	1
P5 P10 P11	r_4	1,2,3,11	3
P12 P13	r_5	1,2	2

Table 4.1: Input data for the template schedule for day d

Table 4.2: Appointment starting times for two template schedules S_1 and S_2 and their density sets

S_1	P1	P3	P8	P12	P10	P2	P11	P5	P13
Starting time-slot	1	3	5	7	9	12	14	15	20
$\mathscr{L}_{d,n}^{(1)}$	*	*	*	*	*	*	*		*
$\mathscr{L}_{d,n}^{(2)}$	*	*	*	*	*	*	*	*	*

S_2	P1	P3	P8	P2	P5	P11	P10	P13	P12
Starting time-slot	1	3	5	8	10	11	14	18	22
$\mathscr{L}_{d,n}^{(1)}$	*	*	*	*	*		*		
$\mathscr{L}_{d,n}^{(2)}$	*	*	*	*	*	*	*	*	*

Table 4.3: Performance metrics of template schedules S_1 and S_2

	level k	$X_d^{(k)}$	$Y_d^{(k)}$	$Z_d^{(k)}$
S_1	$k=1$	0.67	16	0
	$k=2$	1	19	3
S_2	$k=1$	0	12	0
	$k=2$	1	19	5

According to metric $X_d^{(1)}$, schedule S_1 is preferred to schedule S_2 since the proportion of appointments allocated to density level $k=1$ is 0.67 for S_1 and 0 for S_2.

Consider now schedule S_1' which is a modification of schedule S_1, where the appointment for patient P5 starts in time slot 16, as shown in Fig. 4.3. The density sets $\mathscr{L}_{r,d,n}^{(k)}$ are the same for schedules S_1' and S_1, so that S_1 and S_1' are equivalent in terms of metrics $X_d^{(1)}$

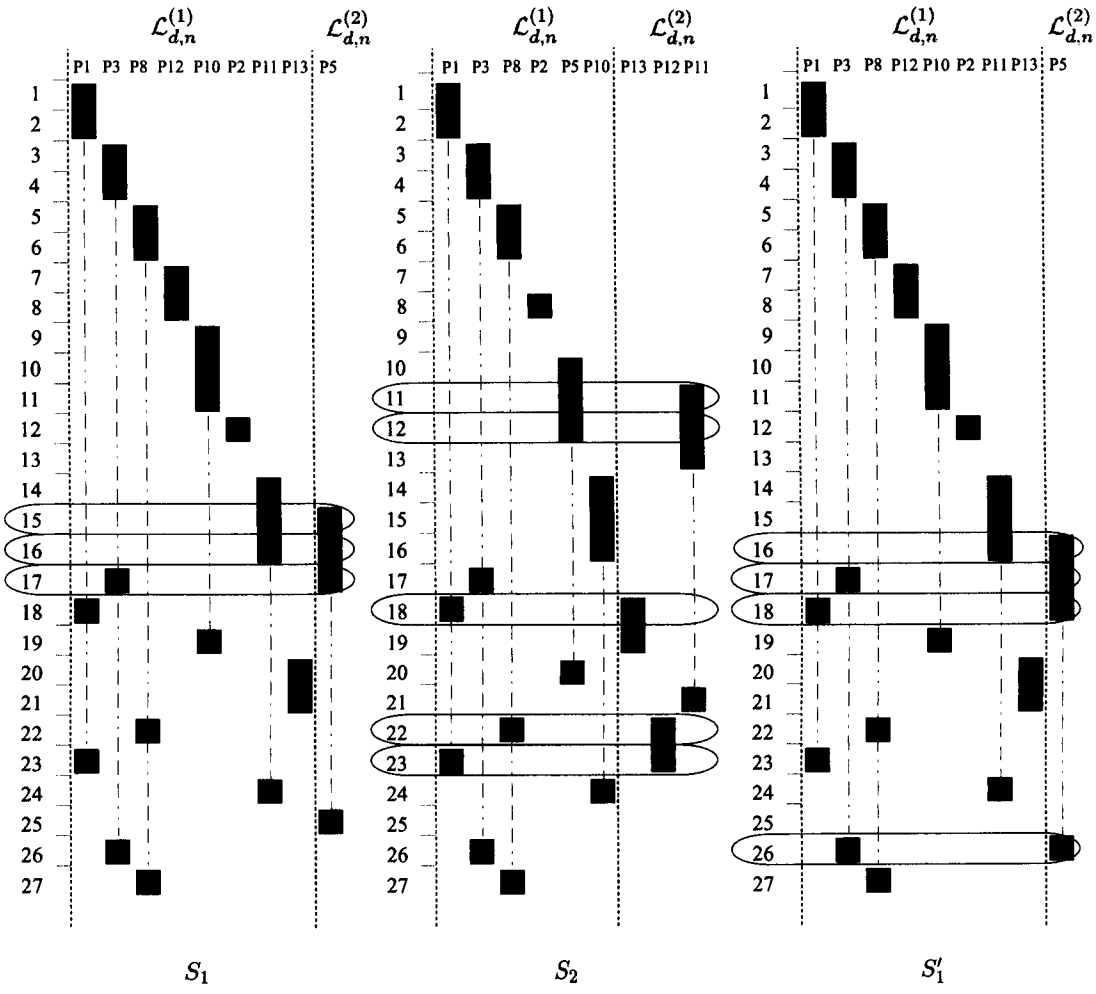

Figure 4.3: Template schedules S_1 and S_2 given by Table 4.2 and schedule S_1' obtained by modifying S_1

and $Y_d^{(1)}$. Still schedule S_1 is preferable in comparison with S_1' due to the metric $Z_d^{(1)}$, as the overall number of clashing activities in density set $\mathscr{L}_{d,n}^{(2)}$ is 3 for schedule S_1 and 4 for schedule S_1'.

4.3.3 Template Schedule Generation

A template schedule is obtained as a solution to the problem of scheduling appointments of artificial patients $P^{\mathscr{A}}$ over time horizon H. The integrated problem of constructing multi-day and intra-day schedules simultaneously appears to be cumbersome due to its size and the complex combination of multi-day and intra-day patterns. On the other hand, it can be naturally decomposed into the following subproblems:

- one subproblem of generating a multi-day schedule for the whole time horizon H;
- $|H|$ subproblems of generating intra-day schedules for each day $d \in H$.

The general idea of this approach is illustrated in Fig. 4.4.

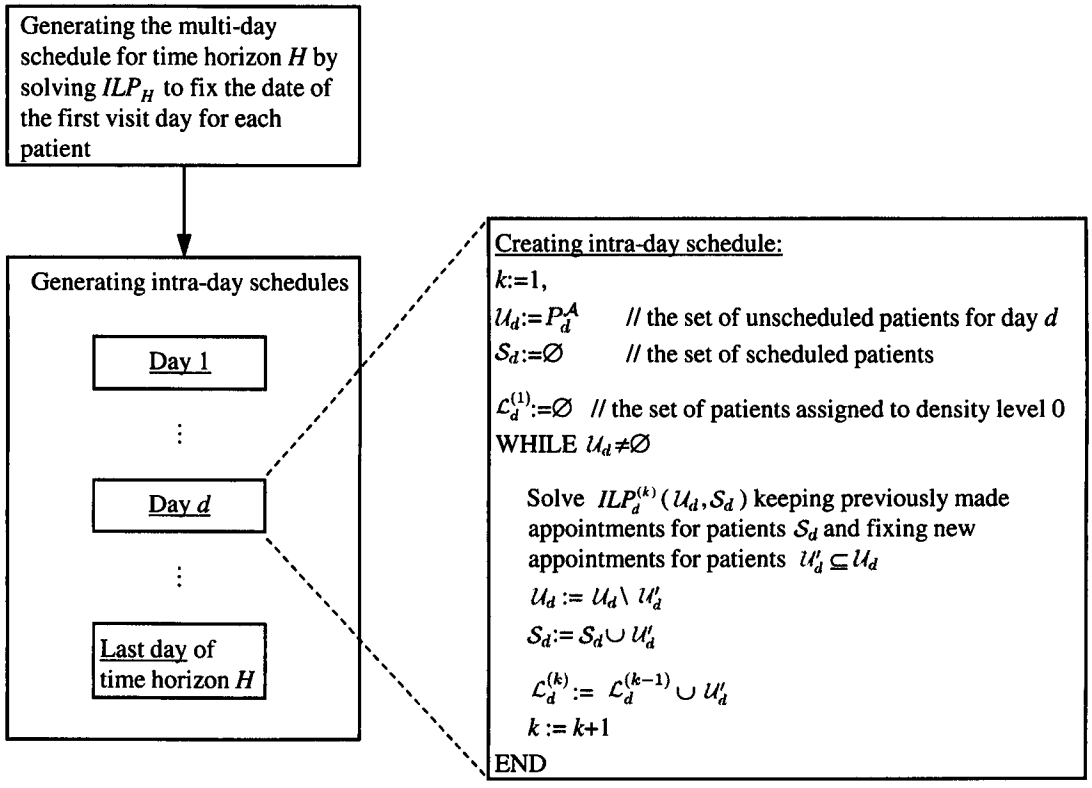

Figure 4.4: Subproblems and algorithms for generating the template schedule

The multi-day subproblem is formulated as an integer linear program ILP_H defined over time horizon H with time-indexed decision variables

$$x_{p,s} = \begin{cases} 1, & \text{if the first appointment of artificial patient } p \text{ is scheduled on day } s, \\ 0, & \text{otherwise.} \end{cases}$$

For each combination of $s \in H$ and $d \in H$, we define the set of artificial patients $\mathcal{P}_{s,d}^{\mathcal{A}}$ having an appointment on day d if the first visit-day is s, $s \leq d$; in addition for each patient $p \in P^{\mathcal{A}}$ we define constant $w_{p,s,d}$ representing the number of nurse activities necessary to treat that patient on day d if the first visit day is s. Constants $w_{p,s,d}$ are determined by the information given by multi-day and intra-day patterns.

Then metrics U and V defined by (4.4) and (4.7) can be represented in the form

$$U = \max_{d\in H}\left\{\max\left\{\sum_{s\in H}\sum_{p\in\mathscr{P}^{\mathscr{A}}_{s,d}}\left(w_{p,s,d}x_{p,s}\right)-C_d,0\right\}\right\},$$
$$V = \frac{1}{|P^{\mathscr{A}}|}\sum_{p\in P^{\mathscr{A}}}\sum_{s\in H}|s-t_p|\,x_{p,s}.$$

We intend to find a solution with the smallest possible value of V among the solutions with the smallest possible value of U. Denoting the lexicographical minimisation of U and V by lex[min$U$, min$V$], we formulate the integer linear program ILP_H for the multi-day problem as follows:

$$ILP_H: \quad \text{lex} \quad [\min U,\ \min V]$$

$$\text{s.t.} \quad U \geq \sum_{s\in H}\sum_{p\in\mathscr{P}^{\mathscr{A}}_{s,d}} w_{p,s,d}x_{p,s}-C_d, \quad d\in H,$$
$$U \geq 0,$$
$$V = \frac{1}{|P^{\mathscr{A}}|}\sum_{p\in P^{\mathscr{A}}}\sum_{s\in H}|s-t_p|\,x_{ps},$$
$$\sum_{s\in H} x_{p,s} = 1, \qquad p\in P^{\mathscr{A}},$$
$$x_{p,s}\in\{0,1\}, \qquad p\in P^{\mathscr{A}},\ s\in H.$$

Having found the solution to problem ILP_H, we generate input data for intra-day problems: for each day $d\in H$ and each regimen r, we construct the set of artificial patients $P^{\mathscr{A}}_{r,d}$ which should visit the clinic on day d and define $P^{\mathscr{A}}_d = \cup_{r\in R}P^{\mathscr{A}}_{r,d}$.

The intra-day problem consists in selecting the starting times for all appointments of patients $P^{\mathscr{A}}_d$ on day d and finding nurse allocation. The suggested approach considers a series of integer linear programs $ILP^{(1)}_d, ILP^{(2)}_d, \ldots, ILP^{(K)}_d$, as shown in Fig. 4.4. During the solution process the set of artificial patients $P^{\mathscr{A}}_d$ is partitioned into the subsets $\mathscr{S}_d$ and $\mathscr{U}_d$, which represent the sets of scheduled and unscheduled patients, respectively. Initially $\mathscr{U}_d = P^{\mathscr{A}}_d$ and $\mathscr{S}_d = \emptyset$.

The solution is found iteratively starting from density level $k=1$. The algorithm fixes the starting times for patients $\mathscr{U}'_d\subseteq\mathscr{U}_d$ by optimizing functions $X^{(k)}_d$, $Y^{(k)}_d$ and $Z^{(k)}_d$; the set $\mathscr{U}_d$ is then updated by moving patients $\mathscr{U}'_d$ to the set of scheduled patients $\mathscr{S}_d$. The corresponding appointments form the density set $\mathscr{L}^{(1)}_d$. Proceeding to density level $k=2$ with updated sets $\mathscr{U}_d$ and $\mathscr{S}_d$, the algorithm keeps previously fixed appointments and finds the starting times for new patients which are then added to $\mathscr{S}_d$. The density set

$\mathscr{L}_d^{(2)}$ is now formed as the union of all previously scheduled appointments $\mathscr{L}_d^{(1)}$ and the appointments scheduled for $k = 2$. The subsequent density sets are considered in a similar fashion until the appointments of all patients are scheduled and $\mathscr{U}_d = \emptyset$.

For each density level k, the intra-day subproblem for day $d \in H$ is formulated as an integer linear program $ILP_d^{(k)}$ with variables

$$x_{p,s,n,d} = \begin{cases} 1, & \text{if artificial patient } p \in P_{r,d}^{\mathscr{A}} \text{ is assigned to nurse} \\ & n \in N_d \text{ with the first treatment procedure in time slot } s, \\ 0, & \text{otherwise.} \end{cases}$$

Initially, $k = 1$ and all x-variables are free. In subsequent iterations k, $k > 1$, the set of scheduled patients $\mathscr{S}_d$ is non-empty and for $p \in \mathscr{S}_d$, the patient's starting time s_p and allocated nurse n_p are fixed, which implies that the value of the corresponding x-variable is also fixed, $x_{p,s_p,n_p,d} = 1$.

Finding now the expressions for the first two metrics $X_d^{(k)}$ and $Y_d^{(k)}$ from (4.8) is straightforward:

$$X_d^{(k)} = \min_{r \in R} \left\{ \frac{1}{|P_{r,d}^{\mathscr{A}}|} \sum_{p \in P_{r,d}^{\mathscr{A}}} \sum_{n \in N_d} \sum_{s \in H_{n,d}} x_{p,s,n,d} \right\},$$

$$Y_d^{(k)} = \sum_{r \in R} \left(\alpha_{r,d} \sum_{p \in P_{r,d}^{\mathscr{A}}} \sum_{n \in N_d} \sum_{s \in H_{n,d}} x_{p,s,n,d} \right).$$

In order to calculate the third metric $Z_d^{(k)}$, we introduce auxiliary variables $y_{i,n,d}$ for each nurse $n \in N_d$ and each time slot $i \in H_{n,d}$ of the nurse's available time slots of day d. The smallest value of $y_{i,n,d}$ is defined as 1 and it represents either of the following two situations which do not incur any penalties:

- nurse n is free in time slot i;
- there is exactly one activity assigned to nurse n in time slot i.

Any greater value $y_{i,n,d} > 1$ represents the number of activities assigned to nurse n in time slot i.

Using the variables $y_{i,n,d}$, the third metric $Z_d^{(k)}$ can be defined as

$$Z_d^{(k)} = \sum_{n \in N_d} \sum_{i \in H_{n,d}} (y_{i,n,d} - 1).$$

For calculating $y_{i,n,d}$, we define the set of patients $P^{\mathscr{A}}_{s,i,d}$ having an activity to be performed in time slot i if the appointment starts at time s, and we set

$$y_{i,n,d} \geq \sum_{s=1}^{\tau_{p,n,d}} \sum_{p \in P^{\mathscr{A}}_{s,i,d}} x_{p,s,n,d}$$

with an additional constraint

$$1 \leq y_{i,n,d} \leq k.$$

Here the constant $\tau_{p,n,d}$ defines the latest time slot when the intra-day pattern of patient p may start so that all activities of the pattern can be performed by nurse n within the working hours.

As a result, we obtain the following integer linear program $ILP^{(k)}_d$:

$$ILP^{(k)}_d(\mathscr{U}_d, \mathscr{S}_d): \quad \text{lex}\left[\max X^{(k)}_d,\ \max Y^{(k)}_d,\ \min Z^{(k)}_d\right]$$

s.t.

$$X^{(k)}_d \leq \frac{1}{|P^{\mathscr{A}}_{r,d}|} \sum_{p \in P^{\mathscr{A}}_{r,d}} \sum_{n \in N_d} \sum_{s \in H_{n,d}} x_{p,s,n,d}, \qquad r \in R,$$

$$Y^{(k)}_d = \sum_{r \in R} \left(\alpha_{r,d} \sum_{p \in P^{\mathscr{A}}_{r,d}} \sum_{n \in N_d} \sum_{s \in H_{n,d}} x_{p,s,n,d} \right),$$

$$Z^{(k)}_d = \sum_{n \in N_d} \sum_{i \in H_{n,d}} \left(y_{i,n,d} - 1\right),$$

$$y_{i,n,d} \geq \sum_{s=1}^{\tau_{p,n,d}} \sum_{p \in P^{\mathscr{A}}_{s,i,d}} x_{p,s,n,d}, \qquad n \in N_d, \quad i \in H_{n,d},$$

$$1 \leq y_{i,n,d} \leq k, \qquad n \in N_d, \quad i \in H_{n,d},$$

$$x_{p,s_p,n_p,d} = 1, \qquad p \in \mathscr{S}_d,$$

$$\sum_{n \in N_d} \sum_{s \in H_{n,d}} x_{p,s,n,d} \leq 1, \qquad p \in P^{\mathscr{A}}_d,$$

$$x_{p,s,n,d} \in \{0,1\}, \qquad p \in P^{\mathscr{A}}_d, \quad s \in H_{n,d}, \quad n \in N_d.$$

After a solution to problem $ILP^{(k)}_d(\mathscr{U}_d, \mathscr{S}_d)$ is found, the sets of scheduled and unscheduled patients $\mathscr{S}_d$ and $\mathscr{U}_d$ are updated so that $\mathscr{U}_d = \mathscr{U}_d \setminus \mathscr{U}'_d$ and $\mathscr{S}_d = \mathscr{S}_d \cup \mathscr{U}'_d$. A patient p is scheduled, if there exists a combination of s and n such that $x_{p,s,n,d} = 1$; we denote such a combination by s_p and n_p.

The formulation can be extended to include various additional constraints. For example, if a meal break is one of the requirements, then $ILP^{(k)}_d$ can be forced to reserve for each nurse a set of contiguous time slots (two 15-minute slots in our experiments) in the middle of the day such that no patient is treated at that time, leaving the nurse free from

treatment activities for that period. This can be achieved by introducing variables $b_{t,n,d}$, which define for nurse n the starting time of a break on day d:

$$b_{t,n,d} = \begin{cases} 1, & \text{if the meal break of nurse } n \in N_d \text{ starts at time } t \in H_{n,d}, \\ 0, & \text{otherwise.} \end{cases}$$

Let $M_{n,d}$ be a set of possible starting times of the meal break of nurse $n \in N_d$. For a patient $p \in P_d^{\mathscr{A}}$, introduce a set $B_{p,t,n,d}$ of appointment starting times which incur a treatment activity during the meal break of nurse n starting at time $t \in M_{n,d}$. Clearly, if the meal break of nurse n is scheduled at time t (i.e., $b_{t,n,d} = 1$), patient $p \in P_d^{\mathscr{A}}$ cannot be allocated to nurse n with appointment starting time $s \in B_{p,t,n,d}$. Then, $ILP_d^{(k)}$ can be adjusted by introducing the following additional constraints:

$$\sum_{p \in P_d^{\mathscr{A}}} \sum_{s \in B_{p,t,n,d}} x_{p,s,n,d} + b_{t,n,d} \leq 1, \quad n \in N_d,\ t \in M_{n,d},$$

$$\sum_{t \in M_{n,d}} b_{t,n,d} = 1, \qquad n \in N_d.$$

Observe that it is easy to ensure that the meal break is of the required length by defining the set $B_{p,t,n,d}$ appropriately.

Suppose the template schedule is created for artificial patients $P^{\mathscr{A}}$ to be treated in time horizon H. In Stage 3, described in the next section, the template schedule is used for generating and maintaining a running schedule for actual patients P by fixing their appointments in time slots reserved for artificial patients. Using the approach of the rolling time horizon, at some point within time horizon H a new template schedule is produced for a new time horizon H' overlapping with H. The new template schedule is found as a solution to a multi-day problem $ILP_{H'}$ and a series of intra-day problems $ILP_d^{(k)}$ for $d \in H'$, $1 \leq k \leq K$, which are similar to problems ILP_H and $ILP_d^{(k)}$, $d \in H$, $1 \leq k \leq K$, with one point of difference: the appointments of actual patients which have been fixed in the running schedule must be kept. This can be achieved by adding the constraints $x_{p,s,n,d} = 1$ for each pre-scheduled actual patient $p \in P$, which have to visit the clinic on day d to be treated by nurse n with the first visit day s, $s \leq d$. On the other hand, the appointments of artificial patients which are not booked so far for actual patients can be rescheduled at no cost, so that the corresponding x-variables are free.

4.4 Running Schedule: Creating and Maintaining

Once a template schedule is generated, it is used to assign appointment dates and times for arriving patients. The booking process for a new patient consists in identifying in the template schedule available appointments of artificial patients with the same regimen and selecting the combination of appointments that satisfies the multi-day pattern and additional preference conditions. Then the dates and starting times of chosen appointments become dates and times of appointments of the new patient. In this section we propose an algorithm that selects appointments from the template schedule with the aim of optimizing the quality of the generated running schedule in terms of metrics F1-F3 (see Section 4.1).

Consider a newly arrived patient $p \in P$ with regimen $r \in R$ whose first visit date d_p has to be within a given time window $[d_p^{\min}, d_p^{\max}]$ determined by patient's requirements and clinic waiting targets. The proposed algorithm searches for a possible date d_p in $[d_p^{\min}, d_p^{\max}]$ that allows the selection of appointments from the template schedule satisfying the multi-day pattern of regimen r.

One simple strategy is to choose in the template a single artificial patient with regimen r and to assign all appointments of actual patient p to the dates and times of pre-scheduled appointments of that artificial patient.

In order to achieve a solution of higher quality, we consider the template schedule as a collection of artificial appointments breaking the link to artificial patients and their multi-day patterns. Then, in order to book appointments for actual patient p with regimen r, we consider all artificial appointments with regimen r and select those which satisfy the multi-day pattern of regimen r and our preference criteria (described below). As a result, the appointments selected for actual patient p might correspond to appointments of several artificial patients, but they satisfy the requirements of patients p and potentially lead to a high quality running schedule.

We describe how to perform a feasibility test for selecting day d_p as the first visit day of actual patient p with regimen r. Let $\pi_r = (\pi_r(1), \pi_r(2), \ldots, \pi_r(z_r))$ be the multi-day pattern associated with regimen r, where $\pi_r(1) = 1$ and

- $\pi_r(j)$ - 1 is the number of days from the first visit to the j^{th} appointment;
- z_r is the total number of appointments of the multi-day pattern of regimen r.

The first visit date d_p of actual patient p should be selected in such a way that for each appointment j, $j = 1, 2, \ldots, z_r$, there exists a matching artificial appointment scheduled

in the template on date $d = d_p + \pi_r(j) - 1$. The days and starting times of the selected artificial appointments are then booked for patient p.

An artificial appointment of day d of the the template matches the j^{th} appointment of actual patient p if

(i) it is associated with regimen r,

(ii) it is free, i.e., it has not been used to book an appointment for another actual patient,

(iii) the clinic workload of day d, if increased by treatments activities needed for additional patient p, does not exceed a given threshold.

Verifying the first two conditions is straightforward. In what follows we clarify the last condition.

In order to keep the workload of the clinic within acceptable limits and to reserve some proportion of working hours for additional nurse duties, the relative workload of a clinic on any day must not exceed a given threshold $\sigma < 1$ called *capacity ratio*. The threshold σ represents the proportion of nurses' time which can be booked for treatment activities. Using the notation from Section 4.3.2 and the notion of the average relative workload $\hat{W}_d$, the corresponding constraint can be expressed as

$$\hat{W}_d \leq \sigma.$$

In accordance with definition (4.6) of the relative workload, the capacity requirement can be re-written as

$$\frac{W_d + w_r}{C_d} \leq \sigma,$$

where W_d is the overall time required for treating actual patients who have appointments on day d, C_d is the capacity of the clinic on day d measured as the total number of 15-minute time-slots when the nurses are available, and w_r is the total number of time-slots needed for treatment activities of patient p.

Summarising, a date $d_p \in [d_p^{\min}, d_p^{\max}]$ is a feasible date for the first appointment of patient p with regimen r if for each day $d_p + \pi_r(j) - 1$, $j = 1, 2, \ldots, z_r$, there exists a matching artificial appointment satisfying conditions (i), (ii) and (iii).

The minimisation of the number of clashing activities is fundamental in order to achieve a successful running schedule (see metric F3). For this reason our algorithm chooses a feasible starting date d_p such that the maximum density level k of the selected artificial appointments is minimum. By the definition of a density set, selecting appoint-

ments belonging to the set $\mathscr{L}_d^{(k)}$ results in an intra-day schedule with clash density not exceeding k.

The formal description of the algorithm is given by procedure 'Match-Appointments(p,σ)' presented below. It is assumed that the template schedule is represented by the set of artificial appointments grouped in density sets $\mathscr{L}_{r,d}^{(k)}$ for each day $d \in H$ and each regimen $r \in R$. The algorithm uses the 'Feasibility-Test(r, j, d, k, σ)' which verifies whether for the jth visit day of the patient with regimen r there exists a matching artificial appointment of set $\mathscr{L}_{r,d}^{(k)}$ satisfying conditions (i), (ii) and (iii).

Procedure 'Match-Appointments(p,σ)'

$k := 0$;

WHILE appointments $\pi_r(1), \pi_r(2), \ldots, \pi_r(z_r)$ for patient p are not booked DO

Set the density level $k := k+1$;

FOR $d_p = d_p^{\min}$ TO $d_p^{\max}$ DO

IF 'Feasibility-Test(r,j,d,k,σ)' confirms for each $j = 1,2,\ldots,z_r$, that the j^{th} appointment of patient p can be assigned to the corresponding day $d = d_p + \pi_r(j) - 1$ and the matching artificial appointments belong to density set $\mathscr{L}_{r,d}^{(k)}$;

THEN book all appointments for patient p with the first visit day d_p; STOP

END FOR

END WHILE

Notice that in some extreme cases a set of feasible dates and times may not be found by procedure 'Match-Appointments(p,σ)'. For example, if patient arrival rates substantially diverge from the forecast of Stage 1, artificial appointments for arriving patients may not be available in the template schedule. Our experiments show that this difficulty can be overcome by using an appropriate overestimation rate of patient arrival at the stage of template generation.

4.5 Daily Rescheduling

In this section we develop an integer linear program to adjust a one-day schedule in order to achieve an improvement in the running schedule in terms of metrics F2, F3 introduced in Section 4.2. In particular, we reduce the number of clashing activities and the maximum clash density of each day by full re-allocation of nurses, introducing minor shifts of nurses' meal breaks and minor delays in starting times of the pre-booked appointments. Observe that changing nurse allocation does not incur any cost as a patient can be treated by different nurses on different visit days.

Rescheduling for day d can be performed when complete information about all booked appointments for that day is known. The integer linear program presented below is a reformulation of $ILP_d^{(k)}$ introduced in Section 4.3 with slightly modified objective functions and constraints.

Consider a set of patients P_d visiting the clinic on day d. For each patient $p \in P_d$, let s_p be the starting time of the pre-booked appointment and $n_p \in N_d$ be the nurse allocated

to patient p on that day. The rescheduling problem consists in finding for each patient p a new starting time s'_p and a new nurse allocation $n'_p \in N_d$ such that the maximum clash density

$$\Delta_d = \max_{n \in N} \Delta_{d,n}$$

and the total number of clashing activities

$$\Omega_d = \sum_{n \in N} \Omega_{d,n}$$

are minimised (see Section 4.1). Since only small delays in patients' starting times are acceptable, a new starting time s'_p of the appointment of patient p can take values from a restricted set $H'_{p,d} \subseteq H_d$, where H_d is the set of 15-minute time intervals of day d. For example, if it is acceptable to delay starting times by at most two 15-minute time slots, then

$$H'_{p,d} = \{t \in H_d \mid s_p \leq t \leq s_p + 2\}.$$

Consider now reallocation of patients to nurses. Introduce a set of patients $P_{s,i,n,d}$ such that the intra-day pattern of patient $p \in P_{s,i,n,d}$ incurs an activity for nurse $n \in N_d$ in time slot i, if the appointment starts at time s. Allocation of patient p with appointment starting time s'_p to nurse n'_p may be infeasible if it is not possible to complete all treatments of the intra-day pattern of patient p within the working hours of nurse n'_p. Therefore we limit our consideration to a set of nurses $N_{p,s,d} \subseteq N_d$ whose working hours on day d allow to perform all treatments of patient p with starting time s.

Similar to formulation $\mathrm{ILP}_d^{(k)}$, we define the decision variables $x_{p,s,n,d}$ and $b_{t,n,d}$:

$$x_{p,s,n,d} = \begin{cases} 1, & \text{if the appointment time of patient } p \text{ is } s \text{ and the allocated nurse is } n, \\ 0, & \text{otherwise.} \end{cases}$$

$$b_{t,n,d} = \begin{cases} 1, & \text{if nurse } n \text{ has a meal break starting at time } t, \\ 0, & \text{otherwise,} \end{cases}$$

and auxiliary variables $y_{i,n,d}$ to measure the number of clashing activities. Recall that

$$y_{i,n,d} \geq 1,$$

where $y_{i,n,d} = 1$ represents either of the cases: the nurse is free in time slot i or performs one activity in time slot i and there are no other clashing activities, see Section 4.3.3. The case $y_{i,n,d} > 1$ corresponds to the number of clashing activities of nurse n happening in

time slot i and it is calculated as

$$y_{i,n,d} \geq \sum_{s \in H_{n,d}} \sum_{p \in P_{s,i,n,d}} x_{p,s,n,d}.$$

The two main objective functions of the rescheduling problem are the maximum clash density Δ_d and the number of clashing activities Ω_d:

$$\Delta_d = \max_{n \in N_d,\ i \in H_{n,d}} \{y_{i,n,d}\},$$

$$\Omega_d = \sum_{n \in N_d} \sum_{i \in H_{n,d}} (y_{i,n,d} - 1).$$

An additional objective is aimed at balancing the workload of different nurses and it is measured as the maximum difference W_d^{diff} in nurses' workloads:

$$W_d^{\text{diff}} = \max_{n_1, n_2 \in N_d} \{|\hat{W}_{n_1,d} - \hat{W}_{n_2,d}|\}.$$

Recall that the relative workload $W_{n,d}$ of nurse n on day d is defined by (4.5) and it can be calculated as

$$\hat{W}_{n,d} = \frac{1}{|H_{n,d}|} \sum_{s \in H_{n,d}} \sum_{i \in H_{n,d}} \sum_{p \in P_{s,i,n,d}} x_{p,s,n,d}.$$

Since the above three criteria are conflicting, we establish an order of their importance for lexicographical optimization. Our first priority is to minimise the maximum clash density Δ_d; secondly, among the solutions with the smallest clash density, we give priority to those with the smallest number of clashing activities Ω_d; finally, among the solutions with the smallest Δ_d and Ω_d we select those with the minimum workload difference W_d^{diff}.

Summarising, the resulting integer linear program ILP_d is of the form:

$$ILP_d: \quad \text{lex}\left[\min \Delta_d,\ \min \Omega_d,\ \min W_d^{\text{diff}}\right]$$

$$\begin{aligned}
\text{s.t.} \quad & \Delta_d \geq y_{i,n,d}, && n \in N_d, && i \in H_{n,d}, \\
& \Omega_d = \sum_{n \in N_d} \sum_{i \in H_{n,d}} \left(y_{i,n,d} - 1\right), \\
& W_d^{\text{diff}} \geq \hat{W}_{n_1,d} - \hat{W}_{n_2,d}, && n_1, n_2 \in N_d, \\
& \hat{W}_{n,d} = \frac{1}{|H_{n,d}|} \sum_{s \in H_{n,d}} \sum_{i \in H_{n,d}} \sum_{p \in P_{s,i,n,d}} x_{p,s,n,d}, && n \in N_d, \\
& y_{i,n,d} \geq \sum_{s \in H_{n,d}} \sum_{p \in P_{s,i,n,d}} x_{p,s,n,d}, && n \in N_d, && i \in H_{n,d}, \\
& y_{i,n,d} \geq 1, && n \in N_d, && i \in H_{n,d}, \\
& \sum_{s \in H'_{p,d}} \sum_{n \in N_{p,s,d}} x_{p,s,n,d} = 1, && p \in P_d, \\
& \sum_{p \in P_d} \sum_{s \in B_{p,t,n,d}} x_{p,s,n,d} + b_{t,n,d} \leq 1, && n \in N_d, && t \in M_{n,d}, \\
& \textstyle\sum_{t \in M_{n,d}} b_{t,n,d} = 1, && n \in N_d \\
& x_{p,s,n,d} \in \{0,1\}, && p \in P_d, && s \in H'_{p,d}, && n \in N_{p,s,d}, \\
& b_{t,n,d} \in \{0,1\}, && t \in M_{n,d} && n \in N_{p,s,d}.
\end{aligned}$$

Notice that the pre-assigned appointment starting time s_p and nurse allocation n_p for every patient $p \in P_d$ define a feasible solution to ILP_d, which can be used as an initial solution for that problem. An optimal solution to ILP_d determines new appointment times s'_p and nurse allocations n'_p for all patients $p \in P_d$.

Such problem needs to be solved in the forth stage of our solution approach (see Section 4.5)

4.5.1 Nurse assignment problem

The nurse assignment problem consists of assigning a set N_d of k nurses to a set P_d of patients whose appointment times have already been fixed for the day d. Each appointment has to be allocated to one of the nurses such that no clashing activities occur in nurses' diaries, i.e. no more than one appointment should be allocated to any time slot of the nurse diary. Notice that, since all appointments and nurses involved in the assignment are known in advance, the problem considered is an offline problem.

In this subsection we study the complexity of the nurse assignment problem which appears in the fourth stage of our solution approach (see Section 4.5). In particular, we show that such a problem is NP-complete and it can be solved by determining a colouring of a special designed graph.

Consider a graph $G = (V,E)$ and a number $k > 2$ of colours where V is the set of vertices and E is the set of undirected edges connecting the vertices. *The k-colouring* problem consists in determining the existence of an assignment of the k colours to vertices such that, for each arc, adjacent vertices have different colours. Such an assignment is called a *k-colouring* of the graph. The k-colouring problem has been proved to be NP-complete in Garey and Johnson [38].

Given an instance of the nurse assignment problem, we define a so-called *conflict graph* $G = (V,E)$ with the node set V and the arc set E as follows. Each node $v \in V$ is associated to a patient appointment to be assigned to a nurse. Given two nodes $u, v \in V$, there exists an undirected arc $(u,v) \in E$ if and only if the patient appointments associated to nodes u and v generate one or more clashes when assigned to the same nurse.

The nurse assignment problem can be modelled as the problem of determining a k-colouring of the corresponding conflicting graph where each of the k colours represents one of the k nurses. In particular, a patient appointment is assigned to a nurse if the node in the conflicting graph associated to the patient appointment is coloured with the colour representing the nurse. Clearly, the solution of the nurse assignment problem can be achieved by two steps: constructing the conflict graph of the given instance and then determining a feasible k-colouring using available algorithms.

In what follows we show that the nurse assignment problem is NP-complete if the number of nurses $|N_d|$ is larger than 2 providing a polynomial reduction from the graph k-colouring problem which is known to be NP-complete. In particular, given an arbitrary instance of the k-colouring problem, we construct an instance of the nurse assignment problem. Then we prove that the nurse assignment problem has a clash-free assignment with no more than k nurses if and only if the k-colouring has a solution with no more than k colours.

To prove the NP-completeness, we derive a polynomial-time algorithm 'Reduction from k-colouring' reducing any instance of the k-colouring problem to an instance of the decision version of the nurse assignment problem. The reduction is performed by the polynomial-time algorithm 'Reduction from k-colouring' which, given an arbitrary undirected graph $G = (V,E)$ and a number of colours k, generates a set N_d of k nurses, a set P_d of patients, their appointment intra-day patterns and starting times (see the formal description below).

The idea behind the reduction algorithm is to create a set of clashing activities for adjacent nodes in the conflict graph. The algorithm initially generates for each node $v \in V$ an empty patient appointment P_v with no activities scheduled, then it adds activities to appointments while iterating the nodes of the graph. The algorithm stops when all nodes

have been visited once. When a node $v \in V$ is visited, the algorithm schedule, starting from the time slot s, as many contiguous unit-length activities as the number of unvisited nodes adjacent to v. Then, for each unvisited node v_j adjacent to v, a unit-time activity is scheduled at time $s+t$ to the patient's appointment P_{v_j}, where t represents the number of neighbours considered. As a result, the contiguous activities constructed for v clash with the unit-time activities generated for nodes adjacent to v. It is worth clarifying that, when we start exploring a new node v, the corresponding patient appointment may already have one or more unit-length activities assigned to time-slots earlier than the current time s.

Algorithm 'Reduction from k-colouring$(G(V,E),k)$'
Set a variable $s := 0$;
Generate a set N_d with k nurses;
Label all nodes in V as 'unmarked' and renumber them arbitrarily;
Create an empty appointment of patient P_v for each vertex $v \in V$;
Consider each unmarked node v according to the numbering given:
    Label the current node v as 'marked';
    Schedule an activity of the appointment for patient P_v at time s with duration equal to the number of unmarked
    neighbours of v, if any;
    Set $t := 0$;
    FOR each unmarked node v_j adjacent to v DO
        create a clashing unit activity at time $s+t$ for appointment of patient P_{v_j};
        set $t := t+1$;
    END FOR
    Set $s := s+t+1$;
Generate the patterns according to the activities assigned.

Example In what follows we give an example of instance of the nurse assignment problem generated by the algorithm 'Reduction from k-colouring$(G(V,E),k)$'.

Consider a number of colours $k = 3$ and the undirected graph G with vertices $V = \{1,2,3,4,5,6\}$ and edges $E = \{(1,2),(1,4),(2,4),(3,5),(3,6),(4,5),(4,6),(5,6)\}$ represented in Figure 4.5. Let us call the three colours A, B and C. The given instance admits the feasible colouring $\mathscr{C} = \{(1,A),(2,B),(3,C),(4,C),(5,A),(6,B)\}$. Using the nodes numbering given in the Figure 4.5, the algorithm 'Reduction from k-colouring' produces the appointments showed in Figure 4.6 and the intra-day patterns listed in Table 4.4.

Notice that also the nurse assignment problem admit a feasible solution. Naming the a

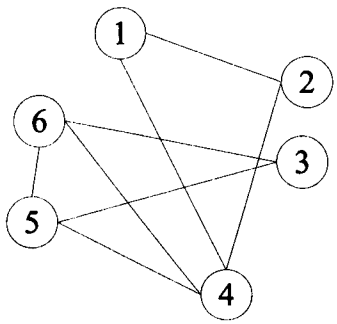

Figure 4.5: Example of undirected graph.

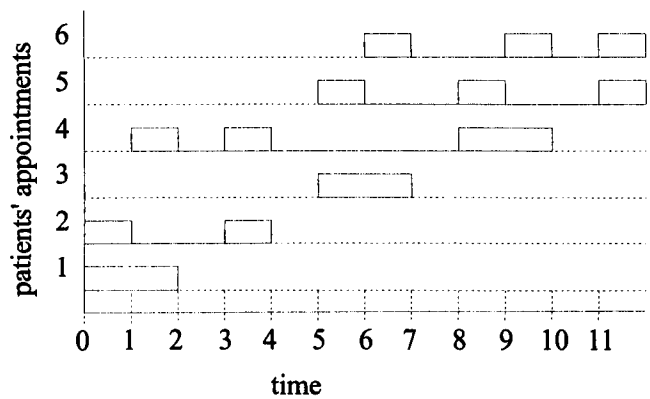

Figure 4.6: Appointment schedule generated by algorithm 'Reduction from k-colouring' from graph in Figure 4.5; the time is expressed in 15-min time-slots

Pattern No.	15-mins timeslots
1	1,2
2	1,4
3	1,2
4	1,3,8,9
5	1,4,7
6	1,4,6

Table 4.4: Intra-day patterns generated by algorithm 'Reduction from k-colouring' from the graph 4.5; 15-min timeslots are counted starting from 1.

set of nurses with colours name A,B and C, a feasible assignment of patients appointments to nurses is:

- appointments 1,5 are assigned to nurse A;

- appointments 2,6 are assigned to nurse B;
- appointment 3,4 are assigned to nurse C.

■

It is easy to see that the number of steps of the algorithm 'Reduction from k-colouring' for the worst case instance is $O(|E|)$ where $|E|$ is the number of arcs in the conflicting graph.

Theorem 9. *The nurse assignment problem is NP-complete.*

Proof: To prove NP-completeness, we show that (1) a k-colouring of the conflict graph corresponds to a clash-free nurse assignment and (2) a clash-free assignment of patients appointments to nurses corresponds to feasible k-colouring of the conflict graph.

The validity of the first statement is a direct consequence of the definition of the conflicting graph. Consider a feasible k-colouring of the conflicting graph. A solution of the nurse assignment problem is constructed assigning appointments associated to nodes with the same colour, to the nurse represented by such a colour. Clearly, since there is no arc connecting nodes with same colour, then no clash appear between patients appointments associated to the same nurse.

We prove the correctness of second statement by contradiction. Suppose that there exists a feasible nurse assignment that leads to an infeasible k-colouring for the reduced graph $G = (V,E)$. Then, there exists two adjacent vertices $u,v \in V$ which have the same colours. Since each colour is associated to a nurse and each node represents a patient appointment, the patients appointments associated to nodes u,v are served by the same nurse. This contradicts the fact that the assignment is feasible since the algorithm 'Reduction from k-colouring' generates two clashing activities between appointments associated to adjacent nodes. ■

Special cases of the nurse assignment problem solvable in polynomial time can be derived by conflict graphs with special structures. An example is given by interval graphs which model the nurse assignment problems where each intra-day pattern is a contiguous set of activities. Since coloring of the interval graph is solvable in polynomial time(see [52]), then the nurse assignment problem with intra-day patterns made of contiguous activities is solvable in polynomial time. Another example is case in which each intra-day pattern is made of a single activity. It is easy to see that in this case the resulting conflict graph is made of a collection of unconnected cliques which can be easily coloured (see [52]). The special case in which only 3 time intervals are defined is also polynomially solvable despite the total number of nurses. In fact, the conflict graphs arising from this

class of instances are chordal graphs for which there exists a polynomial-time algorithm to color them.

However the nurse assignment problem is still NP-complete for many other special cases. Some examples are the case in which each pattern has exactly 3 activities or the case in which each pattern has exactly 2 activities. The complexity of the nurse assignment problem for instances with the same intra-day patterns remains an open question.

4.6 Clinic Appointment Data

In what follows we describe typical problem instances as they appear in the historical appointment records of the chemotherapy outpatient clinic in the St. James's Hospital in Leeds for the period from 1st May 2008 to 1st September 2008. The data analysed has been extracted from the information system in use at the clinic and has been given to us by the management in an anonymised form so that no personal patient information could be identified. Since the data has been input in the system manually, it presents many inconsistencies and missing information that could not be repaired and completed. However, the analysis of such data gives us enough information to understand the number of patients, appointments and treatment involved and to design a realistic and accurate setting for the computational experiments in Section 4.7.

In the chemotherapy outpatient clinic of St. James's Hospital in Leeds an average of 800 appointments are scheduled every month. Treatments are performed by 19 nurses which have different working hours. The clinic provides the outpatient service 5 days a week and every day about 40 patients are treated by 8 nurses. According to the recorded data, the time from the decision to treat to the date of the first appointment is 14 days on average. A schedule in a typical day has about 20 nurse clashing activities which becomes 450 for a typical month. Patients' waiting times on treatment days can be as large as 2 hours. The percentage of time a nurse spends on treatment activities in relation to overall nurse's working hours is 47% on average.
In what follows we provide a simple analysis of the data from two perspectives:

1. we ignore the links between appointments defined by the multi-day patterns for a specific treatment and we analyse the monthly and daily number of appointments;

2. we consider each set of appointments defined by the multi-day patterns for a specific treatment as a single event and we analyse their monthly and daily number.

The first perspective gives us a picture of the distribution of the clinic workload over

the time. Differently, the second perspective helps us to understand the patients' arrival process and waiting days and times.

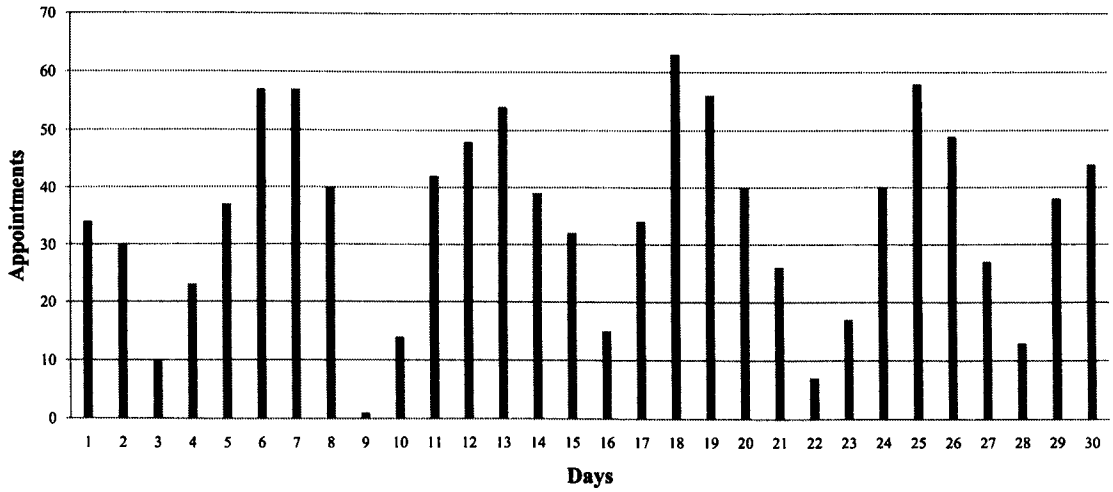

Figure 4.7: Graph representing the number of appointments of a typical month

Figure 4.7 shows a graph representing the number of appointments occurring each day in one month. It is easy to see from the figure that with the current scheduling policy the mid-week load of the clinic is higher. On Mondays and Fridays the clinic load is much lower due to the start-up of the pharmacy (Mondays) and short staff shifts (Fridays). This variation on the number of daily appointments is also confirmed by the high value of standard deviation which is 17.79 where the average number of appointments is 34.21.

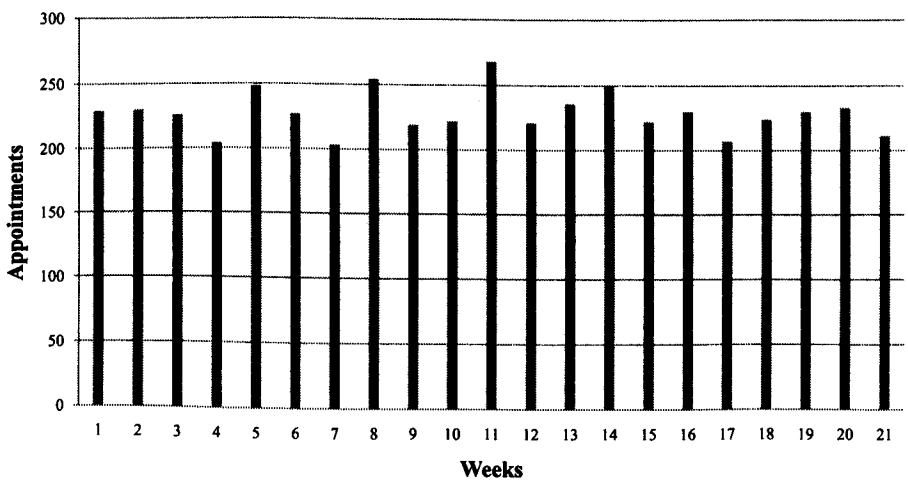

Figure 4.8: Graph representing the number of appointments for each week of the historical data

The number of appointments for each week is shown in Figure 4.8. It is possible to notice that the weekly variation of the number of appointments is relatively much smaller

than for the daily case. In fact, although the standard variation value of the number of appointments is 16.38, it is calculated over an expected number of appointments as large as 228.23.

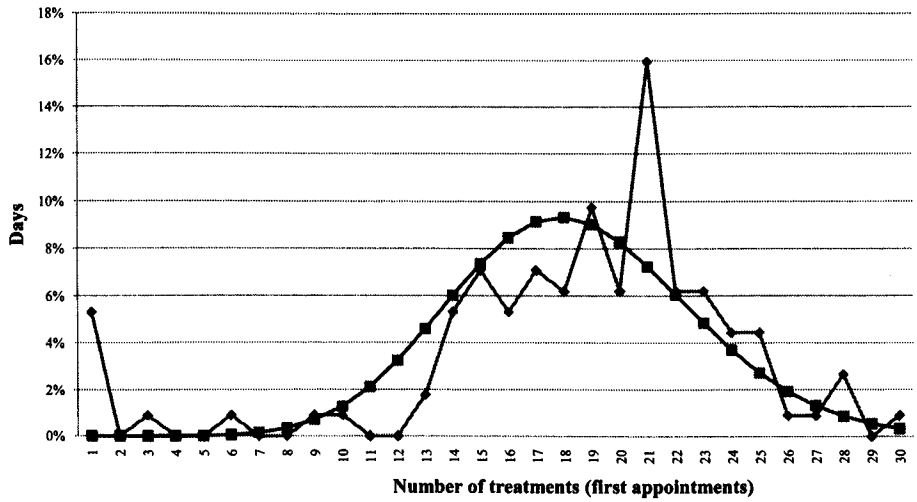

Figure 4.9: Graph representing daily arrival frequency

When considering the treatments as single events, we count the number of first appointments on daily and weekly basis to understand the arrival process. Notice that, although a patient may undergo to many chemotherapy treatments before the recovery, in our analysis we count each treatment as a separate entity ignoring the actual patient it is associated to.

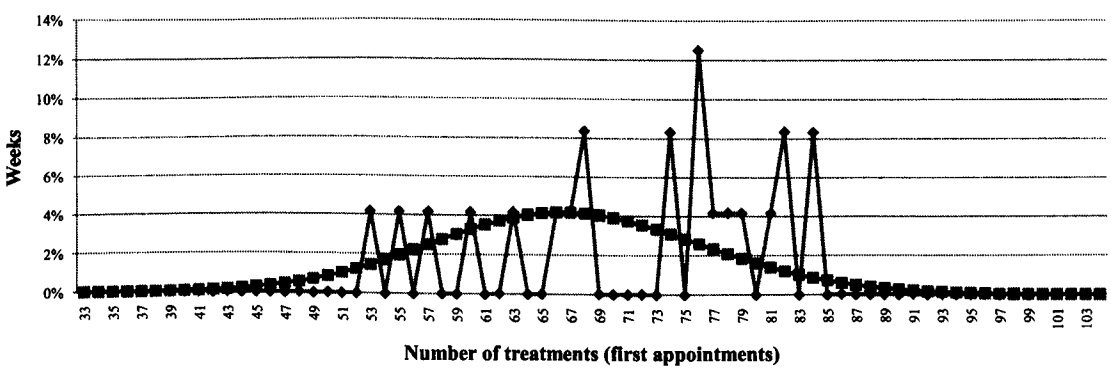

Figure 4.10: Graph representing weekly arrival frequency

Figures 4.9 and 4.10 show the number of the first appointments recorded on a daily and weekly basis expressed in term of their frequency. In particular, for each possible arrival quantity, the number of days (or weeks) in which such number of first appointments arrived is expressed as a percentage over the total number of days (weeks) in the data set.

The dotted lines with squares represent the best fit (using maximum likelihood method) of Poisson distribution to the data. The fitting procedure led to λ parameters equal to 18.37 and 91 for the daily and weekly case respectively. The standard deviations of the two data sets are 5.96 and 18.75 respectively.

In Figures 4.11 and 4.12 we represent the number of weekly and monthly arrivals where the treatments are grouped by the 160 most important regimens. Observe that in the 4-months data considered most of the arrivals are associated to 40 common regimens while other regimens have not been prescribed at all. Notice that, as shown in Section 4.3.3, the 4-stage approach take in consideration such situation ensuring that some time is reserved in the template schedule to easily match appointments for rare regimens.

4.7 Computational Experiments

In this section we describe the design of the experiments performed and their computational results. We perform two types of experiments evaluating the operations of the clinic when

- only the rescheduling procedure (Stage 4 of our approach) is used in addition to the existing manual scheduling policy,
- the whole four-stage scheduling approach is adopted.

The evaluation is based on historical data recorded at the St. James's University Hospital in Leeds during the period from 1st May 2008 to 1st September 2009.

In our first set of experiments we show how the rescheduling procedure corresponding to Stage 4 of our approach can be used to improve daily schedules produced in the hospital manually. We use actual daily schedules for one month of the recorded historical data (May 2008). The results are summarised in Table 4.5 where we compare the average number of daily clashes over the one month period depending on rescheduling constraints.

Our experiments demonstrate that operation of the clinic can be substantially improved if nurse reallocation is done (compare the figures in the first two columns of Table 4.5). Further improvement can be achieved if delays are allowed in starting times of patients' appointments (see the figures in the last four columns of Table 4.5). Observe that such delays do not affect all patients and they are not necessarily of maximum duration (see Table 4.6).

In the second set of experiments, we evaluate the four-stage approach on two scenarios, one with patients' arrival rates similar to actual rates in the real clinic (119 patients

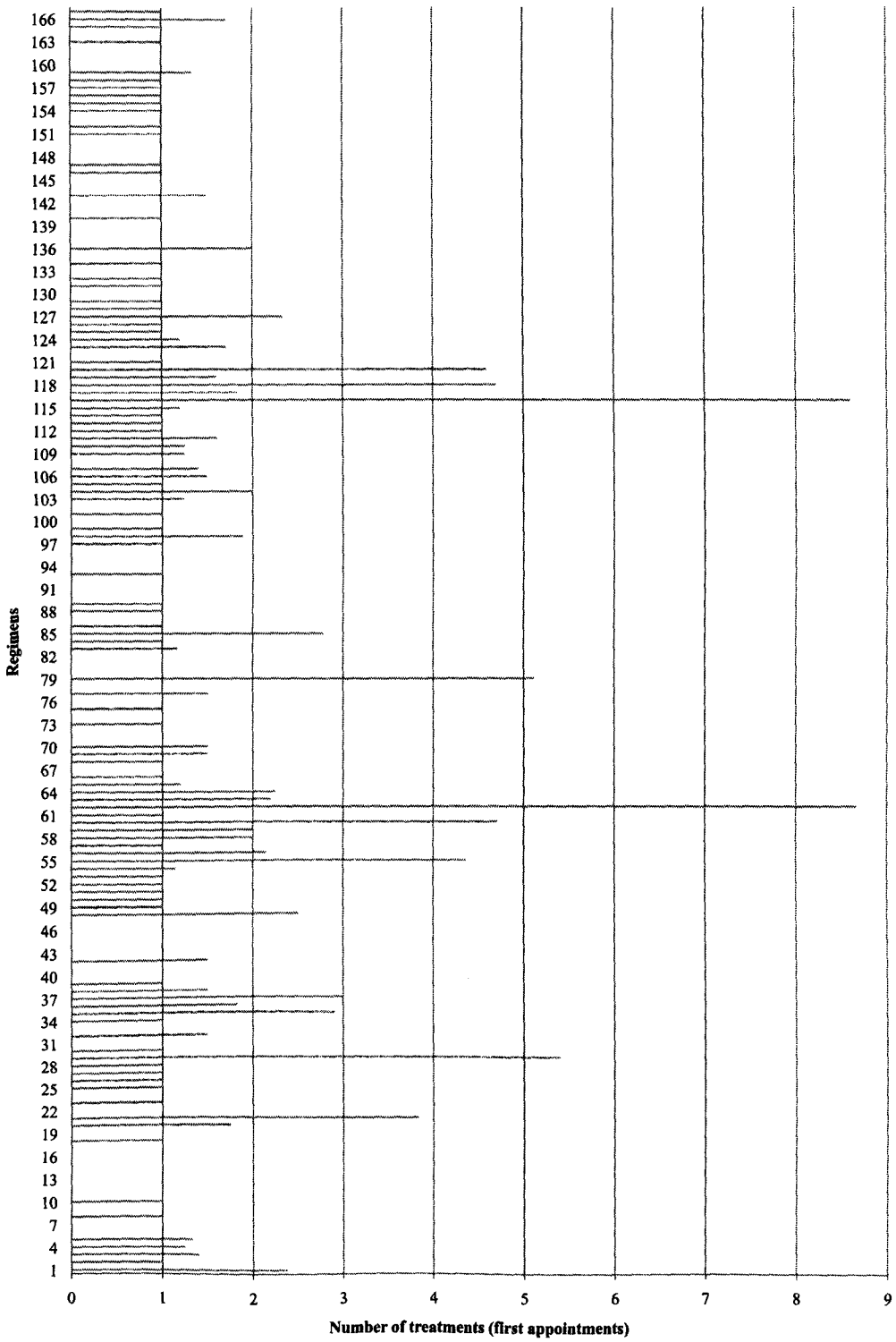

Figure 4.11: Graph representing the weekly arrival frequency grouped by regimens

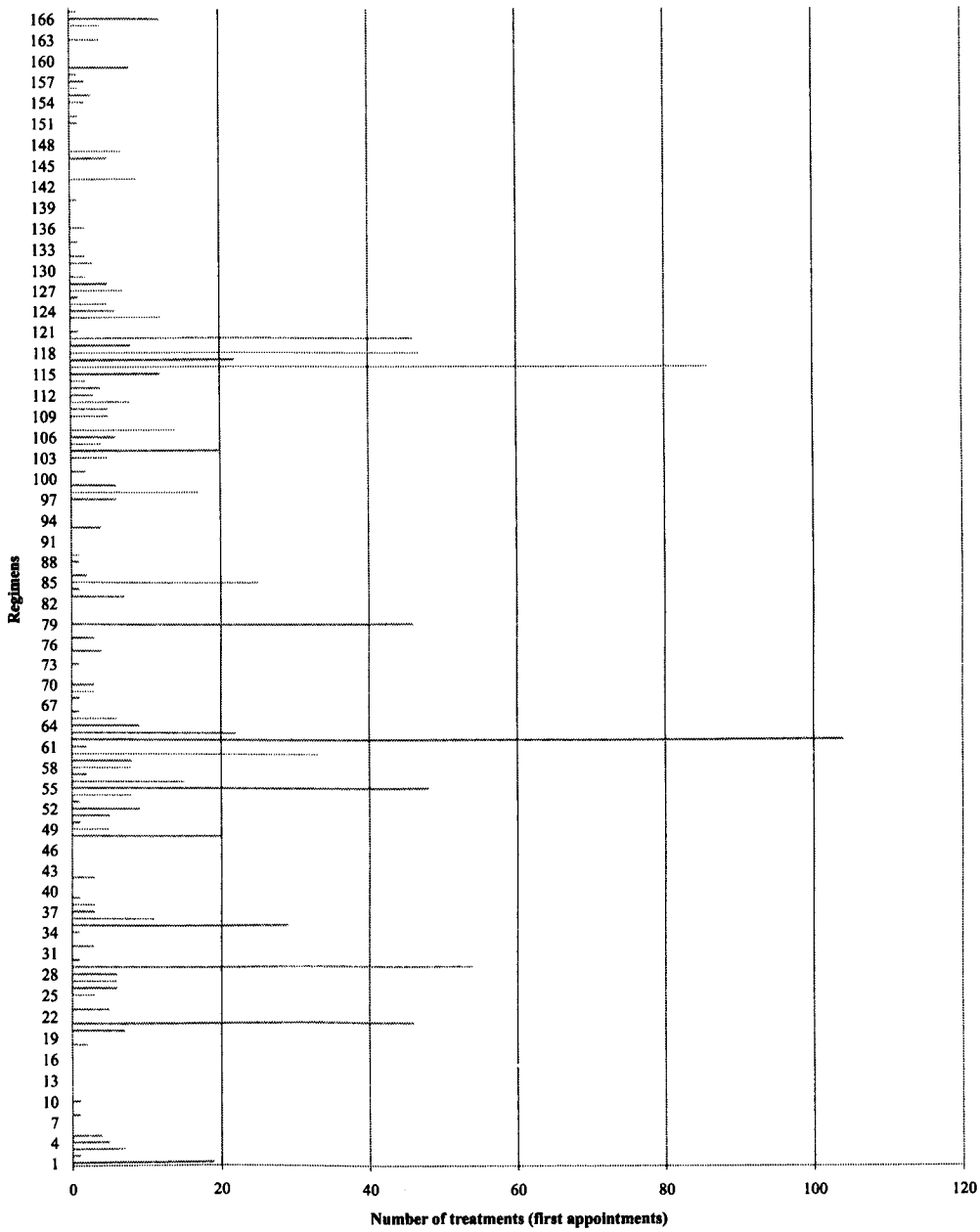

Figure 4.12: Graph representing the monthly arrival frequency grouped by regimens

per week on average) and another one with higher arrival rates (about 141 patients per week). We consider the 160 regimens which occur in the real clinic most often. Our experiments cover only one time horizon H consisting of 134 days with the first 60 days corresponding to the arrival period H_a, see Section 4.2.

Stages 1-2 are performed only once producing one multilevel template schedule, which is then used to generate running schedules for both scenarios. The data for ar-

	Actual schedule	Nurse reallocation; no delays in appointments starting times	Nurse reallocation and possible delays in appointment starting times			
			15 min delay	30 min delay	45 mis delay	complete rescheduling
Number of clashing activities per month	454	110	41	30	22	0
Max clash density	6	4	4	2	2	1
Daily avg. clash density	3.2	1.95	1.55	1.25	1.20	1

Table 4.5: Comparison of actual daily schedules of May 2008 used at the St. James's Hospital (Leeds, U.K.) and those obtained via rescheduling

	Nurse reallocation and possible delays in appointment starting times		
	15 min delay	30 min delay	45 min delay
patients not waiting	53.98 %	44.96 %	40.19 %
patients waiting 15 mins	46.02 %	22.94 %	18.81 %
patients waiting 30 mins	-	32.10 %	17.78 %
patients waiting 45 mins	-	-	23.22 %

Table 4.6: The effect of rescheduling on patient waiting times

tificial patients is generated considering the arrival rate λ_r calculated as the arithmetic mean of weekly arrival rates recorded in the historical data. The value of parameter ξ, which defines the level of overestimation, is set to 5.

Stages 3 and 4 are evaluated on 100 datasets, which describe patients' arrivals simulated using a Poisson process. Each dataset we generate contains arrival dates of patients for all regimens. For the first scenario, arrival rates are the same as λ_r-values used in Stage 2. For the second scenario the arrival rates are increased by about 20%.

All scheduling decisions for a patient have to be made on the arrival date. Depending on the category of a patient, the first appointment should happen within 7, 14 or 28 days from the arrival date. We denote the correspond groups of patients as A, B and C, assuming that the patients are split in proportion 73% , 15% and 12%, respectively.

Both scenarios use the same nurses' weekly rota which is shown in Table 4.7. The shifts have different lengths giving nurses some flexibility in negotiating preferred working hours. Notice that on Monday nurse shifts start no earlier than 11 a.m. since the hospital pharmacy needs additional set up time after a weekend in order to prepare drugs for treatments. It is assumed that each nurse needs a 30-minute lunch break between 11 a.m. and 2 p.m.

The capacity ratio σ used in Stage 3 to generate and maintain running schedules is set to 0.9 bounding the total number of treatment activities performed by a nurse. This implies that in addition to a 30-minute lunch break, nurse's schedule should contain at least 10% of unused time slots for additional duties.

	Mon	Tue	Wed	Thu	Fri
Shift 1	11:00-18:00	9:00-18:15	9:00-18:15	9:00-18:00	9:00-16:45
Shift 2	11:00-18:00	9:00-18:15	9:00-18:15	9:00-18:00	9:00-16:45
Shift 3	11:00-18:15	9:00-18:15	9:00-18:15	9:00-18:00	9:00-16:45
Shift 4	11:30-18:15	9:00-18:15	9:00-18:15	9:00-16:45	9:00-16:45
Shift 5	12:00-18:30	9:00-17:00	9:00-18:15	9:00-16:45	9:00-16:15
Shift 6	12:00-19:00	9:00-17:00	9:00-17:30	9:00-16:45	9:00-16:15
Shift 7	-	9:00-17:00	9:00-17:00	9:00-16:45	9:00-16:15
Shift 8	-	9:00-17:00	9:00-17:00	9:00-16:45	9:00-16:15
Shift 9	-	-	-	-	9:00-16:45

Table 4.7: Weekly nurse rota for scenario 1 and 2

	Average monthly number of	
	appointments	treatments
Real clinic	992	415
Scenario 1	1024	531
Scenario 2	1277	617

Table 4.8: Average number of appointments and treatments performed in a typical month

Tables 4.9 and 4.10 characterise the quality of the generated running schedules. In the first scenario, our approach is able to schedule appointments reducing the average number

	Daily Clashing activities			Daily Clash Density			Relative Workload (%)		
	Avg.	Dev.	Max.	Avg.	Dev.	Max.	Avg.	Dev.	Max.
Real clinic	22.7	10.43	43	3.20	0.80	6	47.28	12.71	62.90
Scenario 1	1.25	2.07	15	1.42	0.51	3	62.71	13.84	84.80
Scenario 2	1.95	2.55	18	1.56	0.52	3	71.13	12.53	85.86

Table 4.9: Characteristics of schedules produced for Scenario 1 and 2

	Waiting days							Waiting times (minutes)	
	Total	Group A		Group B		Group C			
	Avg	Avg.	Dev.	Avg.	Dev.	Avg.	Dev.	Avg.	Dev.
Real clinic	14	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Scenario 1	4.69	3.80	4.85	5.79	4.83	8.97	6.50	11:11	12:28
Scenario 2	5.26	4.42	5.46	6.28	5.26	9.30	6.73	10:59	12:25

Table 4.10: Patients' waiting days and times

of waiting days (see Table 4.10), eliminating almost all clashing activities (see Table 4.9) and, therefore, reducing their density and increasing nurses' relative workloads defined by (4.5).

The results of experiments for the second scenario are quite remarkable. With an increased number of arriving patients it is indeed inevitable that the waiting time characteristics and the number of clashes should increase. It appears that for a 20% increase in the number of patients (about 89 additional patients per month or 253 additional appointments) the deterioration in schedule quality is marginal: for each category of patients the number of waiting days increases on average by no more than 1 day, while the average number of clashing activities changes from 1.25 to 1.95 only. The comparison with the actual booking process of the clinic is even more dramatic: the quality of actual schedules generated manually for 992 appointments is much worse than the quality of schedules generated by our approach for 1277 appointments.

The experiments were performed on Intel Pentium Core 2 Quad CPU 2.5GHz and 3GB RAM. The ILP programs of Stages 2 and 4 were solved by a single thread version of CPLEX 11.2. Time estimates for the four stages of our approach are as follows.

- Stage 1 is implemented in the most simple way by calculating averages of patients'

arrival rates; the time required by these calculation is negligible.

- Since the template schedule is the major factor which affects the quality of running schedules, we allowed more time for solving the associated ILP programs: 1 hour for the solution of the multi-day problem and 8 hours for the solution of all intra-day problems of the time horizon. In the environment of a real clinic, long computation time of Stage 2 is acceptable as template schedules are generated rarely and serve for sufficiently long time. These calculations can be performed, for example, overnight.

- Stage 3 is a fast heuristic requiring less than 1 second to book all appointments for a patient.

- Stage 4 is implemented as a single ILP problem for each day. We set up a time limit of 120 seconds for rescheduling each intra-day schedule; however optimal solutions are often found within 60 seconds.

In order to set up the time limit for the series of intra-day problems of Stage 2, we use the following approach. The 8-hour time limit (28800 seconds) is divided among the days of the time horizon H in proportion to the number of daily appointments assigned. This implies that the time limit T_d for solving the intra-day problem for day d can be defined as

$$T_d = 28800 \times \frac{|P_d^{\mathscr{A}}|}{\sum_{d \in H} |P_d^{\mathscr{A}}|},$$

where $|P_d^{\mathscr{A}}|$ is the number of artificial patients to be treated on day d.

Generating an intra-day schedule for one day of the template involves a number integer linear programs $ILP_d^{(k)}$, one for each clash density level $k = 1, 2, \ldots, K$. Although the total number of density levels K in an intra-day schedule for day d can be as large as the number of artificial patients $|P_d^{\mathscr{A}}|$ scheduled on that day, for our datasets K can be bounded by a much smaller number defined empirically:

$$K \leq 5.$$

Due to this we set up a time limit for each program $ILP_d^{(k)}$ as

$$T_d^{(k)} = \frac{T_d}{5}.$$

Finally, since there are three objective functions for each problem $ILP_d^{(k)}$ optimized lexicographically, a time limit of $T_d^{(k)}/3$ is imposed for optimizing one function.

Summarising we observe that the computational experiments demonstrate the advantages of the proposed approach evaluated against the performance metrics of the model. Further experiments run in a real clinic in parallel with the existing manual system might provide additional evidence of operational benefits which the streamlined scheduling procedures can bring.

4.8 Discussion

In this chapter we have introduced a new appointment scheduling problem which arises in the context of a chemotherapy outpatient clinic. If all information about patients is known in advance, producing good schedules for nurses and patients which treatments should follow a combination of multi-day and intra-day patterns with constraints on waiting times appears to be a difficult activity. The on-line nature of the problem with patients arriving over time adds even more complexity to the scheduling process.

The approach we propose consists of four stages: forecasting, generating a template schedule, producing a running schedule and rescheduling. The major underlying idea is essentially based on the concept of multilevel template schedule which represents a well thought through plan. The template schedule contains more pre-booked appointments than anticipated, providing flexibility in selecting the most appropriate options for arriving patients and for handling unexpected arrivals. It is obtained as a solution to a series of integer linear programs with multiple objectives optimized lexicographically. Due to this, the template ensures that the final running schedule is potentially of high quality and satisfies the requirements of patients and nurses.

In order to use the template for booking actual appointments and creating a running schedule, we design a matching procedure which takes into account characteristics of appointments of the template and requirements of arriving patients. Finally, a running schedule is further improved via rescheduling.

Thus, the novelty of our work lies in

- the introduction of a new scheduling model with jobs consisting of repetitive activities satisfying given multidimensional patterns;
- the formulation of the concept of multi-level template for scheduling patients in an uncertain environment and in its development into a formal optimization model with advanced features;
- the integration of daily rescheduling procedures into long-term planning.

The proposed approach can be enhanced with various additional features. We have demonstrated how it can be extended to take into account the requirements of meal breaks for nurses. Further enhancements may include patients' preferences on appointment starting times (morning, midday or afternoon) or requirements of pharmaceutical suppliers and hospital pharmacy on drug preparation.

Computational experiments demonstrate that our approach can potentially bring substantial improvement in operation of a real clinic in different ways:

- maintaining patients' waiting times within required limits;
- improving nurses' schedules by reducing the number of clashing activities (from an average of 20 clashing activities per day to less than 2 clashes),
- increasing the clinic capacity in terms of additional patients (in our experiments based on real-world data, up to 89 patients can be treated monthly in addition to the current 476 patients on average) without extending nurses' working hours and avoiding essential deterioration in schedule quality.

We strongly believe that the concept of template schedule can be used as a powerful algorithmic tool to tackle complex online scheduling problems, especially those which involve multi-operation jobs with given patterns.

It will be interesting to consider alternative approaches for generating template schedules, for example, an integrated approach for solving multi-day and intra-day problems simultaneously rather than sequentially. Due to the size and complexity of the integrated problem, it will be appropriate to develop meta-heuristics and to compare the quality of the resulting schedules with those produced by the current ILP-based approach which treats multi-day and intra-day problems sequentially. Another possible improvement could be achieved in Stage 1 by developing advanced models for prediction of future demands.

4.9 Notation

Table 4.11 summarises the main notation used throughout Chapter 4. Symbols are indicated in their most general form: when indices are missing in the text, a summation over the elements of the domain is assumed unless differently stated, i.e. $W_d = \sum_{n \in N} W_{d,n}$.

Symbol	Description	Section
H	scheduling horizon	4.2

Symbol	Description	Section
P	set of arriving patients	4.2
$\Delta_{d,n}$	maximum clash density in a schedule of a nurse n on day d.	4.2
$\Omega_{d,n}$	number of clashing activities in a schedule of nurse n on day d.	4.2
Δ	maximum clash density over all days within the considered horizon, i.e. $\Delta = \max_{d\in H, n\in N} \Delta_{d,n}$.	4.2
Ω	total number of clashes for all days within the considered horizon, i.e. $\Omega = \sum_{d\in H, n\in N} \Omega_{d,n}$.	4.2
H_a	*arrival period*: initial part of the scheduling horizon in which patient arrives.	4.3
R	set of chemotherapy regimens. Each regimen is associated to an intra-day and multi-day pattern.	4.3
λ_r	expected number of arriving patients for a regimen r.	4.3
$P_r^{\mathscr{A}}$	set of artificial patients associated to regimen r generated for the template schedule.	4.3
$\mathscr{L}_{r,d,n}^{(i)}$	set of appointments of regimen r scheduled in density level i for nurse n on day d.	4.3
K	constant defining the maximum density clash for a multi-level template schedule.	4.3
N_d	set of nurses available on day d.	4.3
ξ	constant representing the overestimation rate for the generation of a multi-level template schedule.	4.3.1
t_p	target day of artificial appointment p.	4.3.1
$H_{n,d}$	set of working time-slots of nurse n on day d.	4.4
d_p	day scheduled for the first appointment of a patient p.	4.3.2
C_d	capacity of the clinic on day d in terms of available nurse time-slots.	4.3.2
$W_{n,d}$	workload of nurse n on day d measured as the number of 15-minutes activities performed.	4.3.2
$\hat{W}_d$	average relative workload measured as $\hat{W}_d = \frac{W_d}{C_d}$	4.3.2
U	maximum daily workload excess in the template schedule, i.e. $U = \max_{d\in H}\{\max\{W_d - C_d, 0\}\}$.	4.3.2
V	average deviation between expected and actual appointment starting times, i.e. $V = \frac{1}{\lvert P^{\mathscr{A}}\rvert} \sum_{p\in P^{\mathscr{A}}} \lvert d_p - t_p\rvert$.	4.3.2

Symbol	Description	Section
$X_d^{(k)}$	proportion of artificial appointments of different regimens allocated to density level $\mathscr{L}_{d,n}^{(k)}$.	4.3.2
$Y_d^{(k)}$	weighed sum of artificial appointments of different regimens allocated to density level $\mathscr{L}_{d,n}^{(k)}$.	4.3.2
$Z_d^{(k)}$	overall number of clashing activities determined by appointments in density level $\mathscr{L}_{d,n}^{(k)}$.	4.3.2
X_d	group of measures representing the proportion of artificial appointments assigned to density levels (see $X_d^{(k)}$)	4.3.2
Y_d	group of measures representing the weighted number of artificial appointments assigned to density levels (see $Y_d^{(k)}$)	4.3.2
Z_d	group of measures representing the number of clashing activities in density levels (see $Z_d^{(k)}$)	4.3.2
$\alpha_{r,d}$	weight defined for a regimen r on day d used in the calculation of $Y_d^{(k)}$.	4.3.2
$w_{p,s,d}$	number of activities that nurse n have to perform on day d to treat the patient p.	4.3.3
$\mathscr{U}_d$	set of unscheduled patients for day d.	4.3.3
$\mathscr{S}_d$	set of scheduled patients for day d.	4.3.3
$\tau_{p,n,d}$	latest time slot when the appointment of patient p can start to be treated by nurse n on day d.	4.3.3
$M_{n,d}$	set of possible starting times for the meal break of the nurse n on day d.	4.3.3
$B_{p,t,n,d}$	starting times of an appointment for patient p which determine a treatment activity to be performed during a meal break starting at time t.	4.3.3
$d_p^{\min}, d_p^{\max}$	minimum and maximum waiting days for the patient p.	4.4
$\pi_r(j)$	$\pi_r(j)-1$ represents the number of days from the first visit to the j^{th} appointment described in the regimen r.	4.4
z_r	total number of appointments of the multi-day pattern of regimen r.	4.4
σ	capacity ratio, i.e. threshold for maximum relative workload of the clinic.	4.4
w_r	total number of activities for the treatment of a patient associated to regimen r in one day	4.4

Symbol	Description	Section
P_d	set of patients visiting the clinic on day d .	4.5
n_p	nurse allocated to patient p.	4.5
s_p	starting time of appointment for patient p	4.5
$P_{i,s,n,d}$	set of patients whose intra-day pattern incurs in an activity for nurse n at time i is starting at time s on day d.	4.5

Table 4.11: Main notation used throughout Chapter 4.

Chapter 5

Conclusions

Theoretical models are of great importance for the solution of practical scheduling problems. This thesis studied combinatorial optimization aspects of scheduling problems with release/due dates and time-lags constraints inspired by two real-world problems: the transportation of goods by limited capacity vehicles and the booking of treatment appointments for patients of a chemotherapy clinic.

The first problem involves the scheduling of goods delivery between two sites by a number of limited capacity vehicles where the transportation time is fixed to a given value for all vehicles. Since goods are available at different release times and have to be delivered before given due dates, the objective is to find a schedule that minimises the maximum lateness of the deliveries. In the second problem, patients undergoing chemotherapy treatments in an outpatient clinic have to be scheduled for appointments in which a mixture of drugs are administered. All appointments which are part of a complete treatment of a patient have to be issued at once and must follow strict time patterns. In particular the number of days interleaving two appointments is regulated by the type of chemotherapy treatment prescribed. Moreover, each appointment involves a set of tasks to be performed by a nurse during the treatment. The number of tasks to be performed and their distribution over time also depends on the type of chemotherapy treatment and the drugs prescribed. The problem consists of determining the allocation of patient to nurses and in scheduling appointment dates and times such that patient waiting days respect given targets and no clashes occur between nurse tasks.

In Chapter 2 we modelled the transportation problem as a p-batching scheduling prob-

lem with equal processing times, release/due dates and with the maximum lateness objective function. After introducing the problem formally, we provided two polynomial-time algorithms: an algorithm for the single machine environment which outperforms those known in literature for 20 years and an algorithm for the parallel machine environment which represents the first algorithm for the problem. In particular, the algorithms presented improve the best running time for the single machine environment from $O(n^8 \log n)$ in [10] to $O(n^2 \log n)$ and achieve a running time of $O(n^3 \log n)$ for the parallel machine environment. Our algorithm design showed how the forbidden regions and barrier techniques, developed previously for the single non-batching machine environment, can be extended to the parallel batching machine environment. Lastly, we demonstrated the NP-hardness of the problem with finish-start precedence constraints and we provided a polynomial-time algorithm for the case with start-start precedence constraints.

Our work on p-batching machines shows that the powerful forbidden region and barrier techniques known for 20 years and probably overlooked by researchers can actually improve modern scheduling problems with equal processing times and release/due dates. Thus, we believe that one possible research direction is the study of the running time improvements that such techniques could bring to other scheduling problems with equal processing time.

A question that requires addressing is the application of forbidden regions to the p-batch scheduling problem in the parallel machine environment. We believe that the same achievements in terms of running times obtained in [94] where the forbidden regions technique have been extended from the single to the parallel machine environment for classical machines, can be obtained also for the p-batching case.

Another possible research direction is the application of the techniques in complex multi-stage machine environments involving the coordination between p-batching and non-batching machines with equal processing times. Such algorithms would have wide applications in scheduling supply chains and in assembly and production-transportation networks. Currently the team led by Prof. Sigrid Knust at the University of Osnabrück, Germany, is using the algorithms and source codes developed for this thesis in their research on the integrated scheduling of production and transportation stages.

In Chapter 3 a simplified version of the chemotherapy appointment scheduling problem was modelled as a coupled-operations scheduling problem with fixed time-lags and a makespan minimisation objective function. The aim was to develop techniques that could result in efficient solution algorithms and that could be extended to more general problems of appointment scheduling. Firstly we considered a compact representation of solution schedules. We formally proved that the coupled-operation scheduling problem

with a given permutation of first operations and fixed time-lags is NP-hard in the strong sense, even with unit processing times. This ruled out the possibility of a compact representation of feasible solutions based on the sequence of job first operations. Secondly we searched for an efficient neighbourhood to implement a local search algorithm for the general version of the coupled-operation scheduling problem with fixed time-lags such that no unfeasible solutions would be considered during the search. As a result we formulated the problem of generating a neighbour solution as a job reinsertion problem. Using the disjunctive graph representation, we demonstrated that the short cycle property used in [43] for job-shop environment can be adjusted to be applied for our reinsertion problem. Based on this observation we adapted an algorithm from [43] to the solution of the reinsertion problem. Finally we designed an efficient tabu search algorithm based on the neighbourhood introduced and we evaluated its performances through extensive computational experiments. Results presented in the final part of Chapter 3 showed that our local search outperforms the algorithms in literature.

Our study on couple-operation scheduling raised many interesting questions to be investigated. One question regards the connectivity of the neighbourhood designed: in particular it is important to establish whether the solution neighbourhood is optimally connected: does there exists a sequence of neighbourhood moves connecting any feasible solution to an optimal one? Establishing this property would give us an indication on the quality of the solutions generated. In fact, if the neighbourhood is optimally connected, the quality of the solution achieved would depend only on the quality of the anti-cycle and escape mechanisms put in place and on the maximum time allowed for the heuristic to run. Differently, if the neighbourhood is not optimally connected, then, in order to achieve better solutions for any instance, the research should focus on finding a better neighbourhood structure and solution representation.

Another interesting research direction is the study of possible compact representations for the solution schedules. In fact, it would be beneficial to determine whether the coupled-operation scheduling problem with fixed separated permutations of first and second operations is solvable in polynomial time. We believe that the technique used to solve the reinsertion problem with the short cycle property has applications in the generation of compact solution representations.

Finally, further improvement of the tabu-search designed may be achieved via more accurate lower-bound calculation procedures. The necessity of better lower-bound calculation procedure has been underlined during our computational experience which showed that optimality of the solution calculated could be proved only for a few special instances. Such a task is particularly challenging since lower-bounds based on the solution of relaxed

problems seem to be hard to design. In fact, there exists a few polynomial-time algorithms for special constrained problems that could lead to such a type of lower bounds.

A wider research direction includes the investigation of the possibility to apply the techniques introduced in [43] to other scheduling problems. For example, we believe that the short cycle property could be proved to hold for the disjunctive graph representation of other job re-insertion scheduling problems with job chains.

In Chapter 4 we consider a full real-world version of the problem of booking chemotherapy treatment appointments in an outpatient clinic and we develop a complete solution approach for it. In order to deal with the stochastic nature of the patient arrivals and the complex delivery patterns involved, we developed the innovative concept of a multi-level template schedule which consists of a well-thought through schedule in which arriving appointments can be fitted over time. The template schedule is flexible and can be adapted to the variation of patient arrival rates. We designed several integer linear programs which are run iteratively to construct and maintain the template schedule in the long run. We evaluated our approach using artificial data and mirroring the arrival process documented in historical data from a clinic. Obtained results showed that, using the same resources, our approach schedules 20% more appointments, eliminates almost all nurse clashing activities to be performed by the nurse and maintain short patient waiting days and times at the clinic. It has been recognised that the benefits for the clinic are not restricted to resource efficiency but also include safer treatments and a better patient and staff experience.

The introduction of the concept of a multi-level template schedule opens new research directions from both practical and theoretical sides. The first direction comprises the evaluation of the actual performance of the developed approach in a running outpatient chemotherapy clinic. Thus, the natural prosecution of the work in this thesis is the implementation of the designed algorithms in a prototype software and its use to book actual appointments for arriving patients in a real clinic. The deployment of such a software could gather detailed data of the arrival process which could contribute to a better tuning of the algorithms parameters and to refining the algorithm's design. The performance of the approach could also be improved by a deep statistical analysis of the patient arrival process and its influence on the behaviour of the multi-level template schedule.

A wider research direction is the study of possible applications of the multi-level template schedule to other online scheduling problems. The idea of generating a schedule that embeds multiple scheduling decisions for the same activities and resources has great potential to improve the quality of the solution achieved in the presence of uncertain demands. The work in this thesis showed its application only to appointment booking but we

believe that such an approach can be used in many other problems in manufacturing and service industry where scheduling decisions are part of a contained stochastic process.

Bibliography

[1] World cancer report. , P. Boyle and B. Levin, editors, World Health Organization, 2008.

[2] J. Adam, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988.

[3] A. Ageev and A. Barburin. Approximation algorithms for UET scheduling problems with exact delays. *Operations Research Letters*, 35(4):533–540, 2007.

[4] A. Ageev and A. Kononov. Approximation algorithms for scheduling problems with exact delays. *Lecture Notes in Computer Science*, 4368:1–14, 2007.

[5] Z. Agur, R. Hassin, and S. Levy. Optimizing chemotherapy scheduling using local search heuristics. *Operations Research*, 54(5):829–846, 2006.

[6] D. Ahr, J. Békési, G. Galambos, M. Oswald, and G. Reinelt. An exact algorithm for scheduling identical coupled tasks. *Mathematical Methods of Operations Research*, 59(2):193–203, 2004.

[7] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[8] N. Bailey. A study of queues and appointment systems in hospital outpatient departments, with special reference to waiting-times. *Journal of the Royal Statistical Society*, A14:185–189, 1952.

[9] E. Balas, J. K. Lenstra, and A. Vazacopoulos. The one machine problem with delayed precedence constraints and its use in job shop scheduling. *Management Science*, 41(1):94–109, 1995.

[10] P. Baptiste. Batching identical jobs. *Mathematical Methods of Operations Research*, 52:355–367, 2000.

[11] P. Baptiste. A note on scheduling identical coupled tasks in logarithmic time. *Discrete Applied Mathematics*, 158(5):583–587, 2010.

[12] M. Bartusch, R. Möhring, and F. Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16(1):199–240, 1988.

[13] J. Blazewicz, K. Ecker, T. Kis, C. N. Potts, M. Tanas, and J. Whitehead. Scheduling of coupled tasks with unit processing times. *Journal of Scheduling*, 13(5):453–461, 2010.

[14] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[15] M. Brahimi and D. J. Worthington. Queueing models for out-patient appointment systems – a case study. *Journal of the Operational Research Society*, 42(9):733–746, 1991.

[16] N. Brauner, G. Finke, V. Lehoux-Lebacque, C. N. Potts, and J. Whitehead. Scheduling of coupled tasks and one-machine no-wait robotic cells. *Computers & Operations Research*, 36(2):301–307, 2009.

[17] K. Brinkmann and K. Neumann. Heuristic procedures for resource-constrained project scheduling with minimal and maximal time lags: the resource-levelling and minimum project-duration problems. *Journal of Decision Systems*, 5:129–155, 1996.

[18] P. Brucker. A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics*, 94:77–99, 1999.

[19] P. Brucker. *Scheduling Algorithms*. Springer, Berlin, 2007.

[20] P. Brucker, B. Jurisch, and B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:107–127, 1994.

[21] P. Brucker, S. Knust, and C. Oguz. Scheduling chains with identical jobs and constant delays on a single machine. *Mathematical Methods of Operations Research*, 63(1):63–75, 2006.

[22] J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11:42–47, 1982.

[23] J. Carlier, M. Haouari, M. Kharbeche, and A. Moukrim. An optimization-based heuristic for the robotic cell problem. *European Journal of Operational Research*, 202(3):636–645, 2010.

[24] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176, 1989.

[25] T. Cayirli and E. Veral. Outpatient scheduling in health care: a review of literature. *Production & Operations Management*, 12(4):519–548, 2003.

[26] K. Chakhlevitch, C. A. Glass, and H. Kellerer. Batch machine production with perishability time windows and limited batch size. *European Journal of Operational Research*, 210(1):39–47, 2011.

[27] T. C. E. Cheng. Survey of scheduling research involving due date determination decisions. *European Journal of Operational Research*, 38(2):156–166, 1989.

[28] T. C. E. Cheng, J. J. Yuan, and A. F. Yang. Scheduling a batch-processing machine subject to precedence constraints, release dates and identical processing times. *Computers & Operations Research*, 32:849–859, 2005.

[29] D. Conforti, F. Guerriero, and R. Guido. Optimization models for radiotherapy patient scheduling. *4OR: A Quarterly Journal of Operations Research*, 6(3):263–278, 2008.

[30] M. Dell'Amico. Shop problems with two machines and time lags. *Operations Research*, 44(5):777–787, 1996.

[31] R. Dobish. Next-day chemotherapy scheduling: a multidisciplinary approach to solving workload issues in a tertiary oncology center. *Journal of Oncology Pharmacy Practice*, 9(1):37–42, 2003.

[32] M. Elshafei, H. D. Sherali, and J. C. Smith. Radar pulse interleaving for multi-target tracking. *Naval Research Logistics*, 51(1):72–94, 2004.

[33] A. Farina and P. Neri. Multitarget interleaved tracking for phased-array radar. *Communications, Radar and Signal Processing, IEEE Proceedings, Part F*, 127(4):312–318, 1980.

[34] R. B. Fetter and J. D. Thompson. Patients' waiting time and doctors' idle time in the outpatient setting. *Health Services Research*, 1(1):66–90, 1966.

[35] L. Finta and Z. Liu. Single machine scheduling subject to precedence delays. *Discrete Applied Mathematics*, 70(3):247–266, 1996.

[36] J. Fondrevelle, A. Oulamara, and M. C. Portmann. Permutation flowshop scheduling problems with maximal and minimal time lags. *Computers & Operations Research*, 33:1540–1556, 2006.

[37] J. Fondrevelle, A. Oulamara, and M. C. Portmann. Permutation flowshop scheduling problems with time lags to minimize the weighted sum of machine completion times. *International Journal of Production Economics*, 112(1):168–176, 2008.

[38] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, 1979.

[39] M. R. Garey, D. S. Johnson, B. Simons, and R. E. Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM Journal on Computing*, 10(2):256–269, 1981.

[40] F. W. Glover and M. Laguna. *Tabu search*. Springer, 1st edition, 1998.

[41] J. Grabowski, E. Nowicki, and S. Zdrzalka. A block approach for single-machine scheduling with release dates and due dates. *European Journal of Operational Research*, 26:278–285, 1986.

[42] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 4:287–326, 1979.

[43] H. Gröflin and A. Klinkert. Feasible insertions in job shop scheduling, short cycles and stable sets. *European Journal of Operational Research*, 177(2):763–785, 2007.

[44] H. Gröflin and A. Klinkert. A new neighborhood and tabu search for the blocking job shop. *Discrete Applied Mathematics*, 157(17):3643–3655, 2009.

[45] H. Gröflin, A. Klinkert, and N. Dinh. Feasible job insertions in the multi-processor-task job shop. *European Journal of Operational Research*, 185(3):1308–1318, 2008.

[46] D. Gupta and B. Denton. Appointment scheduling in health care: challenges and opportunities. *IIE Transactions*, 40(9):800–819, 2008.

[47] A. Hertz, E. Taillard, and D. De Werra. Tabu search. In J. K. Lenstra and E. Aarts, editors, *Local search in combinatorial optimization*, chapter 5, 91–120. Princeton University Press, 2003.

[48] W. A. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21(1):177–185, 1974.

[49] J. Hurink and J. Keuchel. Local search algorithms for a single-machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics*, 112:179–197, 2001.

[50] Y. Ikura and M. Gimple. Efficient scheduling algorithms for a single batch processing machine. *Operations Research Letters*, 5(2):61–65, 1986.

[51] J. R. Jackson. Scheduling a production line to minimize maximum tardiness. Technical report, University of California, Los Angeles, 1955.

[52] T. R. Jensen and B. Toft. *Graph Coloring Problems*. Wiley Series in Discrete Mathematics and Optimization. Wiley Interscience, 1 edition, December 1994.

[53] P. Kalczynski and J. Kamburowski. On no-wait and no-idle flow shops with makespan criterion. *European Journal of Operational Research*, 178(3):677–685, 2007.

[54] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[55] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244:1093–1096, 1979.

[56] B. J. Lageweg, J. K. Lenstra, and A. H. G. Rinnooy Kan. Minimizing maximum lateness on one machine: computational experience and some applications. *Statistica Neerlandica*, 30(1):25–41, 1976.

[57] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: algorithms and complexity. In S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin, editors, *Logistics of Production and Inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, chapter 9, 445–522. Elsevier, 1993.

[58] C. Y. Lee, R. M. Uzsoy, and L. A. M. Vega. Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40(4):764–775, 1992.

[59] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. *Complexity of Machine Scheduling Problems, Annals of Discrete Mathematics*,1:343–362. Elsevier, 1977.

[60] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1997.

[61] J. Y. T. Leung, H. Li, and H. Zhao. Scheduling two-machine flow shops with exact delays. *International Journal of Foundations of Computer Science*, 18(2):341–359, 2007.

[62] J. Y. T. Leung and H. R. Zhao. Minimizing sum of completion times and makespan in master-slave systems. *IEEE Transactions on Computers*, 55(8):1–22, 2006.

[63] H. Li and H. Zhao. Scheduling coupled-tasks on a single machine. In *2007 IEEE Symposium on Computational Intelligence in Scheduling*, 137–142, 2007.

[64] C. K. Y. Lin and K. B. Haley. Scheduling two-phase jobs with arbitrary time lags in a single-server system. *IMA Journal of Mathematics Applied in Business & Industry*, 5:143–161, 1993.

[65] D. V. Lindley. The theory of queues with a single server. *Mathematical Proceedings of the Cambridge Philosophical Society*, 48:277–289, 1952.

[66] M. Manier and C. Bloch. A classification for hoist scheduling problems. *International Journal of Flexible Manufacturing Systems*, 15:37–55, 2003.

[67] A. Mascis and D. Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498–517, 2002.

[68] A. Munier and F. Sourd. Scheduling chains on a single machine with non-negative time lags. *Mathematical Methods of Operations Research*, 57(1):111–123, 2003.

[69] A. Munier-Kordon and D. Rebaine. Polynomial time algorithms for the UET permutation flowshop problem with time delays. *Computers & Operations Research*, 35(2):525–537, 2008.

[70] A. Munier-Kordon and D. Rebaine. The two-machine open-shop problem with unit-time operations and time delays to minimize the makespan. *European Journal of Operational Research*, 203(1):42–49, 2010.

[71] G. Ochoa, M. Villasana, and E. K. Burke. An evolutionary approach to cancer chemotherapy scheduling. *Genetic Programming and Evolvable Machines*, 8(4):301–318, 2007.

[72] Department of Health. Cancer reform strategy. Technical report, Department of Health, UK, 2009.

[73] S. Ogulata, M. Cetik, E. Koyuncu, and M. Koyuncu. A simulation approach for scheduling patients in the department of radiation oncology. *Journal of Medical Systems*, 33(3):233–239, 2009.

[74] A. J. Orman and C. N. Potts. On the complexity of coupled-task scheduling. *Discrete Applied Mathematics*, 72(1-2):141–154, 1997.

[75] A. J. Orman, C. N. Potts, A. K. Shahani, and A. R. Moore. Scheduling for a multifunction phased array radar system. *European Journal of Operational Research*, 90(1):13–25, 1996.

[76] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1993.

[77] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization : Algorithms and Complexity*. Dover Publications, 1998.

[78] D. Petrovic, M. Morshed, and S. Petrovic. Genetic algorithm based scheduling of radiotherapy treatments for cancer patients. In C. Combi, Y. Shahar, and A. Abu-Hanna, editors, *Artificial Intelligence in Medicine*, volume 5651, 101–105. Springer Berlin, Berlin, 2009.

[79] S. Petrovic, W. Leung, X. Song, and S. Sundar. Algorithms for radiotherapy treatment booking. In R. Qu, editor, *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG'2006)*, 105–112, 2006.

[80] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems (2nd Edition)*. Prentice Hall, 2001.

[81] C. N. Potts and J. D. Whitehead. Heuristics for a coupled-operation scheduling problem. *Journal of the Operational Research Society*, 58(10):1375–1388, 2007.

[82] D. Rebaine and V. A. Strusevich. Two-machine open shop scheduling with special transportation times. *The Journal of the Operational Research Society*, 50(7):756–764, 1999.

[83] J. Riezebos. Time lag size in multiple operations flow shop scheduling heuristics. *European Journal of Operational Research*, 105(1):72–90, 1998.

[84] J. Riezebos, G. J. C. Gaalman, and J. N. D. Gupta. Flow shop scheduling with multiple operations and time lags. *Journal of Intelligent Manufacturing*, 6(2):105–115, 1995.

[85] B. Roy. Contribution de la théorie des graphes à l'étude de certains problèms linéaires. *Comptes rendus de séance de l'Académie des Sciences*, 2437–2439, 1959.

[86] T. Sen and S. K. Gupta. A state-of-art survey of static scheduling research involving due dates. *Omega*, 12(1):63–76, 1984.

[87] D. Shabtay. Due date assignments and scheduling a single machine with a general earliness/tardiness cost function. *Computers and Operations Research*, 35:1539–1545, 2008.

[88] R. D. Shapiro. Scheduling coupled tasks. *Naval Research Logistics*, 27(3):489–498, 1980.

[89] G. Sheen and L. Liao. A branch and bound algorithm for the one-machine scheduling problem with minimum and maximum time lags. *European Journal of Operational Research*, 181(1):102–116, 2007.

[90] H. D. Sherali and J. Smith. Interleaving two-phased jobs on a single machine. *Discrete Optimization*, 2(4):348–361, 2005.

[91] G. Simonin, B. Darties, R. Giroudeau, and J. C. König. Isomorphic coupled-task scheduling problem with compatibility constraints on a single processor. In *MISTA'04 : 4th Multidisciplinary International Scheduling Conference : Theory and Applications*, 378–388, 2009.

[92] G. Simonin, R. Giroudeau, and J. C. König. Complexity and approximation for scheduling problem for a torpedo. *Computers & Industrial Engineering*, 2011.

[93] B. Simons. Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. *SIAM Journal on Computing*, 12(2):294–299, 1983.

[94] B. Simons and M. K. Warmuth. A fast algorithm for multiprocessor scheduling of unit-length jobs. *SIAM Journal on Computing*, 18(4):690–710, 1989.

[95] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceeding of the London Mathematical Society*, 42(1):230–265, 1937.

[96] A. Turkcan, M. Lawley, and B. Zeng. Chemotherapy operations planning and scheduling. *Optimization Online*, 2010.

[97] J. D. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10:384–393, 1975.

[98] N. Vakhania. A better algorithm for sequencing with release and delivery times on identical machines. *Journal of Algorithms*, 48(2):273–293, 2003.

[99] I. Vermeulen, S. Bohte, P. Bosman, S. Elkhuizen, P. Bakker, and J. La Poutré. Optimization of online patient scheduling with urgencies and preferences. *Lecture Notes in Computer Science*, 5651:71–80, 2009.

[100] J. Vissers. Selecting a suitable appointment system in an outpatient setting. *Medical Care*, 17(12):1207–1220, 1979.

[101] A. Wijewickrama and S. Takakuwa. Simulation analysis of appointment scheduling in an outpatient department of internal medicine. In M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, editors, *Proceedings of the 2005 Winter Simulation Conference*, 2264–2273. Winter Simulation Conference, 2005.

[102] E. Wikum, D. C. Llewellyn, and G. L. Nemhauser. One-machine generalized precedence constrained scheduling problems. *Operations Research Letters*, 16(2):87–99, 1994.

[103] P. Williams, G. Tai, and Y. Lei. Simulation based analysis of patient arrival to health care systems and evaluation of an operations improvement scheme. *Annals of Operations Research*, 178:263–279, 2009.

[104] E. Winter and P. Baptiste. On scheduling a multifunction radar. *Aerospace Science and Technology*, 11(4):289–294, 2007.

[105] D. A. Wismer. Solution of the flowshop-scheduling problem with no intermediate queues. *Operations Research*, 20(3):689–697, 1972.

[106] L. A. Wolsey and G. L. Nemhauser. *Integer and Combinatorial Optimization*. Wiley-Interscience, 1999.

[107] D. L. Yang and M. S. Chern. A two-machine flowshop sequencing problem with limited waiting time constraints. *Computers and Industrial Engineering*, 28:63–70, 1995.

[108] W. Yu. *The two-machine flow shop problem with delays and the one-machine total tardiness problem.* PhD thesis, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, 1996.

[109] W. Yu, H. Hoogeveen, and J. K. Lenstra. Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard. *Journal of Scheduling*, 7(5):333–348, 2004.