High multiplicity scheduling problems

High Multiplicity Scheduling Problems

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit Maastricht,
op gezag van de Rector Magnificus,
Prof. dr. A.C. Nieuwenhuijzen Kruseman,
volgens het besluit van het College van Decanen,
in het openbaar te verdedigen
op donderdag 20 november 2003 om 16.00 uur

door

Alexander Grigoriev

Promotores:
Prof. dr. ir. A.W.J. Kolen
Prof. dr. Y. Crama (University of Liege)
Co-promotor:
Dr. J.J. van de Klundert

Beoordelingscommissie:
Prof. dr. C.P.M. van Hoesel (voorzitter)
Prof. dr. V. Strusevich (University of Greenwich)
Dr. M. Uetz
Prof. dr. G. Woeginger (Universiteit Twente)

High Multiplicity Scheduling Problems

Proefschrift Universiteit Maastricht
ISBN 90-9017338-2

Preface

About four years ago I started my PhD-research at the Department of Quantitative Economics of the Faculty of Economics and Business Administration at Maastricht University. These four years proved to be a very interesting and enjoyable period of my life. I was introduced to the international research community and visited many scientific events in different parts of the world.

First of all, I want to heartily thank my supervisors, Antoon Kolen, Yves Crama, and Joris van de Klundert, for haven given me the opportunity to join the scientific world and academic community, for their support, their trust and their continuous advise. This thesis would never exist without the contribution of these people.

I feel much indebted also to Gerhard Woeginger, Frits Spieksma, and Nadia Brauner. Many results of this thesis were written in joint papers and for me it was a big pleasure to work with them. I am also thankful to Wieslaw Kubiak, Renato Monteiro, Leen Stougie, and Jos Sturm for reading the drafts of my papers, for their comments and for providing me with many useful references.

I want to thank Stan van Hoesel, Marc Uetz, and Vitaly Strusevich for awakening and developing my interests in many other fields of combinatorial optimization other than high multiplicity scheduling. My thanks go to Roel Bovendeerd, Corinne Feremans, Sonja Kovaleva, Rudolf Müller, and Dolores Romero for the discussions on numerous topics in operations research and combinatorial optimization.

I am grateful to my Russian colleagues, teachers and friends, Vladimir Beresnev, Youri Kochetov, Alexander Kononov, and Maxim Sviridenko, for their attention, support and help.

My special thanks go to Jan-Willem Goossens and Anton van der Kraaij, who were my C++ tutors and advisors.

I would like to express my gratitude to the Department of Quantitative

Economics and particularly to Hans Peters, Dries Vermeulen, and especially to our secretaries Karin van den Boorn and Haydee Hallmanns.

I am grateful to my friends in The Netherlands Karlygash Abildaeva, Vincent Feltkamp, Ludie Janssen, Annelies and Erik Rulands for creating a nice atmosphere where I was able to work effectively and write the thesis in time.

Finally, I address my warm thanks to my wife, my parents and my son for their moral support during these four years.

Maastricht, August 2003
Alexander Grigoriev

Contents

Chapter 1

Introduction to high multiplicity scheduling

1.1 Introduction

This thesis is about high multiplicity scheduling. The term high multiplicity scheduling appears to have been first used by Hochbaum and Shamir [58], to refer to a special type of scheduling problems in which "the jobs can be partitioned into relatively few groups (or types), and in each group all the jobs are identical, i.e., they have the same set of parameters".

In recent literature (see, e.g., [2, 7, 9, 12, 15, 17, 21, 22, 41, 58, 68, 69, 76, 105]), many scheduling problems have been studied which can be viewed as high multiplicity scheduling problems, according to this informal definition. However, despite their commonalities, these articles have not been considered as a collection of interrelated literature. This thesis aims to collect, structure and advance the theory of high multiplicity scheduling.

To make a more precise definition of high multiplicity scheduling problems, let us go back to the definition of scheduling problems. Scheduling problems are problems focused on the efficient allocation of one or more resources to activities over time [19]. The efficiency of the allocation is measured by some objective function and the problem is to find an allocation such that the corresponding objective function value satisfies a specified property (e.g., attains its minimum or maximum value over all possible allocations).

In machine scheduling problems, see [19] or [40], the activities are usually referred to as jobs, and the resources are machines. Here, the allocation of resources to activities over time requires to specify for each job and for each

machine zero ore more time moments when the job starts to process on the machine and correspondingly time moments when the job stops to process on the machine. Such a pair of time moments is called a pair of starting and ending times, and a set of such pairs is called a schedule.

Depending on the nature of the scheduling problems there are requirements and restrictions on the feasibility of schedules. For example, it is common that machines can only process one job at a time. This entails that for a schedule to be feasible, all pairs of starting and ending times on a machine must define non overlapping time intervals. Similarly jobs can usually only be processed on one machine at a time. Also, it is common that jobs must be processed for a certain amount of time, which is called the processing time. In some cases all processing time of a job on machine must be consecutive, such that the single time interval defined by its starting and ending time must have length equal to the processing time.

In addition to the above described jobs and machines, and schedule requirements and restrictions, other items and issues may play a role in scheduling problems. The processing or transfer between machines or jobs may require additional materials or resources or set up times. There are many such possibilities and, in general, we call them job attributes (see Chapter 2). Notice that if a job requires a resource or a raw material to be processed, the problem input should also define the availability of these requirements.

Scheduling problems in general, and machine scheduling problems in particular have received considerable attention ever since the late 1950's, [38, 62, 63, 86]. Scheduling has become a research field of its own, with many practicable applications and a sizeable and respectable body of theory.

From a theoretical viewpoint, an important characteristic of a scheduling problem is its computational complexity. Computational complexity theory deals with the issue of measuring the time it takes to solve a problem in the size of the problem specification. More precisely, (see Garey and Johnson [34] for terminology), each problem instance I has a certain length $|I|$, and the computation time of an algorithm to solve the problem is defined as a function of $|I|$. The length of a problem instance is usually referred to as the size of the input. An algorithm is said to be polynomial if the computation time is bounded from above by a polynomial function in the size of the input.

With these, somewhat informal, definitions from complexity theory at hand, we are now able to pose a natural definition of high multiplicity scheduling problems: A scheduling problem is a high multiplicity scheduling problem if the minimal cardinality of the set of starting time - ending

time pairs defining a feasible schedule is not necessarily polynomial in the size of the input. Extensions of this definition are investigated in Chapter 5.

The reader should notice the following: any algorithm that gives a solution to a high multiplicity scheduling problem by specifying all pairs of starting and ending times is not polynomial since it is not possible to output a superpolynomial number of pairs in polynomial time. This feature of high multiplicity scheduling problems contrasts sharply with the status of traditional scheduling problems, where the set of pairs is polynomial in the input size. We refer to such problems as single multiplicity problems. From a complexity viewpoint this means that whereas single multiplicity problems, and indeed most traditional scheduling problem, are easily seen to be in NP, this is not the case for high multiplicity scheduling problems. Thus, we conclude that their complexity status differs significantly from the complexity status of traditional scheduling problems.

We conclude this section by providing an example that illustrates the discussion above.

Example 1.1.1 (INTEGER KNAPSACK).

Consider the following scheduling problem. Given is a set of jobs J together with job importance coefficients w_j, $j \in J$, and processing times p_j, $j \in J$. Ideally all the jobs have to be completed by the common due date D. It is required to find a single machine schedule that maximizes the total importance of not late jobs.

This problem can be modelled as the well known 0-1 KNAPSACK problem, see, e.g., [93], and it can be written as the following integer linear program:

$$\max_{x} \sum_{j \in J} w_j x_j \tag{1.1}$$

subject to

$$\sum_{j \in J} p_j x_j \leq D; \tag{1.2}$$

$$x_j \in \{0, 1\}, \quad j \in J, \tag{1.3}$$

where x_j takes value 1 if job j is not late and 0 otherwise.

Notice that a feasible solution of the problem is represented here by a partition of the set J into late and not late jobs. Given such a partition, we

can construct in polynomial time a list of starting-ending times for all the jobs. To do this just schedule first all not late jobs in any order and then schedule the late jobs. Since the set of starting-ending times for any feasible schedule has cardinality $O(|J|)$ the problem is clearly a single multiplicity one.

Now, assume that not individual jobs but groups of individual jobs are given. Set J becomes a set of job types and the input of the problem consists of attribute triples (w_j, p_j, n_j), $j \in J$, where w_j, p_j and n_j are the importance coefficient, the processing requirement and the number of individual jobs of type $j \in J$ respectively. This problem is known as INTEGER KNAPSACK problem and it can be modelled by the following integer linear program (IK):

$$\max_{x} \sum_{j \in J} w_j x_j \tag{1.4}$$

subject to

$$\sum_{j \in J} p_j x_j \leq D; \tag{1.5}$$

$$0 \leq x_j \leq n_j, \quad j \in J; \tag{1.6}$$

$$x_j \in \mathbb{Z}^+, \quad j \in J, \tag{1.7}$$

where x_j is the number of jobs of type j which are not late.

In this case, the cardinality of a set of starting-ending times of a feasible schedule is $O(\sum_{j \in J} n_j)$ which is superpolynomial in the input length $O(\sum_{j \in J} \log n_j)$ of the problem. Thus, INTEGER KNAPSACK is by definition a high multiplicity problem.

As we discussed above, in high multiplicity scheduling the following natural question arises: is it possible to encode the schedule (the set of starting-ending times) polynomially in the input size of the problem. For INTEGER KNAPSACK we are able to manage this. Consider any job type. Let us group late individual jobs of this type in one batch and not late individual jobs in another one. Then given a list of starting-ending times of batches, the starting time of the k-th individual job of type j can be determined by the following polynomial size mapping

$$s_{j,k} = \begin{cases} S_j^1 + (k-1)p_j, & \text{if } k \leq x_j \text{ (if the job is not late);} \\ S_j^2 + (k-1)p_j, & \text{otherwise,} \end{cases} \tag{1.8}$$

where S_j^1 is the starting time of not late batch, S_j^2 is the starting time of late batch, and x_j is the number of not late individual jobs of type $j \in J$. Since the mapping does not involve any attributes of an individual job but just the index of it, the space we need to specify the mapping is polynomial in the input size of the problem. □

1.2 Motivation

High multiplicity scheduling problems possess some characteristics which are, in our opinion, not thoroughly investigated or well understood in scheduling theory. In this section we set out to show that both from a practical as well as from a theoretical background, a better understanding of high multiplicity problems is called for.

Many of the basic scheduling problems stem from applications in production environments or processor scheduling that date back to the 1960's. At the time, two extremes of process structures were common in industrial processes. On the one hand the job shop, in which single products are made to customer order, in a non standard process, yielding high costs. On the other hand, high volumes where being produced to stock in process and discrete manufacturing settings, in highly standardized processes, yielding low costs but zero customization. At present these two process structures are outdated. Advances in marketing, production technology and management, and globalization have in many markets led to conditions in which high degrees of customization must be combined with low production costs. To satisfy these conditions production must be structured so that standardized high volume processes are able to deliver a huge variety of customized products.

The desired customization is often reached by letting the customer compose his or her own product variation by selection optional parts and colors from a prespecified list. Notice how it differs from letting the customer design his or her own product. By letting the customer select options from a prespecified set the process can still be highly standardized and be executed at low cost.

The resulting product portfolios may consist of thousands if not millions of product variations, and many of these variations have rather short life cycles. Hence, keeping inventory of all end product variations is an unaffordable option. By consequence, production must be performed on customer order. Not necessarily so for the complete chain of production steps required

to produce an end item, but it certainly applies to the final assembly or configuration steps which are customer order specific. Process structures which are designed in such a way often produce components or raw materials for components to stock, and end products to customer order. This logistic design is referred to as Assemble to Order (ATO), or Configure to Order (CTO). In view of the short planning horizon and the differences in the production details, scheduling is often an important issue in ATO/CTO environments. See, for instance, in Miltenburg [89] for a seminal paper on scheduling problems in this context. Likewise capacity planning has become much more complicated since each of the many different product variations places different demands on the available capacity. See, for instance, Drexl and Kimms [28].

Such practical settings often entail scheduling problem instances in which there are many jobs, some of which may be identical or almost identical. When the (almost) identical jobs are grouped together however, the instance can often be viewed to be an instance of a slightly different scheduling problem in which there are relatively few groups containing a multiplicity of identical jobs. The resulting high multiplicity scheduling problems in operational sequencing satisfy the definition of high multiplicity scheduling as given in Section 1.1 naturally applies. In addition, high multiplicity problems may arise when dealing with medium or long term objectives such as determining system capacity and/or optimizing average throughput rates. Hence, we conclude that modern logistic and production management approaches naturally give rise to high multiplicity scheduling problems.

The resulting high multiplicity problems have been widely investigated in the context of assembly line balancing [4, 72, 74, 76, 89, 105] and in various flow shop scheduling settings [2, 14]. Some authors somehow overlook or disregard the high multiplicity feature of such problems, and indeed others notice and study them and identify new research directions. In the subsequent section we discuss high multiplicity scheduling problems from a more theoretical background.

As has become clear from Example 1.1.1 and several authors [59, 85] high multiplicity features can often be naturally introduced in many combinatorial problems other than machine scheduling problems. Indeed, INTEGER KNAPSACK is the high multiplicity version of the 0-1 KNAPSACK problem, the TRANSPORTATION PROBLEM is a high multiplicity version of the MATCHING PROBLEM, for definitions see, e.g., [93], and as will become clear in Chapter 2, many counting problems also posses high multi-

plicity characteristics. Hence, we have chosen to study other combinatorial problems in which high multiplicity can be practically motivated to be of interest as well. Chapter 3 contains a thorough investigation of the high multiplicity TRAVELLING SALESMAN PROBLEM, which is closely related to the NO-WAIT FLOW SHOP scheduling problem. Chapter 4 deals with high multiplicity scheduling problems arising in the context of maintenance scheduling [7, 6, 9] and broadcasting [67, 68, 100]. These scheduling problems are not machine scheduling problems as we have defined them above, but are closely related to high multiplicity machine scheduling problems arising in the context of assembly line balancing, as they are studied in Chapter 2 and by [4, 72, 74, 76, 89, 105]. Finally, Chapter 6 addresses project scheduling problems, and some of the high multiplicity issues which can be naturally introduced in this setting.

1.3 Research issues in high multiplicity scheduling

In this section we enlist research issues related to high multiplicity scheduling problems. As mentioned before, high multiplicity scheduling problems fall naturally in the broader classes scheduling and combinatorial optimization, and much of the techniques that have been developed in this area can be applied to high multiplicity scheduling problems. Illustrations can be found throughout this thesis and by references such as [7, 6, 9, 59, 67, 68, 85, 100] to name a few which borrow from different areas of combinatorial optimization. In this thesis we will of course make frequent use of results and techniques which are available, but since it is our intention to explore the new difficulties posed by high multiplicity scheduling problems, the emphasis will be on features of high multiplicity scheduling problems that are not encountered in single multiplicity problems. The same holds for this section.

A first phenomenon in high multiplicity scheduling that makes it different from single multiplicity problems stems directly from its definition: the number of starting and ending time pairs (see Section 1.1) is not polynomial in the input size of the problem. However, in general decision versions of single multiplicity scheduling problems can easily be proven to be in NP because a feasible schedule - in terms of pairs of starting and ending times - suffices as a polynomial certificate. By definition this is not the case for high multiplicity problems, and therefore, their complexity analysis is quite

different to start with. This phenomenon is recognized by many authors [17, 21, 22, 41, 58, 73, 75, 77], but disregarded in several more applied papers [48, 49]. Interestingly, there are several basic high multiplicity scheduling problems whose complexity is open [7, 69, 104].

From the previous section it is clear that high multiplicity scheduling problems can be viewed to be harder than their single multiplicity counterparts. Notice however, that our definition excludes single multiplicity from being a special case of high multiplicity. Nevertheless, viewed from another angle, high multiplicity scheduling problems can also be seen to be significantly more structured versions of single multiplicity problems. Consider any NP-complete machine scheduling problem with n jobs. Now, suppose that it is also given that these jobs fall into K classes, $K \ll n$. Then it may *happen that the problem becomes polynomially solvable when* K is bounded, for example, by $\log n$, or a constant. For instance, it is well known [23] that the TRAVELLING SALESMAN PROBLEM (TSP) is polynomially solvable when the number of cities is fixed. What is more, any scheduling problem which can be modelled as an integer linear program in which the number of variables is a function (not necessarily polynomially bounded) of a number of classes which is bounded by a constant, is polynomially solvable by Lenstra's algorithm [80].

In recent years, techniques have been developed for the design of approximation algorithms and approximation schemes which are also based on the idea of partitioning the job sets into classes of jobs whose processing requirements are alike, see, e.g., [56, 106]. A deeper understanding of the complexity of high multiplicity scheduling problems could therefore also contribute to improvements in approximation algorithms.

A customary, albeit often implicit, requirement in complexity theory, is that the encoding of a problem is efficient (see, for instance, Garey and Johnson [34]). Efficiency here roughly requires that the encoding scheme does not require more space than necessary to specify instances. Now, consider a high multiplicity scheduling problem where the job set can be partitioned into classes where all jobs in any same class have identical processing requirements. For such a problem, any encoding scheme that specifies the processing requirements per job is not an efficient encoding. Indeed, efficient encoding schemes describe the processing requirements per class, and specify (binary encoded) the number of jobs in each class.

By consequence, we have that if a single multiplicity problem is in P, then the corresponding high multiplicity problem can be solved in pseudopolyno-

mial time. If the multiplicities are unary, and hence inefficiently, encoded, the problem becomes polynomially solvable. Notice however, that under the tacit assumption that the encoding scheme is efficient, the problem then does not satisfy our definition of a high multiplicity scheduling problem anymore. Nevertheless, several high multiplicity scheduling problems are considered under such inefficient encoding schemes [48, 49, 89].

For high multiplicity scheduling problems, the complexity issues raised above, which are a mandatory feature, have not always led to satisfactory answers in terms of the traditional complexity classes polynomially solvable, pseudopolynomially solvable, and NP-complete. A similar situation also arises in counting problems. For example, Johnson et al [64] have considered problems such as counting the number of maximal independent sets of a graph. In such a problem, the answer, a number, is not necessarily polynomial in the input size of the problem, nor is the following natural certificate: a list of all maximal independent sets. Nevertheless, many interesting questions about (optimal) solutions remain. For example, in a scheduling problem, one may consider the following question: given a job, what are its starting and ending time? In Chapter 5 we shall see that it is possible that this question can be answered in polynomial time, even when it is not known, how to compute the optimal solution in polynomial time. (Notice however, that answering this question for every job easily entails a pseudopolynomial procedure.)

Complexity issues of this type are investigated by several authors [12, 73, 77, 88, 105]. Chapter 2 provides a thorough discussion of these issues and the related literature. In addition, we propose several different natural questions and output encoding schemes for high multiplicity scheduling problems. The ideas and proposed framework enable us to classify and interrelate the existing literature, and identifies several opportunities for future research.

The high multiplicity TSP as it is studied in Chapter 3 also entails encoding related complexity issues. In the classical TSP, the input consists of a distance matrix, encoding the intercity distances between a set of cities, and it is the task of the the salesman to find a short, (shortest possible) tour in which each of the cities is visited exactly once. In the high multiplicity TSP, every city i has a multiplicity n_i, and the salesman has to make a tour along the cities, such that every city i is visited n_i times. When the output, i.e., the tour of the travelling salesman, is encoded as an explicit sequence of cities, the length of the output is not polynomial in the input size (assuming the input is efficiently encoded). Nevertheless, the problem is known to be poly-

nomially solvable when the number of cities is fixed [23], or using Lenstra's algorithm [80] to solve an ILP with a constant number of variables. This result helps to answer the decision version of the problem, but not necessarily question such as 'What is the j-th city in some optimal tour?" Chapter 3 shows how to efficiently encode a high multiplicity tour, from the solution of a standard ILP formulation for the high multiplicity TSP.

Another question is also easily understood in terms of the high multiplicity TSP. If all multiplicities are doubled, does the length of the optimal tour also double? Consider, for instance, Example 3.3.7 from Chapter 3. Given

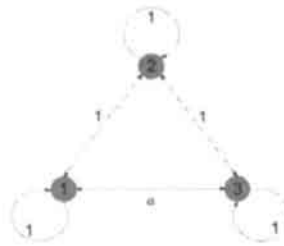

Figure 1.1: High Multiplicity TSP: Every city has to be visited twice

three cities 1, 2, and 3 with distances as on the figure, it is required to find a minimum length tour that visits every city twice. Is the optimal tour twice longer than the optimal tour in the single multiplicity version? It may well be the case that the length of the optimal tour increases by a factor of less than two. Indeed, the length of the optimal tour, take, for instance, (1,2,3,3,2,1), in the high multiplicity version is 6, but the length of the optimal tour (1,2,3) in the single multiplicity version is $a + 2$, which can be even bigger than 6 (take $a > 4$). Likewise, one may ask what happens if the multiplicities are multiplied by a multiplicator l. Is there a finite value l for which the tour length divided by l reaches the overall minimum value? Is this finite value polynomially bounded? Can it be efficiently computed? Let us illustrate this issue also on the example of INTEGER KNAPSACK.

Example 1.3.1 (INTEGER KNAPSACK – continued).

The problem at hand can be seen as follows. Assume that during a month we have to complete some number of jobs and we are trying to maximize the importance of jobs completed in time. On the other hand, if this situation

repeats many months in a row, for example, during a couple of years, we can be interested in the following questions. If we multiply the common due date by a factor l (assuming that the number of jobs to be completed is multiplied by l also), does the importance of the average number of not late jobs increase? The problem can be modelled by the following parametric integer linear program (IK(l)):

$$F(l) = \max_{x} \frac{1}{l} \sum_{j \in J} w_j x_j \tag{1.9}$$

subject to

$$\sum_{j \in J} p_j x_j \leq l \times D; \tag{1.10}$$

$$0 \leq x_j \leq l \times n_j, \quad j \in J; \tag{1.11}$$

$$x_j \in \mathbb{Z}^+, \quad j \in J, \tag{1.12}$$

and we are interested in the analysis of the function $F : \mathbb{N} \to \mathbb{R}^+$. In particular, we are looking for a multiplicator $l^* \in \mathbb{N}$ that maximizes $F(l)$ over all natural numbers.

For INTEGER KNAPSACK the multiplicator l^* always exists. To see this, we notice that the linear relaxations of IK and IK(l) are identical for any $l \in \mathbb{N}$. The optimal objective value of the linear relaxation of IK is an upper bound on the optimal objective value of IK(l) for any $l \in \mathbb{N}$. Now, given a fractional optimal solution of the linear relaxation of IK (we assume that all input data of INTEGER KNAPSACK are integer, and therefore the fractional optimal solution is rational) we can find the common denominator of the fractions in this solution. Let the common denominator be h. Then the numerators of the solution form an integer optimal solution for the problem IK(h) and also for the linear relaxation of IK(h). Hence, $l^* = h$ is required multiplicator. □

In Chapter 3 we investigate the existence of such a multiplicator in the high multiplicity TSP. In contrast to INTEGER KNAPSACK, in TSP the finite optimal multiplicator does not necessarily exist. We present some techniques to analyze the relations between multiplicators l and corresponding optimal solutions of high multiplicity TSP. We hope it serves as a starting point for related high multiplicity studies on combinatorial optimization problems.

Another direction in which this research is extended is in the direction of cyclic scheduling, in which the multiplicity is more implicit than in thus far encountered models.

A general problem description for a set of high multiplicity scheduling problems defines relative quantities in which the jobs of various classes are to be produced. In the literature on repetitive manufacturing [48, 49, 54, 55, 84] such sets are defined as minimal part set. One view of the minimal part set is to see them as the minimal production quantities that yield the desired output ratios, and that repeating sequences in which the parts in a minimal part set are produced is desirable for various reasons. Among these reasons are a balanced demand on sub assemblies and low levels of end item inventory. Alternatively, one may argue that repeating these shortest possible cycles results in a loss in throughput: higher throughput rates may be attainable when allowing more general production sequences. Notice how this resembles the discussion on the high multiplicity TSP. In fact, the analysis on TSP is motivated by its role as a basic sequencing problem that is applicable to a variety of machine scheduling problems, among which NO-WAIT FLOW SHOP scheduling.

High multiplicity scheduling problems in which the problem input can be interpreted in terms of such relative ratios appear also in other applications [7, 68]. A first issue in these problem is to determine the optimal objective function value which minimizes some long run performance measure, such as throughput rate. It is then not clear that the minimum is at all attainable by a finite solution, and hence membership in NP is again a nontrivial issue. For the maintenance scheduling problem in Chapter 4 this complexity has long been completely open, and the complexity of some of its variants is still open. For example, even when the minimum objective function value is attainable by a finite sequence, it remains open to find it. Moreover, if determining (the length of) an optimal sequence is known to be in P or NP, this is not necessarily the case for a sequence of a prespecified length, and vice versa. Chapter 3 deals with many of these problems, and Chapter 4 explores such problems from a slightly different angle.

In Chapter 4 we deal with a problem in which there are costs associated with scheduling and not scheduling a job of each type at a certain time moment, and these costs are dependent on the time interval elapsed since a job of the type was scheduled last. The objective is now to minimize the long run average costs [7], or the cost for a time period of given length. To illustrate the problem consider Example 4.1.1 from Chapter 4. Given

are three operating machines 1, 2, and 3. Let the jobs be the maintenance services of these machines.

Day:	M	T	W	T	F	S	S	Total
Maintenance schedule (machines):	1	2	1	2	1	2	3	
Operating cost on machine 1:	0	10	0	10	0	10	20	50
Operating cost on machine 2:	20	0	10	0	10	0	10	50
Operating cost on machine 3:	1	2	3	4	5	6	0	21
Maintenance cost on machine 3:	1	0	1	0	1	0	0	3
Maintenance cost on machine 3:	0	1	0	1	0	1	0	3
Maintenance cost on machine 3:	0	0	0	0	0	0	1	1
Total operating and maintenance costs:	22	13	14	15	16	17	31	128

Table 1.1: An optimal maintenance schedule

We assume that the processing time of a job has unit length. If a machine is maintained some day the operating cost of that machine decreases to zero, and contrary if a machine is not maintained then the operating cost of that machine increases by some additive factor. Let the additive factors for machine 1 and 2 be 10, and for machine 3 it is 1. Let the maintenance cost for any machine be 1. It is required to find a cyclic maintenance schedule for a week that minimizes the total operating and maintenance costs.

Here, the relative ratios (frequencies or number of maintenances) in which the machines are maintained are not specified in the input, but determining them is part of the problem. Notice how we thus introduce a high multiplicity problem for which there is no direct single multiplicity counterpart. Research on this and closely related problems has thus far naturally focused on complexity and approximation issues. In Chapter 4 we present the first thorough study on exact solution methods for this problem, which is shown to be NP-hard by Bar-Noy et al [9]. This, and several related problems (see also Chapter 5) can be modelled by a SET PARTITIONING formulation based on a time indexed formulation, see, e.g., [5]. An approach that is taken by other authors as well, see, for instance, [10]. Not surprisingly the high multiplicity results in an "avalanche" of symmetry when solving the problems using branch-and-price methods, and we propose and investigate

several techniques to reduce the running time, and consider other mathematical programming formulations as well.

Chapter 5 deals with many of the aspects discussed so far, in the context of supply chain scheduling problems. We investigate scheduling problems that arise in the following materials requirements planning (MRP) based context, see, e.g., [8]. At a certain moment in time a manufacturer enters his booked and/or forecast demand, and executes an MRP run to generate the material requirements for this demand. Usually, since materials have a certain lead time, this is performed some time in advance, such that the situation at this moment of materials requirements planning is different from the actual situation that materializes when assembling to order the customer orders. The difference may be due to forecast inaccuracy, but also to changes in customer orders, or even to changes in delivery orders. This situation is frequently encountered in practice [70], and studied, for example, by Gouw [39] and Holthuijsen [60].

Multiplicity arises naturally in such problems, since usually customer orders and delivery orders contain multiple items. For illustration take Example 5.2.2 from Chapter 5. Assume that there are two types of end items and one type of row material required to produce end items. Customer orders consists of 3 items of type 1 and 5 items of type 2. To produce one item of type 1 we need 3 units of row material and to produce 1 item of type 2 – 2 units. Production of one item of any type takes unit processing time. It is required to find a production schedule which is feasible with respect to a material supply and that minimizes the maximum completion time over all demanded items. In the figure we present the supply schedule of the row material and the optimal production schedule.

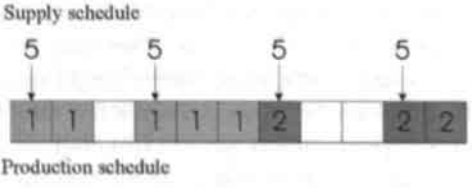

Figure 1.2: Supply and production schedules

In Chapter 5 we present several complexity results such as NP-completeness results, and polynomial algorithms for several classes of such problems. Chapter 5 also investigates scheduling problems and issues as encountered in Chap-

ter 2 - about the representation of a solution -, Chapter 3 - about polynomial encodings of solutions and repetitions of sequences - and Chapter 4, regarding exact solution methods. In addition, we explore other implicit forms of multiplicity where, for example, the demand and supply patterns are given by functions rather than by due dates and delivery dates. This leads again to polynomially solvable problems, and very hard (undecidable) ones, and relates to the analysis presented in Chapter 6.

1.4 How to read this thesis

The subsequent chapters of this thesis are based on articles. Chapter 2 is based on [13] presented on the Fifth Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP'2001). Chapter 3 is based on [43] presented on the 17th International Symposium on mathematical Programming (ISMP'2000). Chapter 4 is based on working paper [44]. Chapter 6 is based on [46] presented on 13th International Symposium on Algorithms and Computation (ISAAC'2002).

We have chosen to present the Chapters in such a way that they can be read independently of one another and of this introductory chapter. We expect from a reader some basic knowledge of machine scheduling terminology, for example, understanding the $\alpha|\beta|\gamma$ notation by Graham et al, see [40] or [19].

Chapter 2

On the complexity of high multiplicity scheduling problems

2.1 Introduction

The purpose of this chapter is to propose a definition of several complexity classes which may appear useful for the analysis of so-called *high multiplicity scheduling problems*. Such problems have been recently investigated by several researchers (see, e.g., [23, 95, 96], and [99] for early references, and other articles cited below for more recent ones). Hochbaum and Shamir [57, 58], in particular, have coined the term "high multiplicity" and have underlined the need of discussing the complexity of such problems with special care. A paper by Clifford and Posner [22] provides a more detailed framework for this complexity analysis, as well as applications to several specific problems.

We take a further step along this line of research, by formulating several proposals to cast high multiplicity scheduling problems into a more precise computational complexity framework. Sections 2.2 and 2.3 describe the class of scheduling problems that we want to consider. Section 2.4 contains a typology of algorithms which is applicable in the simplest case of non-preemptive one-machine scheduling problems. These concepts are illustrated on specific examples in Section 2.5. The typology is extended to more general scheduling problems (involving multiple machines and job preemptions) in Section 2.6, and further illustrated in Section 2.7. The chapter concludes with a brief discussion and an outline of perspectives for further research.

2.2 High multiplicity scheduling problems

For the sake of simplicity, we initially restrict ourselves to non-preemptive one-machine scheduling problems. More complex problem formulations will be tackled in Section 2.6.

The input of a classical scheduling problem $\mathcal{SP}$ consists of a list of n jobs, together with a list of attributes of each job. The attributes of job j ($j = 1, 2, \ldots, n$) typically include its processing time p_j, its release date r_j, its due date d_j, etc. The binary input size of an instance of $\mathcal{SP}$ is $O(nL)$, where L is the largest input size of an attribute.

It frequently happens, however, that the input of a scheduling problem can be described in a much more compact way, due to the fact that the jobs naturally fall into a small number, say $s << n$, of distinct *job types*, where all the jobs of a same type share exactly the same characteristics, i.e., attribute values. When this is the case, we only need to describe *one* representative job in order to completely define a type, so that an instance of the problem $\mathcal{SP}$ only consists of the following data:

- the number of job types, viz. s;
- for each job type $i = 1, 2, \ldots, s$, the number of jobs of type i, viz. n_i;
- for each job type $i = 1, 2, \ldots, s$, the attributes of a representative job of type i.

When the data is encoded in this compact form, we say that $\mathcal{SP}$ is a *high multiplicity* scheduling problem. So, a generic instance of the (one-machine non-preemptive) high multiplicity scheduling problem $\mathcal{SP}$ takes the form $D = (s, n_1, n_2, \ldots, n_s, \Delta)$, where Δ comprises all the relevant job attributes.

This kind of situation is encountered, for instance, in repetitive manufacturing environments. Here, a *minimal part set* (MPS) is described by a vector $(n_1, n_2, \ldots, n_s)$ where n_i represents the number of parts of type i in the MPS ($1 \leq i \leq s$). The whole part set is then viewed as consisting of a large number of copies of the MPS (in the limit, infinitely many copies) to be produced repeatedly (see, e.g., [89], and [94]).

In other applications, the number of job types may be artificially reduced by aggregating jobs with different, but similar characteristics, into a single type. The resulting scheduling problem is only an approximation of the original one, but may prove easier to handle [59].

If we denote by $|D|$ the input size of an instance D of a high multiplicity scheduling problem, then $|D| = O(\sum_{1\le i\le s} \log n_i + sL) = O(s \log n + sL)$, where L is again the largest input size of an attribute value and $n = \sum_{1\le i\le s} n_i$. Typically, this input size is much smaller than nL, as is, e.g., the case when s is viewed as a constant. More precisely, we say that $\mathcal{SP}$ is a *high multiplicity scheduling problem* if n is not polynomially bounded in the input size of the problem, i.e., if there is no constant k such that $n = O\left((sL)^k\right)$ for all instances of $\mathcal{SP}$. Thus, an algorithm for $\mathcal{SP}$ whose complexity is polynomial in s, L *and* n is only *pseudo-polynomial*, but not polynomial in the input size. This distinction has been drawn by several authors and stressed especially by Hochbaum et al. [57, 58, 59], and by Clifford and Posner [22].

In order to discuss it more precisely, we need a formal definition of the problems we are dealing with.

2.3 A scheduling problem is three problems

Consider an instance $D = (s, n_1, n_2, \ldots, n_s, \Delta)$ of a (one-machine non-preemptive) high multiplicity scheduling problem $\mathcal{SP}$. We assume without loss of generality that all the entries of D are integral, and that the jobs are numbered from 1 to n in such a way that jobs 1 to n_1 are of type 1, jobs $n_1 + 1$ to $n_1 + n_2$ are of type 2, etc.

A schedule for the instance D can be seen as an assignment $S^* : \{1, 2, \ldots, n\} \to \mathbb{R}$ where $S^*(j)$ denotes the starting time of job j $(j = 1, 2, \ldots, n)$. However, in order to be able to handle more general problem formulations (as those in Section 2.6), we prefer to define a *schedule* for an instance D as a (finite) *subset* S of $\{1, 2, \ldots, n\} \times \mathbb{R} \times \mathbb{R}$, where S may be (and usually, is) restricted to belong to a set $\mathcal{F}_D$ of *feasible* schedules associated with D. The interpretation of S is that, if $(j, t_1, t_2) \in S$, then job j is processed continuously during the time interval $[t_1, t_2]$. This model allows to recover more traditional readings of the schedule by considering various *mappings* derived from S, for instance:

- a *job-oriented* description of the schedule is defined by the mapping

$$S_J : \{1, 2, \ldots, n\} \to \mathbb{R} \times \mathbb{R} : j \mapsto S_J(j) = (t_1, t_2) \text{ if } (j, t_1, t_2) \in S;$$

- a *time-oriented* description of the schedule is defined by the mapping

$$S_T\colon \mathbb{R} \to \{0,1,\ldots,n\}\times\mathbb{R}\times\mathbb{R} : t \mapsto S_T(t) = \begin{cases} (j,t_1,t_2) & \text{if } (j,t_1,t_2)\in S \text{ and job } j \text{ is the first job to be processed at or after time } t; \\ (0,0,0) & \text{if there are no more jobs to be processed after time } t. \end{cases}$$

Remark 2.3.1. *In Chapter 5 dealing with single machine scheduling problems we consider also the third type of schedule descriptions:*

- *a sequence-oriented description of the schedule is defined by the mapping*

$$S_S\colon \{0,1,\ldots,n\} \to \{0,1,\ldots,n\} : j \mapsto S_S(j) = k \quad \textit{if job } j \textit{ is the } k\textit{-th job to be processed on the machine.}$$

Since the sequence-oriented description does not depend on time at all, it is less informative than the job-oriented or the time-oriented description. On the other hand, in Chapter 5 we will see that for some problems we can easily construct a polynomial time and polynomial space sequence-oriented description, but the existence of polynomial time and polynomial space job-oriented and time-oriented description remain the open questions.

In the one-machine non-preemptive context, the job-oriented description S_J is roughly equivalent to the full description of S, as well as to the mapping S^* mentioned earlier. The time-oriented description corresponds to the viewpoint of a human machine-operator, who must know, at every instant, which job is being processed or which job will be processed next on his or her machine.

Of course, still numerous other mappings could be associated with the schedule S. For instance, a mapping from time instants to *job types*, rather than to individual jobs, may prove useful in some frameworks. But in this discussion, we shall restrict ourselves to the above-mentioned mappings.

Let us now turn to the objective function of $\mathcal{SP}$. If $\mathcal{F}_D$ denotes again the set of feasible schedules associated with D, we let $f_D\colon \mathcal{F}_D \to \mathbb{R}$ be the cost function to be minimized over $\mathcal{F}_D$ (for instance, $f_D(S)$ could measure the

makespan or the weighted tardiness of schedule S). For the sake of simplicity, we assume that $\mathcal{F}_D$ is non-empty for every D, and that f_D always attains its minimum over $\mathcal{F}_D$. Moreover, we also assume that, given a description of S *in extension* (i.e., given a list of the elements of S), $f_D(S)$ can be computed in time polynomial in $|D|$ and $|S|$ (note that this is a rather weak hypothesis).

As in [93] (from where we have borrowed the title of this section), we now define three distinct scheduling problems associated with $\mathcal{F}_D$ and f_D (see also [22]).

RECOGNITION PROBLEM $\mathcal{SP}_1$:
INSTANCE: $D = (s, n_1, n_2, \ldots, n_s, \Delta)$ and $K \in \mathbb{R}$.
OUTPUT: Yes if there is a schedule $S \in \mathcal{F}_D$ with $f_D(S) \leq K$. No otherwise.

EVALUATION PROBLEM $\mathcal{SP}_2$:
INSTANCE: $D = (s, n_1, n_2, \ldots, n_s, \Delta)$.
OUTPUT: The minimum value of f_D over $\mathcal{F}_D$.

OPTIMIZATION PROBLEM $\mathcal{SP}_3$:
INSTANCE: $D = (s, n_1, n_2, \ldots, n_s, \Delta)$.
OUTPUT: A schedule $S \in \mathcal{F}_D$ which minimizes $f_D(S)$ over $\mathcal{F}_D$.

Issues related to the complexity classification of $\mathcal{SP}_1$ or $\mathcal{SP}_2$ fall within the traditional scope of complexity analysis, as discussed, e.g., by Garey, and Johnson in [34] or Papadimitriou, and Steiglitz in [93]. However, analyzing the complexity of any specific high multiplicity problem may turn out to be a tricky matter. Indeed, proving that $\mathcal{SP}_1$ is in NP, for instance, requires the existence of an algorithm $\mathcal{A}$ and of a polynomial-size *certificate* $c(D, K)$ for each Yes-instance (D, K) of $\mathcal{SP}_1$, with the property that, when applied to $c(D, K)$, $\mathcal{A}$ returns the answer Yes after a polynomial number of steps (we use the terminology of Papadimitriou, and Steiglitz[93]). Intuitively, when the answer to $\mathcal{SP}_1$ is affirmative, the certificate provides a concise proof that it is indeed so. Now, the most natural certificate for problem $\mathcal{SP}_1$ would be a feasible schedule S such that $f_D(S) \leq K$. But in many cases, obvious descriptions of S are not concise, i.e., not polynomial in the size of (D, K). (As a matter of fact, it may even be the case that some high multiplicity recognition problems are neither in NP nor in co-NP.) Similar problems pop up, of course, if we are to prove that $\mathcal{SP}_1$ or $\mathcal{SP}_2$ is in P.

In spite of these difficulties, many high multiplicity scheduling problems have been proved to be polynomially solvable (see, for instance, [14, 21, 22,

41, 57, 58, 59, 61, 85], etc.) or in co-NP ([12]) or NP-hard [9, 21, 22, 95], etc. Such results (and other similar results found in the literature) can be established by displaying (optimality or feasibility) certificates whose size is polynomial in the input length $O(s \log n + sL)$. The certificates, clearly, do not enumerate the list of n starting times, but rather provide an implicit, concise encoding of these starting times. Let us illustrate this on a problem solved in [58].

Example 2.3.2. (Weighted number of tardy jobs). *In the scheduling three-field $(\alpha|\beta|\gamma)$ notation, see [40], this is the problem $1|p_j = 1|\sum_j w_j U_j$. Its input takes the form*

$$D = (s, n_1, n_2, \ldots, n_s, d_1, d_2, \ldots, d_s, w_1, w_2, \ldots, w_s),$$

where d_i is the due-date for jobs of type i and w_i is their weight. All jobs are assumed to have unit-processing time. The objective function is to minimize the weighted number of tardy jobs. Hochbaum and Shamir proved that the problem can be transformed into a transportation problem with constraint matrix of size $s \times (s+1)$, where variable $x_{i,t}$ indicates the number of jobs of type i processed in the interval $(d_{t-1}, d_t]$, for $i = 1, 2, \ldots, s$, $t = 1, 2, \ldots, s+1$ (without loss of generality, $d_0 = 0 \leq d_1 \leq \ldots \leq d_s \leq d_{s+1} = n$). The variables $x_{i,t}$ must satisfy the transportation constraints

$$\sum_{i=1}^{s} x_{i,t} = d_t - d_{t-1}, \qquad t = 1, 2, \ldots, s+1;$$

$$\sum_{t=1}^{s+1} x_{i,t} = n_i, \qquad i = 1, 2, \ldots, s.$$

Because the transportation problem is polynomially solvable, the recognition and the evaluation version of $1|p_j = 1|\sum_j w_j U_j$ can be solved in (strongly) polynomial time (in fact, in $O(s \log s)$ time if a specialized greedy algorithm is used). □

Let us now turn to the optimization problem $\mathcal{SP}_3$. Few authors have attempted to discuss precisely what it means to "solve" $\mathcal{SP}_3$. Namely, the way in which the optimal schedule S should be described, is usually not explicitly stated.

Hochbaum et al. in [58] and [59] have observed that $\mathcal{SP}_3$ can sometimes be solved by first obtaining a concise encoding of the optimal schedule, then

applying a decoding algorithm to generate all the elements of the schedule. For instance, in Example 2.3.2 above, the solution $(x_{i,t})$ of the transportation problem provides a concise encoding of the solution. In order to obtain a schedule in extension, i.e., in order to compute a starting time for each job, one needs to "decode" the solution $(x_{i,t})$ by carrying out additional computations (see Section 2.5).

Clifford and Posner in [22] established a clear distinction between the recognition version and the optimization version of high multiplicity problems. They noticed: "Under high multiplicity encoding, we cannot output a job-by-job, machine-by-machine schedule if we want the output length to be a polynomial function of the input length. Consequently, we allow for a high multiplicity description of the schedule." Clifford and Posner set out to define a *job group* as an arbitrary subset of jobs of a particular type and describe a high multiplicity schedule as an ordered set of job groups (their definition is actually complicated by the fact that they consider multiple machines). In [22], only this type of schedule is regarded as "legal". Under this hypothesis, the authors argue that certain high multiplicity problems are in $EXP \setminus P$, meaning that such problems are solvable in exponential time, but not in polynomial time. More precisely, they establish that there is no polynomial description of the optimal schedule in terms of job groups.

This view of high multiplicity schedules, however, may appear to be too restrictive. In the next section, we propose more general models for describing a solution of problem $\mathcal{SP}_3$. An advantage of these alternate models is that they allow to obtain a more precise complexity classification of algorithms solving $\mathcal{SP}_3$.

2.4 Complexity models

In this section, we shall rely on several interpretations of the task "output an optimal schedule S". A main distinction takes place according as we view schedules as sets, or as we focus on one of their derived (single-valued) mappings.

2.4.1 List-generating algorithms

In our first interpretation, we assume that the set S is to be generated in extension.

Definition 2.4.1. *An algorithm $\mathcal{A}$ is a list-generating algorithm for $\mathcal{SP}_3$ if, for every instance of $\mathcal{SP}_3$, $\mathcal{A}$ successively outputs the elements (π^1, t_1^1, t_2^1), (π^2, t_1^2, t_2^2), $\ldots$, (π^n, t_1^n, t_2^n) of an optimal schedule S, where $(\pi^1, \pi^2, \ldots, \pi^n)$ is permutation of the elements of job-set.*

For a list-generating algorithm $\mathcal{A}$, we let $\tau(0) = 0$ and for $j = 1, 2, \ldots, n$, we denote by $\tau(j)$ the running time required by $\mathcal{A}$ in order to output the first j elements of the schedule, i.e., $(\pi^1, t_1^1, t_2^1), (\pi^2, t_1^2, t_2^2), \ldots, (\pi^j, t_1^j, t_2^j)$. So, $\tau(n)$ is the total running time of $\mathcal{A}$, and $\tau(j) - \tau(j-1)$ is the time elapsed between the $(j-1)$-st and the j-th outputs.

The classification of list-generating algorithms to be described in Definition 2.4.2 is based on a proposal due to [64], for problems in which the size of the output may be exponentially larger than the size of the input (such as, for instance, the problem of listing all maximal independent sets of a graph, or all vertices of a polyhedron; see also [79] or [29] for related concepts).

Definition 2.4.2. *A list-generating algorithm $\mathcal{A}$ for $\mathcal{SP}_3$ runs in:*

- *pseudo-polynomial time if $\tau(n)$ is polynomially bounded in $|D|$ and M, where M is the largest number appearing in D;*
- *polynomial total time if $\tau(n)$ is polynomially bounded in n and $|D|$;*
- *incremental polynomial time if $\tau(j) - \tau(j-1)$ is polynomially bounded in j and $|D|$, for $j = 1, 2, \ldots, n$;*
- *polynomial delay if $\tau(j) - \tau(j-1)$ is polynomially bounded in $|D|$, for $j = 1, 2, \ldots, n$;*
- *polynomial time if $\tau(n)$ is polynomially bounded in $|D|$.*

These definitions will be illustrated on several scheduling problems in Section 2.5. For now, let us place a few comments on the above definitions.

Pseudo-polynomiality and polynomiality are usual concepts from complexity theory and are only mentioned here for the sake of completeness. In particular, if $\mathcal{SP}_3$ can be solved in polynomial time, then n is bounded by a polynomial in $|D|$ for all instances of this problem, and the problem does not qualify as a high multiplicity problem. On the other hand, if $\mathcal{SP}_3$ can be solved in pseudo-polynomial time, then the same complexity holds for $\mathcal{SP}_1$

and $\mathcal{SP}_2$ (since we assumed that $f_D(S)$ can be computed in time polynomial in $|D|$ and $|S|$).

Total time polynomiality is, in a sense, the weakest notion of polynomiality which can be applied to $\mathcal{SP}_3$, since the running time of any algorithm which lists the starting times of all n jobs must grow at least linearly with n.

Incremental polynomial time captures the idea that the algorithm outputs the starting times sequentially and does not spend "too much time" between two successive outputs. In computing the starting time of job π^j, however, the algorithm may need to look at the starting times of $\pi^1, \pi^2, \ldots, \pi^{j-1}$ (for instance, to check feasibility of the partial schedule) and therefore we allow $\tau(j) - \tau(j-1)$ to depend on j as well as on $|D|$.

Remark 2.4.3. *The intuition behind this concept is slightly different from that in* [64]. *A more straightforward adaptation of the definition proposed by Johnson et al. would go something like this: given* any *subset of jobs, say* $J \subset \{1, 2, \ldots, n\}$, *as well as the starting times of all the jobs in* J, *the algorithm should be able to output a job* $k \notin J$ *and its processing interval* $[t_1^k, t_2^k]$ *in time polynomially bounded in* $|J|$ *and* $|D|$. *This notion is stronger than the one we introduced, but we do not feel that it presents much interest in the scheduling context.* □

Remark 2.4.4. *Johnson et al in* [64] *also consider the case where* $\mathcal{A}$ *is required to produce its outputs in some specified order. Such a requirement may easily be adapted for scheduling problems, see Chapter 5. For instance, since schedules are to be implemented in the course of time, it may appear "natural" to ask for* $\mathcal{A}$ *to output* $S_J(1)$, $S_J(2)$,...,$S_J(n)$ *in increasing order of the starting times. Whether such requirements actually are meaningful or not may be debatable. At any rate, the corresponding algorithms can still be analyzed in terms of the above classification.* □

Finally, an algorithm runs with polynomial delay when the time elapsed between two successive outputs is polynomial in the input size of the problem. This is a rather strong requirement, the strongest, in fact, among those discussed in [64]. We also feel that it is one of the most meaningful requirements that may apply to algorithms for high multiplicity scheduling problems.

The following statement summarizes the above discussion.

Proposition 2.4.5. *If* $\mathcal{A}$ *is a list-generating algorithm for the optimization version* $\mathcal{SP}_3$ *of a single-machine scheduling problem without preemptions,*

then:

$$\begin{aligned} \mathcal{A} \textit{ runs in polynomial time} &\Longrightarrow \mathcal{A} \textit{ runs with polynomial delay} \\ &\Longrightarrow \mathcal{A} \textit{ runs in incremental polynomial time} \\ &\Longrightarrow \mathcal{A} \textit{ runs in polynomial total time} \\ &\Longrightarrow \mathcal{A} \textit{ runs in pseudo-polynomial time.} \end{aligned}$$

Proof. All the implications are easy. For instance, if $\mathcal{A}$ runs in incremental polynomial time, then the whole schedule can be generated in time $\tau(n) = \sum_{j=1}^{n} [\tau(j) - \tau(j-1)]$, which is polynomial in n and $|D|$. Hence, $\mathcal{A}$ runs in polynomial total time.

Moreover, if $\mathcal{A}$ runs in polynomial total time, then $\mathcal{A}$ also runs in pseudo-polynomial time, since $n = \sum_{1 \le j \le s} n_j \le M^2$ by definition of M. □

2.4.2 Pointwise job-oriented algorithms

In this and the next section, we assume that algorithm $\mathcal{A}$ is not necessarily required to produce the optimal schedule in extension, but that it should only be able to compute one of the mappings derived from S as explained in Section 2.3.

Definition 2.4.6. *A pointwise job-oriented algorithm for $\mathcal{SP}_3$ is an algorithm $\mathcal{A}$ which, on every input (D, j) $(j \in \{1, 2, \ldots, n\})$, outputs a pair $S_J(j) = \big(t_1(j), t_2(j)\big)$ such that*

$$S = \{ \big(j, t_1(j), t_2(j)\big) : j \in \{1, 2, \ldots, n\} \}$$

is an optimal schedule for D.

As usual, a pointwise job-oriented algorithm for $\mathcal{SP}_3$ is polynomial if, for every instance D of $\mathcal{SP}_3$ and every $j \in \{1, 2, \ldots, n\}$, $\mathcal{A}$ outputs $S_J(j)$ in time polynomially bounded in $|D|$. Thus, the existence of a polynomial pointwise algorithm $\mathcal{A}$ simply means that the job-oriented description of the optimal schedule can be queried in polynomial time or, in other words, that the function $S_J \colon \{1, 2, \ldots, n\} \to \mathbb{R} \times \mathbb{R}$ can be computed in polynomial time (in the sense of [34]).

The following relations hold.

Proposition 2.4.7. *For a single-machine scheduling problem without preemptions,*
(a) *if $\mathcal{SP}_3$ has a polynomial list-generating algorithm, then $\mathcal{SP}_3$ has a polynomial pointwise job-oriented algorithm;*
(b) *if $\mathcal{SP}_3$ has a polynomial pointwise job-oriented algorithm, then $\mathcal{SP}_3$ has a polynomial-delay list-generating algorithm.*

Proof. Assertion (a) holds trivially, since all the elements of an optimal schedule can be generated in polynomial time when a polynomial list-generating algorithm is available.

Conversely, if $\mathcal{A}$ is a polynomial pointwise job-oriented algorithm, then $\mathcal{A}$ can be called n times to compute successively $S_J(1), S_J(2), \ldots, S_J(n)$. Since the running time of each call is polynomial in $|D|$, the resulting list-generating algorithm runs with polynomial delay. □

So, intuitively, polynomial pointwise job-oriented algorithms fall somewhere between polynomial and polynomial-delay list-generating algorithms in the hierarchy described in Proposition 2.4.5.

We shall present some examples of polynomial pointwise algorithms in Section 2.5. Interestingly, all such algorithms actually consist of two distinct algorithms: a first algorithm $\mathcal{A}_e$ which solves $\mathcal{SP}_2$ while producing a compact "encoding" ("certificate") Σ of the optimal schedule S, and a second algorithm $\mathcal{A}_u$ which computes S_J by "decoding" the output produced by $\mathcal{A}_e$. Let us formulate this notion in more precise terms.

Definition 2.4.8. *A 2-phase job-oriented algorithm for $\mathcal{SP}_3$ is a pair of algorithms $(\mathcal{A}_e, \mathcal{A}_u)$ such that*
1) on the input D, $\mathcal{A}_e$ outputs a string (f_D^{opt}, Σ) where f_D^{opt} is the optimal objective value of $\mathcal{SP}_3$;
2) on the input $(D, j, f_D^{opt}, \Sigma)$, $\mathcal{A}_u$ outputs a pair $S_J(j) = \big(t_1(j), t_2(j)\big)$ such that

$$S = \{ \big(j, t_1(j), t_2(j)\big) : j \in \{1, 2, \ldots, n\} \}$$

is an optimal schedule for D.

The string Σ in this definition represents the encoding of the optimal solution.

Now, a 2-phase job-oriented algorithm runs in polynomial time if both $\mathcal{A}_e$ and $\mathcal{A}_u$ run in time polynomial in the size of their respective inputs (this implies, in particular, that the size of Σ must be polynomially related to the

size of D). Note that these requirements are stronger than the requirements for a polynomial pointwise job-oriented algorithm: indeed, for a polynomial pointwise job-oriented algorithm to be polynomial, it need not compute the optimal objective value f_D^{opt} in time polynomial in $|D|$. In fact, for such an algorithm, the only requirement on the computation of the objective function value comes from our blanket assumption that, given a description of S in extension, the corresponding objective function value can be computed in time polynomial in $|D|$ and $|S|$ (cf. Section 2.3). From a complexity viewpoint, the following problem captures the difference between polynomial 2-phase job-oriented algorithms and polynomial pointwise job-oriented algorithms:

3-SAT SCHEDULING:
INSTANCE: $D = (2^N, C)$, where C is a 3-SATISFIABILITY (3-SAT) instance on N variables, and there are $n = 2^N$ jobs of the same type.
SCHEDULES: There is only one feasible schedule, namely $S_J = \{(j, j, j+1) : j \in \{1, 2, \ldots, 2^N\}\}$.
OBJECTIVE FUNCTION: $f_D = 1$ if C is satisfiable, $f_D = 0$ otherwise.

The reader may verify that all of our blanket assumptions concerning high multiplicity problems hold for 3-SAT SCHEDULING; in particular, the optimal value of the schedule can be computed in time polynomial in the length of the schedule, viz. 2^N.

Proposition 2.4.9. *Problem* 3-SAT SCHEDULING *has a polynomial pointwise job-oriented algorithm, but it has no polynomial 2-phase job-oriented algorithm unless* $P = NP$.

Proof. For 3-SAT SCHEDULING, the mapping S_J can trivially be computed in polynomial time: given (D, j), just return $(j, j+1)$. Hence, the problem admits a polynomial pointwise job-oriented algorithm. But computing the optimal value of the problem (i.e., solving $\mathcal{SP}_2$) is NP-hard, since this amounts to solving the 3-SAT problem in polynomial time. Since a 2-phase job-oriented algorithm requires an algorithm $\mathcal{A}_e$ which outputs a string (f_D^{opt}, Σ) where f_D^{opt} is the optimal objective value of $\mathcal{SP}_3$, such an algorithm cannot be polynomial unless $P = NP$. □

The following proposition identifies the conditions under which the existence of a polynomial 2-phase job-oriented algorithm is equivalent to the existence of a pointwise job-oriented algorithm.

Proposition 2.4.10. *The optimization version $\mathcal{SP}_3$ of a single-machine scheduling problem without preemptions admits a polynomial 2-phase job-oriented algorithm if and only if it has a polynomial pointwise job-oriented algorithm and the corresponding evaluation problem $\mathcal{SP}_2$ is polynomially solvable.*

Proof. If $\mathcal{SP}_3$ has a polynomial 2-phase job-oriented algorithm $(\mathcal{A}_e, \mathcal{A}_u)$, then a polynomial pointwise job-oriented algorithm is obtained as follows. When handed the input (D, j) with $j \geq 1$, the algorithm first runs $\mathcal{A}_e$ on D to obtain (f_D^{opt}, Σ), then runs $\mathcal{A}_u$ on $(D, j, f_D^{opt}, \Sigma)$ to compute $S_J(j)$. Note that the total running time of this procedure is polynomial in $|D|$. Moreover, $\mathcal{A}_e$ solves $\mathcal{SP}_2$ in polynomial time.

Conversely, let $\mathcal{A}^*$ be an algorithm that solves $\mathcal{SP}_2$ in polynomial time, then identify $\mathcal{A}_e$ with $\mathcal{A}^*$. The output of $\mathcal{A}^*$ is simply f_D^{opt}, thus the string Σ is empty. If $\mathcal{SP}_3$ has a polynomial pointwise job-oriented algorithm $\mathcal{A}$ then the decoding algorithm $\mathcal{A}_u$ can be identified with $\mathcal{A}$: when running on the input (D, j, f_D^{opt}), the algorithm simply ignores the information f_D^{opt}. □

By definition, if $\mathcal{SP}_2$ is polynomially solvable, then f_D^{opt} can be computed in time polynomial in $|D|$. This property is clearly different from our general assumption for high multiplicity scheduling problems, namely that given a description of S in extension, $f_D(S)$ can be computed in time polynomial in $|D|$ and $|S|$. This difference is responsible for the two different results stated in Proposition 2.4.10 and Proposition 2.4.9 above.

Remark 2.4.11. *The reader may note that there remains some room for an intermediate result between Proposition 2.4.10 and Proposition 2.4.9. Indeed, it is open whether polynomial 2-phase job-oriented algorithms are equivalent to polynomial pointwise job-oriented algorithms under the following condition: given a description of an optimal schedule S, $f_D(S)$ can be computed in time polynomial in D. This condition implicitly requires that the description of an optimal schedule is not the extensive list of jobs, since such a list is by definition of superpolynomial length. However, it does not require that the optimal schedule S be found in time polynomial in D, nor does it require that $\mathcal{SP}_2$ be polynomially solvable.* □

2.4.3 Pointwise time-oriented algorithms

Of course, a definition similar to Definition 3 can be proposed for time-oriented descriptions of the optimal schedule.

Definition 2.4.12. *A pointwise time-oriented algorithm for $\mathcal{SP}_3$ is an algorithm $\mathcal{A}$ which, on the input (D,t) with $t \in \mathbb{R}$, outputs $S_T(t)$, where S_T is the time-oriented mapping derived from an optimal schedule S.*

Similarly to Proposition 2.4.7, the following relations hold.

Proposition 2.4.13. *For a single-machine scheduling problem without preemptions,*
(a) *if $\mathcal{SP}_3$ has a polynomial list-generating algorithm, then $\mathcal{SP}_3$ has a polynomial pointwise time-oriented algorithm;*
(b) *if $\mathcal{SP}_3$ has a polynomial pointwise time-oriented algorithm, and if an upper-bound U on the makespan of the optimal schedule can be computed in polynomial time, then $\mathcal{SP}_3$ has a polynomial-delay list-generating algorithm.*

Proof. Assertion (a) holds as in Proposition 2.4.7.

Conversely, if $\mathcal{A}$ is a polynomial pointwise time-oriented algorithm, then $\mathcal{A}$ can be used to determine the makespan of S, by binary search over the interval $[0,U]$ (note that $\mathcal{A}$ returns $S_T(t) = (0,0,0)$ if and only if t exceeds the makespan of the optimal schedule). This requires $O(\log U)$ calls on $\mathcal{A}$, where $\log U$ is polynomial in $|D|$. Say the makespan is equal to C_{max}, and $S_T(C_{max}) = (j,t_1,t_2)$. Then, j is the last job to be scheduled. The same procedure can be iterated over the interval $[0,t_1]$, and eventually generates the processing intervals of all the jobs in reverse order. The whole procedure runs with polynomial delay. □

2.5 Applications: single-machine models

We now illustrate the concepts introduced in the previous sections on several examples of high multiplicity problems.

Example 2.5.1. (Weighted number of tardy jobs – continued). *A discussion of this problem was already started in Section 2.3. Let us now show that the approach in [58] leads to a polynomial two-phase job-oriented algorithm (and hence, to a polynomial pointwise job-oriented algorithm) for*

the optimization version of this problem. First, the solution $(x_{i,t})$ of the transportation problem can be computed in $O(s \log s)$ time and constitutes the required encoding Σ. Then, given a job index j, we first determine the type of this job, i.e., the unique index i^* such that $\sum_{1 \le i < i^*} n_i < j \le \sum_{1 \le i \le i^*} n_i$. If $r = j - \sum_{1 \le i < i^*} n_i$, we look at job j as the r-th replication of job type i^*. Next, we compute the index of the interval where j must be scheduled: this is the value of t^* such that $\sum_{1 \le t < t^*} x_{i^*,t} < r \le \sum_{1 \le t \le t^*} x_{i^*,t}$. Then, we compute the number of jobs which must be processed before j in the interval $(d_{t^*-1}, d_{t^*}]$. We can assume that this is

$$q = \sum_{1 \le i < i^*} x_{i,t^*} + (r - 1 - \sum_{1 \le t < t^*} x_{i^*,t})$$

(i.e., the number of jobs of type $i < i^*$ processed in the interval t^*, plus the number of jobs of type i^* processed before j but not already processed in a previous interval). Finally, the starting time of j is given by $d_{t^*-1} + q$. Clearly, this procedure yields $S_J(j)$ in (strongly) polynomial time. Similar arguments show that S_T can be computed pointwise in polynomial time. □

Example 2.5.2. (Total deviation JIT). *An instance of this problem has the form $D = (s, n_1, n_2, \ldots, n_s)$, with the usual interpretation. All jobs have unit processing time. Assume that all jobs have been sequenced on a single machine, and let $x_{i,t}$ denote the number of jobs of type i which have been sequenced in the interval $[0, t]$ $(i = 1, 2, \ldots, s;\ t = 1, 2, \ldots, n)$. The total weighted deviation JIT problem asks for a sequence which minimizes the total weighed deviation*

$$\sum_{i=1}^{s} \sum_{t=1}^{n} F(x_{i,t} - t\frac{n_i}{n}), \tag{2.1}$$

where F is a unimodal, convex function which penalizes the deviation between the actual cumulated production x_{it} and the ideal production tn_i/n up to time t.

Kubiak and Sethi in [72] and [74] gave a polynomial total time list-generating algorithm with complexity $O(n^3)$ for this problem, by reformulating it as an assignment problem. It is unknown whether its recognition or its evaluation versions can be solved in polynomial time, or even whether they are in NP or co-NP. □

Example 2.5.3. (Maximum deviation JIT). *This problem is similar to the previous one, except that the objective function (2.1) is replaced by a function penalizing the maximum deviation, namely*

$$\max_{1\le i\le s}\max_{1\le t\le n}|x_{i,t}-t\frac{n_i}{n}|. \tag{2.2}$$

[12] *showed that the recognition version of the maximum deviation JIT problem, i.e., $\mathcal{SP}_1$, is in co-NP, but the exact complexity of $\mathcal{SP}_1$ is currently unknown.* [105] *gave a polynomial total time list-generating algorithm for this problem. Interestingly, when the optimal objective value is known, then their algorithm produces the optimal schedule with polynomial delay (nothing similar seems to be known for the total deviation JIT problem, for instance).*

[12] *proved that the evaluation version $\mathcal{SP}_2$ of the maximum deviation JIT problem can be solved in polynomial time when s is fixed. In view of the previous remark, this also implies that the optimization version $\mathcal{SP}_3$ can be solved with polynomial delay when s is fixed. But even in this case, we do not know whether there is a polynomial pointwise algorithm for computing S_J or S_T.* □

2.6 General scheduling problems

In this section, we propose an extension of the above discussion which encompasses more general scheduling problems involving multiple machines and job preemptions. We first have to agree on the definition of a schedule in this framework. Several options exist, but the following one seems quite generic. For an instance involving n jobs and m machines, we define a schedule to be a (finite) subset S of $\{1,2,\ldots,n\}\times\{1,2,\ldots,m\}\times\mathbb{R}\times\mathbb{R}$. The interpretation is that, if $(j,k,t_1,t_2)\in S$, then job j must be processed on machine k without preemption from time t_1 to time t_2. So, the elements of S completely describe the Gantt chart of the schedule.

As in Section 2.4.1, a list-generating algorithm $\mathcal{A}$ for problem $\mathcal{SP}_3$ successively outputs the elements of an optimal schedule S. We denote by $\tau(l)$ the running time of $\mathcal{A}$ until it outputs the l-th element of S. In particular, $\tau(|S|)$ represents the running time of $\mathcal{A}$. All the complexity classes introduced in Section 2.4.1 can be generalized in a straightforward and consistent way: simply substitute $|S|$ for n in all definitions. Proposition 2.4.5 can also be partially generalized as follows.

Proposition 2.6.1. *If $\mathcal{A}$ is a list-generating algorithm for the optimization version $\mathcal{SP}_3$ of a general scheduling problem, then:*

$$\begin{aligned} \mathcal{A} \text{ runs in polynomial time} &\Longrightarrow \mathcal{A} \text{ runs with polynomial delay} \\ &\Longrightarrow \mathcal{A} \text{ runs in incremental polynomial time} \\ &\Longrightarrow \mathcal{A} \text{ runs in polynomial total time.} \end{aligned}$$

Proof. The same arguments apply as for Proposition 2.4.5. □

Remark 2.6.2. *For preemptive problems, the size of an optimal schedule S is not easily determined as a function of the input parameters. It may even happen that several optimal schedules exist, where some optimal schedules would be exponentially larger than others. In such a case, it may seem desirable to require that the algorithm $\mathcal{A}$ generate a schedule whose size is polynomially bounded in the size of the smallest optimal schedule. (Note that this difficulty is not specific to high multiplicity problems, but may even arise for traditional "low-multiplicity" problems.)*

For many preemptive scheduling problems, it can be shown that there exists an optimal schedule S involving a "small number" of preemptions, meaning typically that $|S| = O(nm)$. When this is the case, it it reasonable to require that S outputs a schedule of size polynomial in nm. □

Remark 2.6.3. *For general scheduling problems, contrary to the single-machine non-preemptive case, we cannot claim that polynomial total time algorithms also run in pseudo-polynomial time. This is because the size of the optimal schedule $|S|$, as opposed to the number of jobs n, may not be bounded by a polynomial in M, viz. the largest number occurring in the instance (cf. the proof of Proposition 2.4.5). If the size of the schedule generated by the algorithm is polynomial in nm (cf. previous remark), then polynomial total time implies pseudo-polynomial time, as in Proposition 2.4.5.* □

As in Section 2.3, our definition of a schedule allows for the derivation of several associated mappings, which can be viewed as providing alternative readings of the schedule. For instance, we can define:

- a *job-oriented* description of the schedule, corresponding to the mapping

$$S_J : j \mapsto S_J(j) = \{(k, t_1, t_2) \mid (j, k, t_1, t_2) \in S\}$$

(this is a natural extension of the definition given in Section 2.3, which describes the complete routing of a job through the shop);

• a *machine-oriented* description of the schedule, corresponding to the mapping

$$S_M : k \mapsto S_M(k) = \{(j, t_1, t_2) \mid (j, k, t_1, t_2) \in S\}$$

(this gives the Gantt chart for a particular machine);

• a *(machine,time)-oriented* description of the schedule, corresponding to the mapping

$$S_{MT} : \{1, 2, \ldots, m\} \times \mathbb{R} \to \{0, 1, \ldots, n\} \times \mathbb{R} \times \mathbb{R} : \ (k, t) \mapsto S_{MT}(k, t) =$$

$$= \begin{cases} (j, t_1, t_2) & \text{if } (j, k, t_1, t_2) \in S \text{ and either} \\ & t_1 \leq t \leq t_2, \text{ or } t_1 > t \text{ and } t_1 \text{ is the} \\ & \text{first instant when a job is processed} \\ & \text{on machine } k \text{ after time } t; \\ \\ (0, 0, 0) & \text{if there are no more jobs to be} \\ & \text{processed on machine } k \text{ after time } t. \end{cases}$$

($S_{MT}(k, t)$ describes the state of machine k at time t, or at the first moment when the machine is busy after time t).

Here again, many other partial descriptions of the schedule could be thought of.

Extending our previous definitions, we can say that an algorithm $\mathcal{A}$ is a polynomial pointwise algorithm for S_{MT} if, given any machine k and time t, $\mathcal{A}$ outputs $S_{MT}(k, t)$ in polynomial time.

However, analyzing the complexity of the derived mappings S_J or S_M deserves more care, since these mappings are set-valued, rather than single-valued as in Section 2.3. In their case, it seems natural to combine the notions of list-generating and pointwise algorithms. For instance, we could say that $\mathcal{A}$ runs (pointwise) with polynomial delay for S_J if, given any job j, $\mathcal{A}$ outputs the elements of the set $S_J(j)$ with polynomial delay. Such definitions may or may not prove useful or meaningful, depending on the context, and we will not dive deeper into their discussion.

2.7 Applications: general case

Clifford and Posner in [22] investigate the complexity of several parallel machine scheduling problems with high multiplicity. They establish that several of these problems are polynomially solvable both in their recognition and in their optimization versions (e.g., $P|\ |\sum_j C_j$ or $Q|\ |\sum_j C_j$), but they also argue that there is no polynomial description of the optimal schedule in terms of job groups for some other problems (e.g., $P|pmtn|C_{max}$ and $Q2|pmtn|\sum_j C_j$). We now show, however, that this does not preclude other efficient descriptions of the optimal schedule. We only handle two simple cases, as these suffice to illustrate our claim.

Example 2.7.1. (Makespan minimization). *Consider first $P|pmtn|C_{max}$, i.e., the makespan minimization problem for a set of jobs to be processed preemptively over parallel identical machines. An instance of the problem is a vector*

$$D = (s, n_1, n_2, \ldots, n_s, p_1, p_2, \ldots, p_s, m),$$

where m is the number of machines and p_i is the processing time of a job of type i ($i = 1, 2, \ldots, s$).

Clifford and Posner in [22] observe that the evaluation version of $P|pmtn|C_{max}$ can be solved in polynomial time. Indeed, in view of a well-known result of [86], the optimal value of this problem is equal to

$$C^*_{max} = \max\{(\sum_{i=1}^{s} n_i p_i)/m, p_1, p_2, \ldots, p_s\},$$

which can be efficiently computed.

McNaughton's algorithm determines a schedule with makespan equal to C^*_{max}. It first lists all jobs in the natural order $1, 2, \ldots, n$. Then, it cuts this sequence, viewed as a single-machine schedule, into at most m subsequences of length C^*_{max}. Finally, the k-th subsequence is assigned to machine k, for $k = 1, 2, \ldots, m$ (see [86, 94]).

Even with a single job type, the optimal schedule may require $\Omega(m)$ preemptions, where m is exponential in the input size. From this, Clifford, and Posner [22] conclude that "it is not possible to create an optimal schedule (...) in polynomial time" and hence, that the optimization version of $P|pmtn|C_{max}$ in $EXP \setminus P$. This is rather surprising, in view of the simplicity of the evaluation problem $\mathcal{SP}_2$ and of McNaughton's algorithm. As a

matter of fact, we note that the optimal schedule can actually be computed with polynomial delay. More precisely, the job-oriented description S_J can be computed (pointwise) with polynomial delay, as follows easily from the description of McNaughton's algorithm. This implies (as in Proposition 2.4.7) that an optimal schedule can be generated with polynomial delay. Similarly, the (machine,time)-oriented description S_{MT} can be computed pointwise in polynomial time. □

Example 2.7.2. (Sum of completion times). *Consider the problem $Q2|pmtn|\sum_j C_j$. An instance is a vector*

$$D = (s, n_1, n_2, \ldots, n_s, p_1, p_2, \ldots, p_s, v_1, v_2),$$

where $p_1 < p_2 < \ldots < p_s$, v_1 (resp. v_2) denotes the speed of machine 1 (resp. machine 2) and $v_1 \geq v_2$. In an optimal schedule, the first job of type 1 starts on machine 1 at time 0. All other jobs start processing on machine 2 in SPT order, and are moved to machine 1 whenever this machine becomes available.

Clifford and Posner [22] define the quantity $\sigma_i(j)$, representing the amount of time that the j-th job of type i spends on machine 1. They prove that, for $j = 1, 2, \ldots, n_i$ and $i = 1, 2, \ldots, s$,

$$\sigma_i(j) = \frac{p_i}{v_1 + v_2} - \left(\frac{p_i}{v_1 + v_2} - \sigma_i(0)\right)\left(-\frac{v_2}{v_1}\right)^j, \tag{2.3}$$

where $\sigma_1(0) = 0$ and $\sigma_i(0) = \sigma_{i-1}(n_{i-1})$ for $2 \leq i \leq s$. From these difference equations, they derive an expression of the optimal value which can be computed in $O(s^2)$ time, thus proving that the evaluation version of the problem is solvable in polynomial time. However, even when $s = 1$, each job may have a different processing time on machine 1. So, here again, Clifford and Posner [22] conclude that the optimization version of $Q2|pmtn|\sum_j C_j$ is in $EXP \setminus P$.

Note that, in view of the above description, every job j is preempted at most once in the optimal schedule, so that the job-oriented description $S_J(j)$ contains at most two elements for $j = 1, 2, \ldots, n$. We claim that $S_J(j)$ can be computed in polynomial time for all j, and hence, an optimal schedule can be generated with polynomial delay.

To establish our claim, consider a job j^*. As in Example 1, assume that j^* is the r-th replication of job type i^*. Job j^* starts on machine 1 as soon as all previous jobs have been completed on this machine, meaning at time $t_1 =$

Problem	$\mathcal{SP}_1$	$\mathcal{SP}_2$	$\mathcal{SP}_3$	
			List-generating	Pointwise
$1\|p_j = 1\|\sum_j w_j U_j$	P	P	polynomial delay	S_J, S_T polynomial
total deviation JIT	?	?	total polynomial	?
max deviation JIT	co-NP	?	total polynomial	?
max deviation JIT, fixed s	P	P	polynomial delay	?
$P\|pmtn\|C_{max}$	P	P	polynomial delay	S_J polynomial delay, S_{MT} polynomial
$Q2\|pmtn\|\sum_j C_j$	P	P	polynomial delay	S_J polynomial

Table 2.1: Complexity of various problems

$\sum_{1 \le i < i^*} \sum_{1 \le j \le n_i} \sigma_i(j) + \sum_{1 \le j < r} \sigma_{i^*}(j)$. *Standard summation formulas for power series allow to compute t_1 in polynomial time. We can also easily compute how much time j^* spends on machine 2 (namely, $(p_{i^*} - v_1 \sigma_{i^*}(r))/v_2$ units of time) and, subtracting this quantity from t_1, deduce the starting time of j^* on machine 2. This yields a complete description of $S_J(j^*)$ in polynomial time.* □

The results concerning the different models discussed in this section, and in Section 2.5, are summarized in Table 1. Note that all these problems can be solved in pseudo-polynomial time. A question mark in the table means that we do not know anything beyond this fact (which is often nontrivial in itself).

2.8 Discussion

The complexity of high multiplicity scheduling problems has been discussed by several authors, but it seems that a fully satisfactory framework has been missing so far for this discussion. The aim of this chapter is to propose such a framework.

A main (albeit obvious) observation is that the complexity of the task "output an optimal schedule" cannot be meaningfully discussed unless we explicitly clarify the form of the output. It makes a big difference, for instance, whether we want to generate all elements of a schedule, viewed as a set, or whether we just want to compute some elements of the schedule, viewed as a mapping.

In the classification scheme that we propose, we see that there is in fact no essential difference between the nature of the simple, compact encoding of an optimal schedule for $P|pmtn|C_{max}$ provided by the description of McNaughton's algorithm [86], and the nature of the compact encoding of an optimal schedule for $1|p_j = 1|\sum_j w_j U_j$ provided by the solution of the transportation problem in [58] (cf. Table 1). In both cases, what comes out of the algorithm and of its proof of correctness is not an *explicit* description of the optimal schedule, but an *implicit* description – an encoding – which can be used either to generate the schedule in extension, or to compute various derived mappings in a pointwise fashion (in a 2-phase approach).

Different encodings of the optimal schedule, however, may differ in the extent to which they allow an efficient decoding into explicit schedules. The classification of algorithms proposed in this chapter provides one way of distinguishing algorithms on this basis.

Algorithms for high multiplicity scheduling problems may also be compared on other grounds than those discussed in previous sections. In particular, it may be desirable to classify them on the basis of their space complexity, rather than time complexity only. For instance, Kubiak and Sethi's ([72] and [74]) formulation of the total deviation just-in-time problem requires $\Omega(n)$ time *and* space. Similar issues have been discussed, in various contexts, by [29, 79] and [64].

Finally, the results displayed in Table 1 suggest that the relationship between different complexity classes may go deeper than the simple implications mentioned in Propositions 2.4.5 or 2.6.1. It would be useful to investigate some of these relations in future work.

Chapter 3

The high multiplicity travelling salesman problem

3.1 Introduction

Mass customization continues to place new demands on manufacturers. These new demands concern manufacturing technology, as well as planning and scheduling issues. The customization is often achieved by offering a choice of (optional) parts on a generic product. In manufacturing terms, this comes down to being able to customize the generic product during the assembly stage, using various components. This concept has become known as assemble to order [8]. Together with lean manufacturing requirements, assemble to order techniques pose new challenging planning and scheduling problems for manufacturers.

In a mass customization environment, demand for products is not entirely handled at the customer level. For several stages of the production process, demand is rather dealt with at the level of product variations, for instance, because the demand figures are (partly) forecasts. The idea is that there is one product variation for each possible choice of components (or several closely resembling choices combine into one variation). In this chapter we refer to the variations as types. Further, we deal with a situation where the assembly process is executed on an assembly line. As is customary in such settings, we assume that demand is specified at the level of product types: expected number of products per type for the next planning period, e.g., a month. The question that we would like to answer is: how to sequence all products of the various types through the assembly process?

Of course, this question has been studied before in several different contexts. First of all, there is a line of research dealing with so-called assembly line balancing problems (see, e.g., [4, 76, 89]). This research originates from car assembly line scheduling problems at Toyota. Mostly, these papers assume that each assembly step takes a same, constant, amount of time. The task might now be to sequence the cars in such a way that the output levels are at any time proportional to the demand ratios. Alternatively, the sequencing problem might be to sequence the cars in such a way that the lines feeding the assembly line have a balanced workload.

Another line of research, having its roots in a variety of different manufacturing applications, also deals with related sequencing problem. These problems may again assume a linear production system, but the sequencing objective is now to optimize throughput rather than the the line balance. Related models can be found in maintenance and data broadcast scheduling [7, 9, 68, 107].

The problem under investigation can be further defined and specified as follows. We denote by $J = \{1, 2, \ldots, s\}$ the set of product (job) types. For each product (job) type $j \in J$ there is a nonnegative integer D_j describing the demand of the j-th product for the next planning period, which specifies the number of type j jobs to be assembled. Further, we denote by $c_{i,j}$ the time interval required between inputting a type i job and a type j job, $i, j \in J$, into the system. These $c_{i,j}$'s can be considered as sequence dependent processing times in a single machine scheduling problem, as sequence dependent switch over times, or as the waiting times between inputting parts into a permutation no-wait flow shop. Such a permutation no-wait flow shop system is often an adequate model for an assembly system. Notice that this problem specification is quite general. (For instance, it contains scheduling problems in which the processing times are not sequence dependent as a special case.)

When $D_j = 1$ for all $j \in J$, and interpreting the $c_{i,j}$ as the distance between a city i and a city j, the sequencing problem that minimizes the sum of the sequence dependent $c_{i,j}$ is the classic travelling salesman problem (TSP). For reasons that will become clear soon, we refer to scheduling problems in which $D_j = 1$ for all job types as *single multiplicity* problems. In the classical, single multiplicity, TSP, we have the number of cities to be visited $n = s$ and an explicit input parameter $c_{i,j}$ for each pair of cities. Hence, the length of the encoded input is $O(s^2 \max_{i,j \in J} \log c_{i,j})$. In the sequencing problem that we investigate, there are multiplicities D_j, $j \in J$, of jobs having

the same processing requirements. Therefore, in any compact encoding the input is of length $O(s^2 \max\{\max_{j\in J} \log D_j; \max_{i,j\in J} \log c_{i,j}\})$ and n may well be exponential with respect to this input size. For purposes of this chapter the scheduling (sequencing) problems in which the input is specified this way, we call *high multiplicity scheduling (sequencing)* problems.

As we know already from the introduction in Chapter 1, from a complexity perspective, high multiplicity scheduling problems give rise to several characteristic questions. Let us demonstrate this more specifically by considering high multiplicity sequencing problems with a makespan minimization objective. First of all, note that explicit specification for a high multiplicity sequence of jobs requires an amount of space that is not polynomial in the input size. Therefore, it is not a priori clear that high multiplicity sequencing problem are in P-SPACE, let alone in NP. Hence, researchers have investigated whether optimal sequences can be compactly encoded. In particular, a question that has attracted attention is: Can polynomial algorithms for single multiplicity problem be extended so as to solve in polynomial time, the high multiplicity version (see, e.g., [2])? Another question that has been researched is: For problems that are known to be NP-complete in the single multiplicity version, is the high multiplicity version solvable in polynomial time if the number of types is fixed? This question is especially interesting since in many practical applications, the number of different types will be small compared to the numbers of multiplicities.

In practical settings in which the above high multiplicity scheduling problems arises, makespan optimization is important but not necessarily the dominant objective function. In addition, managers and planners think it to be desirable to have the output over time more or less in proportion to the multiplicities of the types. Among the advantages of such a policy are low inventories and a smooth workload. Consequently, it has become standard practice to repeatedly execute a short assembly sequence in which the jobs are produced in the required relative quantities. The concept of *minimal part sets* builds on this idea [54, 55]. A minimal part set (MPS) is constructed as follows. Let D_0 be the greatest common divisor of the set of multiplicities $D_0 = GCD(D_1, D_2, \ldots, D_s)$. Further let $n_j = D_j/D_0$, for $j \in J$. The minimal part set is described by the vector $(n_1, n_2, \ldots, n_s)$. A popular approach to find a good sequence is now to find a sequence for the MPS that, when repeatedly executed, yields among all such sequences the highest throughput rate.

In assembly or production systems, the approach described above ide-

ally results in low inventories, balanced workloads, and a simple and stable schedule, which is not too complicated to find. However, this approach may well yield sub-optimal throughput rates. In this chapter we investigate high multiplicity sequencing problems with a cycle time minimization objective. In particular, we are interested in the trade-off between a long run cycle time versus the length of the MPS sequence that is repeatedly executed. A basic question that we will address is whether for a given instance there exists a finite repetitive cyclic policy that yields the optimal throughput rate.

The permutation no-wait flow shop scheduling problem $F|no-wait|C_{\max}$ in the three field representation scheme by [40], is an important application for the research presented in this chapter. We therefore quickly review its complexity status as well as complexity status of its high multiplicity version. It is well known, see, e.g., [109], that the permutation no-wait flow shop can be modelled as a travelling salesman problem. For the case of two machines, the resulting TSP instances have a distance matrix with a special structure, and, in the single multiplicity case, the cycle time minimization problem can be solved in polynomial time by the Gilmore & Gomory algorithm, see [38]. Agnetis in [2] shows that the high multiplicity version of the two machine no-wait flow shop scheduling problem can also be solved in polynomial time, by an extension of the Gilmore & Gomory algorithm. From [98] we know that for three and more machines the decision versions of permutation no-wait flow shop scheduling problems are NP-complete in the strong sense as well as the travelling salesman problem in general.

The idea of finding an optimal sequence for an MPS is encountered in various other settings, such as the literature on assembly line balancing, or the literature on maintenance scheduling. Another previously investigated problem that fits in this framework is the high multiplicity TSP (HMTSP), which is sometimes referred to as "the travelling salesman problem with many visits to few cities". We derive several results that can be applied to HMTSP. In this respect, our work extends earlier work on the HMTSP, as it can be found in [23, 65, 69, 96, 107].

3.2 Problem statement and formulations

In this chapter we consider a general high multiplicity scheduling problem with sequence dependent processing times. In fact, we will not be interested so much in the actual scheduling of tasks, i.e., specifying starting times, but rather in sequencing jobs, i.e., determining the order in which they are to be

processed. More specifically, we will be interested in what we call the high multiplicity travelling salesman problem, since it is general enough to serve as a model for a variety of scheduling applications (see, e.g., Remark 3.2.3 or [23, 107]).

Definition 3.2.1. *The single multiplicity NO-WAIT FLOW SHOP problem (NWFS):*
Given is the set of jobs $J = \{1, 2, \ldots, n\}$ *where each job* $j \in J$ *consists of* m *operations* $O_{1j}, O_{2j}, \ldots, O_{mj}$. *Operation* O_{kj}, $1 \leq k \leq m$, *requires nonnegative processing time* p_{kj}. *To complete the job* $j \in J$ *all operations* $O_{1j}, O_{2j}, \ldots, O_{mj}$ *must be completed without any interruptions (neither operation interruption nor interruption between the operations). At any time only one operation of any job can be processed. The operations order is the same for all jobs:* $1 \prec 2 \prec \ldots \prec m$. *It is required to find a schedule (define the starting times of the jobs or define the execution sequence of the jobs) that minimizes the maximum completion time (makespan) overall jobs.*

Definition 3.2.2. *The high multiplicity travelling salesman problem:*

$$F(P) = \min_{x} \sum_{i \in J} \sum_{j \in J} c_{ij} x_{ij} \tag{3.1}$$

subject to

$$\sum_{i \in J} x_{ij} = n_j, \quad j \in J; \tag{3.2}$$

$$\sum_{j \in J} x_{ij} = n_i, \quad i \in J; \tag{3.3}$$

$$\sum_{i \in J'} \sum_{j \in J \setminus J'} x_{ij} \geq 1, \quad \forall J' \subset J \text{ such that } J' \neq \emptyset; \tag{3.4}$$

$$x_{ij} \in \mathbb{Z}^+, \quad i \in J, \ j \in J. \tag{3.5}$$

For a generic instance we will refer to this formulation as P, and to the value of its optimal solution as $F(P)$.

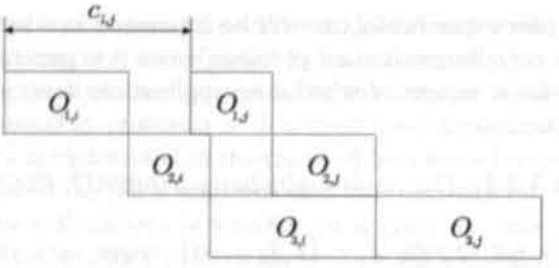

Figure 3.1: Computing $c_{i,j}$, $i \in J$, $j \in J$

Remark 3.2.3. *Consider the high multiplicity formulation of the NWFS for a single MPS with cycle time minimization objective. For each ordered pair of types $(i,j) \in J \times J$ it is possible to compute a number $c_{i,j}$ which is the time that must elapse between consecutively sequenced products (jobs, parts) of type i and type j respectively.*

It is well known (see, e.g., [109]) that using the distances $c_{i,j}$, $i \in J$, $j \in J$, computed as follows

$$c_{i,j} = p_{1,i} + \max_{k=1,2,\ldots,m-1} \left\{ 0, \sum_{t=1}^{k} (p_{t,j} - p_{t+1,i}) \right\}, \tag{3.6}$$

NWFS can be straightforwardly formulated as TSP. Indeed, letting the jobs correspond to cities, and letting $c_{i,j}$, $i \in J$, $j \in J$, be the distance between cities i and j, the task is to find a closed walk through all cities such that it is visited every city exactly once and such that the total length of the walk is minimal overall such walks.

As we already mentioned, the HMTSP is also known as the traveling salesman problem with many visits to few cities. From its formulation as an integer linear program we conclude that the decision version of the HMTSP, is in NP. Despite the exponential number of constraints, this is true due to the fact that there exists a well-known polynomial time test whether the so-called *subtour elimination constraints* (3.4) are satisfied or not, see, e.g., [101]. Moreover, using Lenstra's algorithm for ILP in fixed dimension, see [80], HMTSP can be solved in polynomial time when the number of cities (types) is fixed. In fact, combinatorial, strongly polynomial, algorithms for the case where the number of cities is bounded from above by a constant

have been developed by Cosmadakis, Papadimitriou, Kanelakis, and Van de Klundert in [23, 65, 69].

We now explain how an (optimal) *sequence* can be constructed from an (optimal) solution x^0 to P (see, for instance, [30, 69, 107]). Let $[x^0_{i,j}]_{s\times s}$ be an (optimal) solution to (P). We are going to describe a procedure *ConvertToSequence* that can be repeatedly executed to convert the solution and matrix $[x^0_{i,j}]_{s\times s}$ into a closed walk that traverses arc (i,j), exactly $x_{i,j}$ times $(i,j \in J)$, and which represents a high multiplicity sequence in which to visit the cities.

We denote by $G(J,A)$ the multigraph where the nodes correspond to the cities in the HMTSP, and each arc (i,j) occurs with multiplicity $x^0_{i,j}$. In the procedure *ConvertToSequence* we use the phrase "simple cycle", which is to be understood as a simple cycle in G. More precisely, we define a cycle $C = ((i_1,i_2),(i_2,i_3),\ldots,(i_t,i_1))$, $i_\tau \in J$, $\tau \in \{1,2,\ldots,t\}$ to be a *simple cycle* if it does not contain any subcycle, i.e., if $i_{\tau'} \neq i_{\tau''}$ for any two different indices $\tau',\tau'' \in \{1,2,\ldots,t\}$.

Definition 3.2.4. *Procedure ConvertToSequence:*

Input: $[x^0_{i,j}]_{s\times s}$.

Output: *A closed walk, represented by a finite collection $\mathcal{C}$ of pairs (m_C, C), where C is a simple cycle and m_C is an integer corresponding to the number of copies of cycle C in the collection $\mathcal{C}$.*

1. $\mathcal{C} := \emptyset, q = 1$.
2. *Find a cycle* $C_q = ((i_1,i_2),(i_2,i_3),\ldots,(i_t,i_1))$, $i_\tau \in J$, $\tau \in 1,2,\ldots,t$, *such that* $x^{q-1}_{i_\tau,i_{\tau+1}} > 0$, $\tau \in \{1,2,\ldots,t-1\}$ *and* $x^{q-1}_{i_t,i_1} > 0$.
3. *Let* $m_q = \min\{x^{q-1}_{i,j} | (i,j) \in C_q\}, \mathcal{C} := \mathcal{C} \cup \{(m_q, C_q)\}$ *and* $x^q_{i,j} := x^{q-1}_{i,j} - m_q$ *for* $(i,j) \in C_q$ *and* $x^q_{i,j} := x^{q-1}_{i,j}$ *otherwise.*
4. *If* $[x^q_{i,j}]_{s\times s} = [0]_{s\times s}$, *output:* $\mathcal{C}$ *and stop, else set* $q := q+1$, *and goto step 2.*

The reader may verify that this algorithm can be read as an extension of the text book algorithm, see, e.g., [92], to find a Eulerian trail in a Eulerian graph in the following way. We construct a Eulerian walk by traversing the first edge i_1, i_2 of the first simple cycle C_1. Then, we check whether city i_2 is the first city in any other cycle C_l. If this is the case we continue by recursively traversing C_l. Otherwise, or after having recursively traversed C_l, we continue by traversing the second edge of C_1, i.e., (i_2, i_3). For city i_3, again we look whether there is some simple, and yet untraversed cycle C_q that starts in i_3. If so, we start traversing C_q, and otherwise, or after having recursively traversed C_q, we continue traversing C_1 and so on. When C_1 is completely traversed, i.e., after having traversed the arc (i_t, i_1), and recursively all cycles starting at i_1, we finish by $m_1 - 1$ times traversing C_1 without making recursive traversals. In general, when during the construction of the Eulerian walk some simple cycle C_q is completely traversed, we traverse it additionally $m_q - 1$ times, without making the recursive traversals.

By constraints (3.2) and (3.3), for every city $i \in J$ it holds that $\sum_{j \in J} x_{i,j} = \sum_{j \in J} x_{j,i} = n_i$. It implies that in the graph G every vertex has an even degree and therefore, see [92], multigraph G is Eulerian, i.e., it contains an Eulerian cycle (trail, walk). For the correctness of the procedure *ConvertToSequence* we also refer to [30]. Let us briefly discus the complexity of the procedure. First of all, it is not hard to verify that step 2 can be executed in $O(s^2)$ time. Further, $[x^n_{i,j}]_{s \times s}$ contains at least one more zero element than $[x^{n-1}_{i,j}]_{s \times s}$, by definition of step 3. Steps 3 to 4 are executed in at most $O(s)$ time. This leaves the overall time complexity of *ConvertToSequence* to be $O(s^4)$. In addition, this reasoning implies that the collection $\mathcal{C}$ consists of at most $O(s^2)$ pairs (m_C, C). Since m_C and C are also polynomially bounded in the input size, the output of *ConvertToSequence* encodes an optimal solution polynomially in the input size. Hence, we have two compact encoding schemes of optimal solutions: $\mathcal{C}$, and $[x^0_{i,j}]_{s \times s}$. In subsequent sections these compact encoding schemes will be contrasted with not necessarily polynomial encoding schemes in which solutions are more explicitly specified. An example of such an encoding scheme is to disregard the multiplicities and completely specify the sequence in which the cities are to be visited.

We now continue by considering some relaxations and extensions of Definition 3.2.2. For a generic instance we define T to be the problem that results from P by relaxing the subtour elimination constraints (3.4):

Definition 3.2.5.

$$F(T) = \min_{x} \sum_{i \in J} \sum_{j \in J} c_{i,j} x_{i,j} \tag{3.7}$$

subject to

$$\sum_{i \in J} x_{i,j} = n_j, \quad j \in J; \tag{3.8}$$

$$\sum_{j \in J} x_{i,j} = n_i, \quad i \in J; \tag{3.9}$$

$$x_{i,j} \in \mathbb{Z}^+, \quad i \in J,\ j \in J. \tag{3.10}$$

We denote the value of the optimal solution of T by $F(T)$. Problem T is a transportation problem, and therefore always has an integral optimal solution that can be found in time polynomial in the input size, see, e.g., [92].

Finally, we introduce a more general problem formulation of the cycle time minimization problem. Since the research aims to find input sequences that yield maximum throughput, we propose a formulation that allows more general input sequences, not just input sequences for MPSs:

Definition 3.2.6. *Consider a positive integer parameter* $l \in \mathbb{N}$. *Let us define the following problem, which is parametric on* l*:*

$$F(l) = \frac{1}{l} \min_{x} \sum_{i \in J} \sum_{j \in J} c_{i,j} x_{i,j} \tag{3.11}$$

subject to

$$\sum_{i \in J} x_{i,j} = l \times n_j, \quad j \in J; \tag{3.12}$$

$$\sum_{j \in J} x_{i,j} = l \times n_i, \quad i \in J; \tag{3.13}$$

$$\sum_{i \in J'} \sum_{j \in J \setminus J'} x_{i,j} \geq 1, \quad \forall J' \subset J \text{ such that } J' \neq \emptyset; \tag{3.14}$$

$$x_{i,j} \in \mathbb{Z}^+, \quad i \in J, j \in J. \tag{3.15}$$

Let us call this problem $P(l)$. We shall again refer to the objective function of $P(l)$ by $F(P(l))$, or $F(l)$ for short.

Now, the problem is to find such a parameter l^* that minimizes $F(l)$ over all $l \in \mathbb{N}$. Indeed, given an optimal pair (l^*, x^*), it encodes a sequence in which each city $j \in J$ is visited $\sum_{i \in J} x^*_{i,j} = l^* \times n_j$ times. Hence, the resulting sequence contains l^* MPSs. Let us recall that the aim is to maximize the throughput rate. This is achievable by minimizing the minimum cycle time over all natural numbers l. Thus, in a scheduling context, and allowing l^* be infinity, the optimal solution value $F(l^*)$ specifies the maximum throughput rate attainable while producing all types in the prespecified ratios.

Notice that $P(l)$ can be solved in polynomial time when s is fixed [80].

As before, we define $T(l)$ to be the problem that results from $P(l)$ after relaxing the subtour elimination constraints:

Definition 3.2.7.

$$F(T(l)) = \frac{1}{l} \min_{x} \sum_{i \in J} \sum_{j \in J} c_{i,j} x_{i,j} \tag{3.16}$$

subject to

$$\sum_{i \in J} x_{i,j} = l \times n_j, \quad j \in J; \tag{3.17}$$

$$\sum_{j \in J} x_{i,j} = l \times n_i, \quad i \in J; \tag{3.18}$$

$$x_{i,j} \in \mathbb{Z}^+, \quad i \in J,\ j \in J. \tag{3.19}$$

Notice that defining $y_{i,j} = x_{i,j}/l$, the problem $T(l)$ can be rewritten as

$$F(T(l)) = \min_{y} \sum_{i \in J} \sum_{j \in J} c_{i,j} y_{i,j} \tag{3.20}$$

subject to

$$\sum_{i \in J} y_{i,j} = n_j, \quad j \in J; \tag{3.21}$$

$$\sum_{j \in J} y_{i,j} = n_i, \quad i \in J; \tag{3.22}$$

$$y_{i,j} \in \mathbb{Z}^+, \quad i \in J,\ j \in J. \tag{3.23}$$

Hence, we conclude that $T(l)$ and T are identical transportation problems. By consequence, for any natural number l the value $F(T)$ yields a lower bound on $F(l)$.

3.3 General properties of optimal solutions

In this section we derive some basic properties of the problem $P(l)$. We derive some results on how the optimal value $F(l)$ decreases when l increases. All of these results are based on the following inequality:

Theorem 3.3.1. *For any natural number l, the following inequality holds*

$$F(l+1) \leq \frac{l}{l+1}F(l) + \frac{1}{l+1}F(T). \tag{3.24}$$

Proof. Let x^{l+1} and $x^l, l \in \mathbb{N}$, denote optimal solutions for $P(l+1)$ and $P(l)$ respectively, and let x^T denote an optimal solution for T. Now, notice that the vector $x^l + x^T$ is a feasible solution for $P(l+1)$. Thus, for any number $l \in \mathbb{N}$ we have

$$\begin{aligned}
F(l+1) &= \frac{1}{l+1}\sum_{i\in J}\sum_{j\in J} c_{ij}x_{ij}^{l+1} \\
&\leq \frac{1}{l+1}\sum_{i\in J}\sum_{j\in J} c_{ij}x_{ij}^{l} + \frac{1}{l+1}\sum_{i\in J}\sum_{j\in J} c_{ij}x_{ij}^{T} \\
&= \frac{l}{l+1}\left(\frac{1}{l}\sum_{i\in J}\sum_{j\in J} c_{ij}x_{ij}^{l}\right) + \frac{1}{l+1}\sum_{i\in J}\sum_{j\in J} c_{ij}x_{ij}^{T} \\
&= \frac{l}{l+1}F(l) + \frac{1}{l+1}F(T)
\end{aligned}$$

as required. □

Corollary 3.3.2. *$F(l) \geq F(l+1)$ holds for any $l \in \mathbb{N}$.*

Proof. Using Theorem 3.3.1 we straightforwardly derive,

$$\begin{aligned}
F(l+1) &\leq \frac{l}{l+1}F(l) + \frac{1}{l+1}F(T) \\
&= \frac{l}{l+1}F(l) + \frac{1}{l+1}F(T(l)) \\
&\leq \frac{l}{l+1}F(l) + \frac{1}{l+1}F(l) \\
&= F(l),
\end{aligned}$$

where the second inequality follows from the fact that $T(l)$ is obtained from $P(l)$ by relaxing the sub-tour elimination constraints. □

Theorem 3.3.3. *$F(l) > F(l+1)$ if and only if $F(l) > F(T)$.*

Proof. If $F(l) > F(T)$, then it must hold that

$$\begin{aligned} F(l) &= \frac{l}{l+1}F(l) + \frac{1}{l+1}F(l) \\ &> \frac{l}{l+1}F(l) + \frac{1}{l+1}F(T) \\ &\geq F(l+1) \end{aligned}$$

by Theorem 3.3.1. Conversely, if $F(l) = F(T)$, it must hold that

$$\begin{aligned} F(l+1) &\geq F(T(l+1)) \\ &= F(T) \\ &= F(l). \end{aligned}$$

Combined with Corollary 3.3.2 this yields that $F(l+1) = F(l)$. □

By consequence, we also have the following corollary:

Corollary 3.3.4. *If there exists l^0 such that $F(l^0) = F(T)$ then $F(l) = F(T)$ for any number $l \geq l^0$.*

Hence, the question arises whether for a given instance there exists l^0 such that $F(l^0) = F(T)$? Moreover, in the case that the answer is affirmative, one would like to efficiently compute or at least estimate the smallest number l^0 such that the equality holds. These questions will be discussed and answered in Sections 3.4 and 3.5. In the case that the answer is negative, there might still be some long run stabilization worthy of characterization. Pursuing this direction, we first consider the following theorem, which is an extension of Theorem 3.3.1.

Theorem 3.3.5. *For any fixed natural number l^* and any natural number $l \geq l^*$ the following inequality holds*

$$F(l) \leq \frac{l^*}{l}F(l^*) + \frac{l - l^*}{l}F(T). \tag{3.25}$$

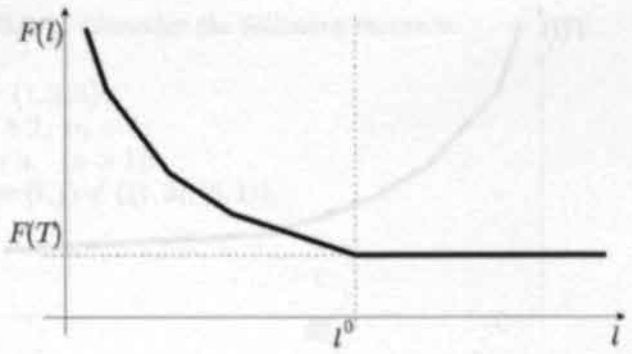

Figure 3.2: Stabilization on $F(T)$

Proof. We shall prove the theorem by induction on l from the basis l^*. For any natural number l^* we derive that $F(l^*) \leq l^* \times F(l^*)/l^* + (l^* - l^*) \times F(T)/l = F(l^*)$, and hence (3.25) holds for $l = l^*$.

Now, suppose that for some number $l \geq l^*$ property (3.25) holds. We prove that based on this induction hypothesis the validity of this property for $l+1$ can be derived. From Theorem 3.3.1 we obtain that $F(l+1) \leq l \times F(l)/(l+1) + F(T)/(l+1)$ holds for every natural number l. Hence, by induction we have that

$$\begin{aligned} F(l+1) &\leq \frac{l}{l+1}F(l) + \frac{1}{l+1}F(T) \\ &\leq \frac{l}{l+1}\left(\frac{l^*}{l}F(l^*) + \frac{l-l^*}{l}F(T)\right) + \frac{1}{l+1}F(T) \\ &= \frac{l^*}{l+1}F(l^*) + \frac{l+1-l^*}{l+1}F(T) \end{aligned}$$

and the proof is complete. □

Finally, we prove that the value $F(l)$ converges to $F(T)$ when l goes to infinity.

Theorem 3.3.6. *If there exists a (finite) natural number l^* such that $F(l^*)$ is finite then*

$$\lim_{l \to +\infty} F(l) = F(T). \tag{3.26}$$

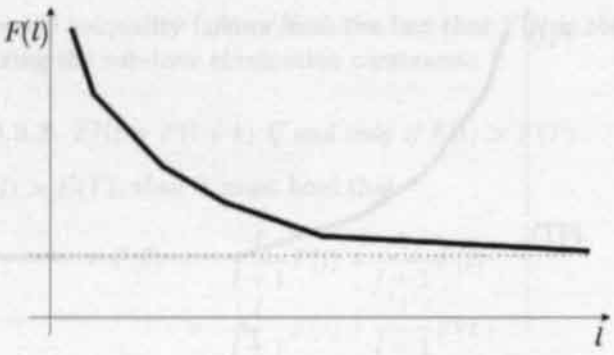

Figure 3.3: Convergency to $F(T)$ in general case

Proof. Using Theorem 3.3.5, and letting $l > l^*$ we derive

$$\begin{aligned} F(T) &\leq F(l) \\ &\leq \frac{l^*}{l} F(l^*) + \frac{l - l^*}{l} F(T) \\ &= F(T) + \frac{l^*}{l} \left(F(l^*) - F(T) \right). \end{aligned}$$

Hence,

$$\begin{aligned} F(T) &\leq \lim_{l \to +\infty} F(l) \\ &\leq \lim_{l \to +\infty} \left(F(T) + \frac{l^*}{l} \left(F(l^*) - F(T) \right) \right) \\ &= F(T). \end{aligned}$$

which yields the desired result. □

In Section 3.5 we obtain a much stronger result which improves Theorems 3.3.1 and 3.3.5. Namely, we shall find that for a very general class of instances, and for all $l \geq s - 1$, it holds that $F(l) = F(s-1) \times (s-1)/l + F(T) \times (l - s + 1)/l$.

We finish this section by demonstrating how optimal solutions and objective function values change with l.

Example 3.3.7. *Consider the following instance:*

Let $J = \{1,2,3\}$;
$n_1 = 1,\ n_2 = 1,\ n_3 = 1$;
$c_{1,3} = c_{3,1} = a,\ \ (a > 1)$;
$c_{i,j} = 1 \quad$ *for* $(i,j) \notin \{(1,3),(3,1)\}$.

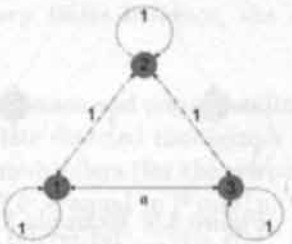

Figure 3.4: Example 3.3.7

The reader is encouraged to verify that

- *For the transportation problem T it holds that $F(T) = 3$, $x^T_{i,j} = 0$ for $i \neq j$ and $x^T_{i,j} = 1$ for $i = j$. The collection $\mathcal{C}$ as output by Convert-ToSequence in this case is $\mathcal{C} = \{(1,(1,1)),(1,(2,2)),(1,(3,3))\}$.*

- *For the problem $P(1)$ we have $F(P) = a + 2$. An optimal solution for this problem is $x^1_{i,j} = 1$ for $(i,j) \in \{(1,2),(2,3),(3,1)\}$ and $x^1_{i,j} = 0$ for any other arcs. In this case $\mathcal{C}$ is $\{(1,((1,2),(2,3),(3,1)))\}$.*

- *For any natural number $l \geq 2$ we get $F(l) = 3$. An optimal solution is specified by $x^l_{1,2} = x^l_{2,3} = x^l_{3,2} = x^l_{2,1} = 1$; $x^l_{1,1} = x^l_{3,3} = l - 1$; $x^l_{3,1} = x^l_{1,3} = 0$ and $x^l_{2,2} = l-2$. Further, $\mathcal{C} = \{((1,(1,2)(2,3),(3,2),(2,1)),((l-1),(1,1)),((l-2),(2,2)),((l-1),(3,3)))\}$.*

In this example it holds that $F(l) = F(T)$, for $l \geq 2$. This example also demonstrates that the ratio between $F(P)$ and $F(l)$ can be arbitrarily large, namely $F(P)/F(l) = (a+2)/3$, where a may be chosen arbitrarily.

Example 3.3.8. *Consider another instance which is identical to the one described in Example 3.3.7, but for the following arc lengths:* $c_{1,2} = c_{2,1} = b$, *where* $1 < b < a$.

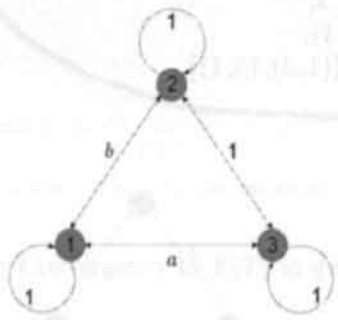

Figure 3.5: Example 3.3.8

It can be checked that in all cases discussed in Example 3.3.7 the same solutions are the optimal ones, but the objective function values are different. More specifically,

- *For the transportation problem* T *it holds that* $F(T) = 3$, $x^T_{i,j} = 0$ *for* $i \neq j$ *and* $x^T_{i,j} = 1$ *for* $i = j$. *The collection* $\mathcal{C}$ *as output by Convert-ToSequence in this case is* $\mathcal{C} = \{(1,(1,1)),(1,(2,2)),(1,(3,3))\}$.

- *For the problem* $P(1)$ *we have* $F(P) = a + b + 1$. *An optimal solution for this problem is* $x^1_{i,j} = 1$ *for* $(i,j) \in \{(1,2),(2,3),(3,1)\}$ *and* $x^1_{i,j} = 0$ *for any other arcs. In this case* $\mathcal{C}$ *is* $\{(1,((1,2),(2,3),(3,1)))\}$.

- *For any natural number* $l \geq 2$ *we get* $F(l) = (3l + 2b - 2)/l = 3 + (2b - 2)/l$. *An optimal solution is again specified by* $x^l_{1,2} = x^l_{2,3} = x^l_{3,2} = x^l_{2,1} = 1$; $x^l_{1,1} = x^l_{3,3} = l - 1$; $x^l_{3,1} = x^l_{1,3} = 0$ *and* $x^l_{2,2} = l - 2$. *Further,* $\mathcal{C} = \{((1,(1,2)(2,3),(3,2),(2,1)),((l-1),(1,1)),((l-2),(2,2)),((l-1),(3,3)))\}$.

In this example, $F(l)$ *strictly decreases when* l *increases, and indeed there does not exist* l^0 *such that* $F(l^0) = F(T)$. *Nevertheless, the optimal solution displays the same (stable) behavior as in the previous example.*

3.4 Optimal sequences in the stable case

Definition 3.4.1. *Consider an instance of the problem for which there exists a number $l^0 \in \mathbb{N}$ such that $F(l) > F(T)$ for any $l < l^0$ and $F(l) = F(T)$ for any $l \geq l^0$. Such an instance is called stable and l^0 is called the stabilization number.*

Theorem 3.4.2. *For every stable instance, the stabilization number $l^0 \leq (s+1)^2/4$.*

Proof. Consider a stable instance and corresponding stabilization number l^0. Let $G = (J, A)$ be a complete directed multigraph on the set of nodes J and with sufficiently large arc multipliers (for the purposes of this chapter we can take the multiplier of $(i,j) \in A$ equal to $l^0 \min\{n_i, n_j\}$). In G let us consider a closed walk $W^0 = ((i_1,i_2),(i_2,i_3),\ldots,(i_t,i_1))$, $i_\tau \in J$, $\tau \in \{1,2,\ldots,t\}$, which represents an optimal solution for the problem $P(l^0)$. Recall that any closed walk which represents a feasible solution of the problem $P(l)$ passes through the city $j \in J$ exactly $l \times n_j$ times.

In addition, let $W = ((j_1,j_2),(j_2,j_3),\ldots,(j_w,j_1))$, $j_\tau \in J$, $\tau \in \{1,2,\ldots,w\}$ be any closed walk in the graph G. We say closed walk W is *complete* if it passes through all vertices in G.

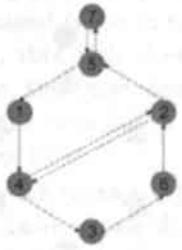

Figure 3.6: A complete closed walk

More formally, we define:

Definition 3.4.3. *Given a multigraph $G = (J, A)$ and a closed walk $W = ((j_1,j_2),(j_2,j_3),\ldots,(j_w,j_1))$, $j_\tau \in J$, $\tau \in \{1,2,\ldots,w\}$ in G. We say W is complete if for every $j \in J$ there exists $\tau \in \{1,2,\ldots,w\}$ such that $j_\tau = j$.*

Definition 3.4.4. *Given a multigraph $G = (J, A)$, we say closed walk $W = ((j_1, j_2), (j_2, j_3), \ldots, (j_w, j_1))$, $j_\tau \in J$, $\tau \in \{1, 2, \ldots, w\}$ in G is a minimal complete walk if for any closed walk $W' = ((j_{t_1}, j_{t_1+1}), (j_{t_1+1}, j_{t_1+2}), \ldots, (j_{t_1+t_2-1}, j_{t_1+t_2}))$, $j_\tau \in J$, $\tau \in \{t_1, t_1+1, \ldots, t_1+t_2\}$, $j_{t_1+t_2} = j_{t_1}$, $t_2 \geq 1$, contained as a subsequence in W, the closed walk $W \setminus W' = ((j_1, j_2), (j_2, j_3), \ldots, (j_{t_1-1}, j_{t_1}), (j_{t_1}, j_{t_1+t_2+1}), \ldots, (j_w, j_1))$ is not complete.*

Figure 3.7: A minimal complete closed walk

Definition 3.4.5. *If closed walk W' is contained as a subsequence in W and $W \setminus W'$ is complete, we call W' omissible.*

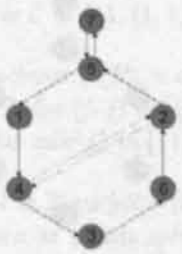

Figure 3.8: $((2,4),(4,2))$ is an omissible closed walk

Consider a minimal complete walk $W' = ((j_1, j_2), (j_2, j_3), \ldots, (j_t, j_{t+1}))$ where $j_1 = j_{t+1}$ which is obtained from the complete closed walk W^0 by removing omissible closed walks from it. We are going to prove that W' contains at most $(s+1)^2/4$ arcs. To see this, let f be the number of vertices that are passed in W' exactly once. More formally, f is the number of vertices

j for which there is exactly one arc $(j_\tau, j_{\tau+1}), \tau \in \{1, 2, \ldots, t\}$ in W' such that $j_\tau = j$.

First, we make clear that $f \geq 2$. Consider an arbitrary cycle C that is contained as a subsequence in W'. If no such cycle C exists, we are done since W' is complete and hence $f = s$. Thus, let us assume C exists. Then C (and also $W' \setminus C$) consists of at least two arcs, since otherwise it forms a loop, and therefore is omissible, which contradicts the minimality of W'.

Now, let C_1 be a cycle that is contained as a subsequence in C with the minimum number of arcs and let C_2 be defined similarly with respect to $W' \setminus C$. There are two cases for C_1.

Assume first that all vertices passed in the cycle C_1 are represented in W' more than once. If all vertices passed in C_1 are passed also in $W' \setminus C_1$ then C_1 is omissible contradicting the minimality of W'. Hence, at least one vertex must be passed more than once in C_1. Thus, C_1 contains a cycle. Again we have reached a contradiction since by definition of C_1 it does not contain any cycles as a subsequence. Thus, we conclude that there is at least one vertex passed in C_1 which is passed in W' only once. Applying the same reasoning to C_2, we conclude that $f \geq 2$.

Now, let i and j be two vertices that are passed once in W' and such that the part Q of the closed walk from i to j does not contain any other vertex that is passed once in W'. We will say that i and j are *neighboring*. It can be seen as follows that in Q no vertex is passed more than once. Indeed, if there exist a vertex that is passed twice in Q, then we have found a cycle. Since i and j are neighboring, this cycle does not contain vertices that are passed in W' just once. Hence, the cycle must contain as a subsequence an omissible cycle contradicting the minimality of W'. Thus, all vertices that are passed in Q are passed once in Q.

Now, we know that in the walk there are $f \geq 2$ vertices which are passed exactly once and that between pairs of neighboring vertices, no other vertices occur more than once. This allows us to bound the total number of vertices in W'. Indeed, it must hold that the total number of vertices and hence arcs in W' does not exceed $f + f(s-f)$. Maximizing $f + f(s-f)$ over $f = 2, \ldots, s$ yields $f_{\max} = (s+1)/2$. It then follows that $f_{\max} + f_{\max}(s - f_{\max}) = (s+1)^2/4$ and therefore W' has length at most $(s+1)^2/4$.

Using that the length t of W' is less than or equal to $(s+1)^2/4$, we now finish the proof by constructing a solution to $P(t)$ such that $F(t) = F(T)$.

Since we consider a stable instance, by definition $F(l^0) = F(T)$. Therefore the walk W^0 is also an optimal solution for the problem $T(l^0)$. Let $x^0_{i,j}$, $i \in$

J, $j \in J$, be the number of times arc (i,j) is travelled in W^0. Then, x^0/l^0 is a fractional optimal solution for problem T. Since the transportation problem has a totally unimodular constraint matrix, the polyhedron of the problem T is integral. By consequence, any fractional optimal solution can be represented as a convex combination of integer optimal solutions. Hence, we have that for every arc (i,j) contained in the walk W^0, there exists an integer optimal solution for T for which $x_{ij} > 0$.

Now, let us consider the walk W^0 and a corresponding minimal complete closed walk W'. By our observation above, for every arc $(i,j) \in W'$ there exists a solution x^{ij} of the problem T such that $x^{ij}_{ij} \geq 1$. Let x' be obtained by summing all $x^{ij}, (i,j) \in W'$. Since W' is a complete walk, x' satisfies the subtour elimination constraints (3.4), and hence it is an optimal solution to $P(t)$.

Since x' is obtained by summing t solutions for problem T, x' is a solution for $P(t)$. Therefore $F(t) = F(T)$ and thus $t \leq l^0$. It then follows that $l^0 \leq t \leq (s+1)^2/4$ as required. □

With Theorem 3.4.2 at hand, we are now able to address the computational complexity issue. Despite the fact that even for fixed l the decision version of $P(l)$ is a generalization of the TSP and therefore NP-complete, we have the following theorem:

Theorem 3.4.6. *The problem of deciding whether an instance is stable can be solved in polynomial time.*

Proof. Firstly, we are going to check for each arc $(i',j') \in J \times J$ whether there exists an optimal solution of T that contains this arc. This can be achieved by solving a modification of T, where the modification consists of decreasing $n_{i'}$ by one in the outflow constraint (3.3) for city i', and decreasing $n_{j'}$ by one in the inflow constraint (3.2) for city j':

$$F(T_{i'j'}) = \min_{x} \sum_{i \in J} \sum_{j \in J} c_{ij} x_{ij} \tag{3.27}$$

subject to

$$\sum_{i \in J} x_{ij} = n_j, \quad j \in J \setminus \{j'\}; \tag{3.28}$$

$$\sum_{j \in J} x_{ij} = n_i, \quad i \in J \setminus \{i'\}; \tag{3.29}$$

$$\sum_{i \in J} x_{ij'} = n_{j'} - 1; \tag{3.30}$$

$$\sum_{j \in J} x_{i'j} = n_{i'} - 1; \tag{3.31}$$

$$x_{ij} \in \mathbb{Z}^+, \quad i \in J,\ j \in J. \tag{3.32}$$

Let us call this modified problem $T_{i'j'}$ and denote the optimal objective value by $F(T_{i'j'})$. Obviously, $T_{i'j'}$ is in turn a transportation problem that can be solved in polynomial time. The aforementioned optimal solution of T containing (i', j') exists if and only if $F(T) = F(T_{i'j'}) + c_{i'j'}$.

We subsequently construct a directed multigraph $G = (J, A)$ as follows. For all (i', j') such that $F(T) = F(T_{i'j'}) + c_{i'j'}$, let $x^{i'j'}$ be obtained by slightly modifying an optimal solution x' of $T_{i'j'}$. The modification consists of increasing of the (i', j')-component by one: $x^{i'j'}_{i'j'} = x'_{i'j'} + 1$ if $(i, j) = (i', j')$ and $x^{i'j'}_{ij} = x'_{ij}$ otherwise. Now, the arc set A of G is obtained as follows. For all (i, j), the multiplicity of arc (i, j) is defined to be the sum of the $x^{i'j'}_{ij}$ over all (i', j') such that $F(T) = F(T_{i'j'}) + c_{i'j'}$.

Now, if G is strongly connected then it contains a Eulerian tour (see, e.g., [30]), i.e., a closed walk visiting every arc in this directed multigraph exactly once. Clearly this tour visits every vertex as well as every arc and hence satisfies the subtour elimination constraints. But then this Eulerian tour contains a solution to $P(l)$ for some $l \in \mathbb{N}$ with value $l \times F(T)/l = F(T)$.

Conversely, suppose that G is not strongly connected. Thus, there exists a subset $J' \subset J$, $J' \neq \emptyset$, such that G does not contain any arc (i, j) such that $i \in J', j \in J \setminus J'$. Now, for contradiction, assume that there is some number l such that $F(l) = F(T)$ and let x^l be an optimal solution to $P(l)$. Define $y^l_{ij} = x^l_{ij}/l$, $(i, j) \in J \times J$. Note, that y^l is an optimal solution for T. Notice also, that x^l is a feasible solution to $P(l)$ and hence satisfies the subtour elimination constraints. Then for J' there must exist an arc (i, j) such that $i \in J', j \in J \setminus J'$, and $y^l_{ij} > 0$. Then, in some optimal solution z^l of the transportation problem $T(l)$ it holds that $z^l_{ij} > 0$. By total unimodularity, there must exists an optimal solution z^0 for T in which z^0_{ij} is a positive integer. But then, by definition of A, it holds that $(i, j) \in A$. Since arc (i, j) connects J' and $J \setminus J'$ we have arrived at a contradiction.

Thus, we conclude that an instance I is stable if and only if the directed multigraph G is strongly connected. As described above, G can be constructed in polynomial time. Further, it is easy to check in polynomial time

whether G is strongly connected or not (see, e.g., [3]), which completes the proof. □

We finish this section by giving an example showing that the analysis for the bound in Theorem 3.4.2 can not be improved.

Example 3.4.7. *Consider the following instance.* $J = \{1,2,3,4,5,6\}$, *and*

$$[c_{i,j}]_{s\times s} = \begin{pmatrix} a & a & a & +1 & a & a \\ a & a & a & a & +1 & a \\ a & a & a & a & a & +1 \\ a & a & +1 & +1 & +1 & +1 \\ +1 & a & a & +1 & +1 & +1 \\ a & +1 & a & +1 & +1 & +1 \end{pmatrix}$$

where $a > 1$, $[n_i]_s = (1,1,1,3,3,3)$. *This instance has the minimal complete closed walk and also the optimal sequence* $[1,4,5,6,2,5,6,4,3,6,4,5]$:

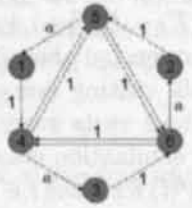

Figure 3.9: Closed walk in Example 3.4.7

Here $(s+1)^2/4 = (6+1)^2/4 = 49/4 \simeq 12$ which is equal to the length of the given minimal complete closed walk.

It is straightforward to generalize Example 3.4.7, in particular the distance matrix C to the case where $J = \{1,2,\ldots,s\}$ and a corresponding minimal complete closed walk W' of length $(s+1)^2/4$ exists that entails the optimum solution value. Let it be noted, however, that this does not mean that every optimal solution necessarily results in a minimal closed walk of length $(s+1)^2/4$. Indeed we shall see in Section 3.5, that a better bound can be obtained when using linear programming techniques. Nevertheless, if we only encode solutions as sequences and consider resulting closed walks, the bound in Theorem 3.4.2 is tight.

3.5 Optimal solutions in the general case

For further investigations it will be convenient to define for the given vector $\delta \in (\mathbb{Z}^+)^s$ the following transportation problem $T(l, \delta)$ and its optimal objective value $F(T(l, \delta))$:

Definition 3.5.1.

$$F(T(l,\delta)) = \frac{1}{l} \min_{x} \sum_{i \in J} \sum_{j \in J} c_{i,j} x_{i,j} \tag{3.33}$$

subject to

$$\sum_{i \in J} x_{i,j} = l \times n_j - \delta_j, \quad j \in J; \tag{3.34}$$

$$\sum_{j \in J} x_{i,j} = l \times n_i - \delta_i, \quad i \in J; \tag{3.35}$$

$$x_{i,j} \in \mathbb{Z}^+, \quad i \in J,\ j \in J. \tag{3.36}$$

Theorem 3.5.2. *Consider an instance of the problem such that the following holds*

- $F(s-1)$ *is finite;*
- *The problem* $T(s-1, \alpha)$ *has a finite optimal solution where the* i*-th component of the vector* $\alpha \in (\mathbb{Z}^+)^s$ *represents the number of times city* i *is passed in a minimal complete closed walk obtained from an optimal solution of the problem* $P(s-1)$ *(see Theorem 3.4.2).*

Then, for any natural number $l \geq s-1$ *the following holds:*

$$F(l+1) = \frac{l}{l+1} F(l) + \frac{1}{l+1} F(T), \tag{3.37}$$

and

$$F(l) \;=\; F(s-1)\frac{s-1}{l} \;+\; F(T)\frac{l-s+1}{l}, \tag{3.38}$$

and the solution $x^l = x^{s-1} + (l-s+1)x^T$ *is an optimal solution of the problem* $P(l)$*, where* x^{s-1} *and* x^T *are optimal solutions of* $P(s-1)$ *and* T *respectively.*

Proof. To prove the theorem we have to show that equation (3.37) holds. Equation (3.38) then follows by induction as in Theorem 3.3.5.

To prove equation (3.37), we show that $(l+1)F(l+1) \geq lF(l) + F(T)$ for any natural number $l \geq s-1$. Together with Theorem 3.3.1 this yields the desired result.

Consider an optimal solution x^0 of the problem $P(l+1)$, $l \geq s-1$. As before, let W^0 be the corresponding closed walk, and let W' be a minimal complete closed walk obtained from W^0 by deleting omissible cycles. Let α be the vector such that α_i, $i \in J$, is the number of times city i is passed in W'. It can be seen that since W' is minimal and complete, $1 \leq \alpha_i \leq s-1$ for all $i \in J$.

Now, consider a solution x^l of $P(l)$ which is constructed as follows: $x^l = x^{W'} + x^{T(l,\alpha)}$ where $x^{W'}_{i,j}$, $i \in J$, $j \in J$, equals the number of times arc (i,j) occurs in W' and $x^{T(l,\alpha)}$ is a solution of the problem $T(l,\alpha)$. Since $l \times n_i \geq (s-1) \times n_i \geq s-1 \geq \alpha_i$ for any index $i \in J$, we may conclude that W' is a feasible complete tour for the problem $P(l)$. Thus, the solution x^l is feasible for $P(l)$ and the transportation problem $T(l,\alpha)$ is correctly defined with strictly positive right hand-side coefficients $l \times k - \alpha$. Since x^l is a feasible solution of $P(l)$, we have

$$
\begin{aligned}
& l \times F(l) + F(T) \\
= \; & l \times \frac{1}{l} \sum_{i \in J} \sum_{j \in J} c^l_{i,j} x^l_{i,j} + F(T) \\
= \; & l \times \frac{1}{l} \left(\sum_{(i,j) \in W'} c_{i,j} x^{W'}_{i,j} + \sum_{i \in J} \sum_{j \in J} c_{i,j} x^{T(l,\alpha)}_{i,j} \right) + F(T) \\
\leq \; & \sum_{(i,j) \in W'} c_{i,j} x^{W'}_{i,j} + l \times F(T(l,\alpha)) + F(T) \\
= \; & \sum_{(i,j) \in W'} c_{i,j} x^{W'}_{i,j} + (l+1) \times F(T(l+1,\alpha)) + l \times F(T(l,\alpha)) + F(T) \\
& -(l+1) \times F(T(l+1,\alpha)) \\
= \; & (l+1)F(l+1) + l \times F(T(l,\alpha)) + F(T) - (l+1) \times F(T(l+1,\alpha)).
\end{aligned}
$$

Hence, we can prove equation (3.37) by proving

$$l \times F(T(l,\alpha)) + F(T) = (l+1) \times F(T(l+1,\alpha)) \text{ for any } l \geq s-1. \quad (3.39)$$

Since the polyhedron of the transportation problem is integer,we can assume without loss of generality that $F(T)$, $F(T(l,\alpha))$ and $F(T(l+1,\alpha))$ are linear programs. Further, changing notation from x to x/l yields that instead of proving that $l \times F(T(l,\alpha)) + F(T) = (l+1) \times F(T(l+1,\alpha))$ it suffices to prove that $l \times F(T(1,\alpha/l)) + F(T) = (l+1) \times F(T(1,\alpha/(l+1)))$. The reader is encouraged to verify that the three linear programs $T(1,\alpha/l), T, T(1,\alpha/(l+1))$ only differ in the right hand sides. Let it be noted also that T is by definition identical to $T(1,0)$, and hence $F(T)$ equals $F(T(1,0))$.

Let us denote the dual of $T(1,\alpha/l)$ by $DT(1,\alpha/l)$. It can be described as follows:

Definition 3.5.3.

$$F(DT(1,\frac{\alpha}{l})) = \max_{u,v} \left(\sum_{i \in J} (n_i - \frac{\alpha_i}{l}) u_i + \sum_{j \in J} (n_j - \frac{\alpha_j}{l}) v_j \right) \tag{3.40}$$

subject to

$$u_i + v_j \leq c_{i,j}, \quad i \in J,\ j \in J. \tag{3.41}$$

We simply write DT for $DT(1,0)$, and indeed DT is the dual of T. Since the three problems $T(1,\alpha/l), T, T(1,\alpha/(l+1))$ only differ in the right hand sides, the three problems $DT(1,\alpha/l), DT, DT(1,\alpha/(l+1))$ only differ in the coefficients of the objective function.

Now, suppose that despite the differences in the coefficients of the objective function, some optimal solution (u^*, v^*) of DT is also optimal for $DT(1,\alpha/l)$ and $DT(1,\alpha/(l+1))$. Then, by strong duality, we have

$$
\begin{aligned}
& (l+1)F(T(l+1,\alpha)) \\
= \; & (l+1) \times F(T(1,\frac{\alpha}{l+1})) \\
= \; & (l+1) \times \left(\sum_{i \in J} (n_i - \frac{\alpha_i}{l+1}) u_i^* + \sum_{j \in J} (n_j - \frac{\alpha_j}{l+1}) v_j^* \right) \\
= \; & \sum_{i \in J} ((l+1) \times n_i - \alpha_i) u_i^* + \sum_{j \in J} ((l+1) \times n_j - \alpha_j) v_j^* \\
= \; & \sum_{i \in J} (l \times n_i - \alpha_i) u_i^* + \sum_{j \in J} (l \times n_j - \alpha_j) v_j^* + \sum_{i \in J} n_i u_i^* + \sum_{j \in J} n_j v_j^* \\
= \; & l \times \left(\sum_{i \in J} (n_i - \frac{\alpha_i}{l}) u_i^* + \sum_{j \in J} (n_j - \frac{\alpha_j}{l}) v_j^* \right) + \left(\sum_{i \in J} n_i u_i^* + \sum_{j \in J} n_j v_j^* \right) \\
= \; & l \times F(T(1,\frac{\alpha}{l})) + F(T) \\
= \; & l \times F(T(l,\alpha)) + F(T),
\end{aligned}
$$

which proves equation (3.39) and hence, it proves also equation (3.37) as required to prove the theorem.

Thus, we have to show that for l large enough, or more specifically $l \geq s-1$, the differences in the coefficients of the objective function between the problems DT, $DT(1,\alpha/l)$ and $DT(1,\alpha/(l+1))$ are small enough to yield a solution (u^*,v^*) that is optimal for all three of DT, $DT(1,\alpha/l)$ and $DT(1,\alpha/(l+1))$. Such a solution (u^*,v^*) can be found as follows. Consider the polyhedron of DT. Let us make some small modifications to the objective function, and hence to the objective hyperplane. Assume that these modifications are obtained by adjusting the orthogonal vector by a small vector σ. It is clear that if an optimal face of the polyhedron is bounded in the σ-direction, then we preserve at least one optimal point of this face. By the second condition of the theorem, namely the fact that $F(T(s-1,\alpha))$ is finite, it follows that in the direction α the optimal face is bounded. Thus, with a small adjustment of the orthogonal vector in the α-direction, we keep at least one dual optimal solution (u^*,v^*) of T for the problems $DT(1,\alpha/l)$ for any sufficiently big number $l \in \mathbb{N}$. The only question remains whether $l = s-1$ is big enough or not.

For the formal proof of the equality (3.39), we use the following theorem from sensitivity analysis:

Theorem 3.5.4. Partition Perturbation Theorem *(see, [1] or [42]). Consider linear program* $LP(b) = \min\{cx|x \geq 0, Ax \geq b\}$ *and its dual* $DLP(b) = \max\{\pi b|\pi \geq 0, \pi A \leq c\}$. *Suppose* (x^0, π^0) *is a pair of strictly complementary solutions for* $LP(b)$ *and* $DLP(b)$ *respectively, and* γ *is an admissible direction, i.e.,* $LP(b + \theta\gamma)$ *has an optimal solution for some* $\theta > 0$. *Define the differential linear program:* $\Delta(b) = \max\{\pi\gamma|\pi$ *is optimal solution of* $DLP(b)\}$. *Then there exists* $\theta^* > 0$ *such that the following holds true for* $\theta \in [0, \theta^*]$*: the optimal partition (basic/non-basic rows/columns) for* $LP(b+\theta\gamma)$ *or, equivalently,* $DLP(b+\theta\gamma)$ *is the same as the optimal partition for* $\Delta(b)$.

By the conditions of Theorem 3.5.2, $F(T(s-1,\alpha))$ and hence $F(T(1,\alpha/(s-1)))$ are finite and therefore (α,α) is an admissible direction.

Let (u^*, v^*) be the optimal solution of DT that is also optimal for $\Delta(b) = \max\{(u,v)^T(\alpha,\alpha)|(u,v)$ is an optimal solution of DT$\}$. Then, by the Partition Perturbation Theorem, there exists $\theta^* > 0$ such that for any $\theta \in [0,\theta^*]$ the optimal partition of $T(1,\theta\alpha)$ (and hence $DT(1,\theta\alpha)$) is the same as the optimal partition for T (and DT). This in turn implies that the optimal dual solution (u^*, v^*) of DT is also optimal for $DT(1,\theta\alpha)$ for any $\theta \in [0,\theta^*]$. By consequence, we can complete the proof by showing that $\theta^* \geq 1/l$.

To estimate θ^* for a general linear program LP, the following approach is proposed in [87]. Consider the system of linear equations $A_B z = \gamma$ where A_B denotes the submatrix of A whose columns correspond to the indices from $B = \{n|x_n > 0\}$. In [87], the authors list the following two mutually exclusive cases:

- If the system has no solution, that is γ is not in the range of A_B, then there is no number $\theta > 0$ that preserves the optimal partition.
- Assume that the system has a solution, say z, then $\theta^* = +\infty$ if $z \leq 0$ and $\theta^* \geq \min\{x_n/z_n|n \in B, z_n > 0\}$ otherwise.

Since we have chosen the conditions of the theorem such that an admissible direction exists, we are clearly in the second case. Thus, there exists a solution z^* of the system $A_B z = (\alpha,\alpha)$. Letting x^* be an optimal solution to T, the value $\min\{x^*_{i,j}/z^*_{i,j}|x^*_{i,j} > 0; z^*_{i,j} > 0\}$ bounds θ^* from above. Since the polyhedron of P is integral $x^*_{i,j} \geq 1$ if $x^*_{i,j} > 0$. Hence, $\theta^* \geq 1/Z^*$, where $Z^* = \max\{z^*_{i,j}|z_{i,j} > 0\}$. Thus, it remains to bound $\max\{z^*_{i,j}|z_{i,j} > 0\}$.

Let us remind that A_B is obtained from the constraint matrix of T by elimination of all columns corresponding to variables $x^*_{i,j}$ for which $x^*_{i,j} = 0$. The problem $A_B z = (\alpha, \alpha)$ can be represented as the following problem on a graph. Given a graph $G = (J, E)$ where $E = \{(i,j) \in J \times J |\ x^*_{i,j} > 0\}$. The problem consists in finding a circulation flow such that every city $j \in J$ has inflow and outflow equal to α_j. Negative components of the vector z are allowed, hence we may disregard the orientation of the $x^*_{i,j}$ in the edges. Consider the circulation flow z^*. Since every city $j \in J$ of the graph G has inflow and outflow of α_j, we conclude that $z_{i,j} \leq \alpha_j$ for all i, j. Hence, $z_{i,j} \leq \alpha_j \leq s-1$ for all $i, j \in J$, and thus $Z^* \leq \max_{j \in J} \alpha_j$. But then, $\theta^* \geq 1/\max_{j \in J} \alpha_j \geq 1/(s-1)$, and therefore for any $l \geq s-1$, as stated in the theorem, $\theta^* \geq 1/l$ and the proof is complete. □

Corollary 3.5.5. *For every instance of $P(l)$, $l \geq s-1$, with finite input data*

$$F(l) = F(s-1)\frac{s-1}{l} + F(T)\frac{l-s+1}{l}$$

and the solution $x^l = x^{s-1} + (l-s+1)x^T$ is an optimal solution for $P(l)$.

Proof. All conditions of Theorem 3.5.2 are satisfied when the input data are finite. Indeed, if the objective coefficients $c_{i,j}$, $i \in J$, $j \in J$, and multiplicity coefficients n_j, $j \in J$, are finite, then the values $F(s-1)$ and $F(T(s-1,\alpha))$ are finite. □

Notice that Corollary 3.5.5 provides an approximation preserving approach for construction of a good solution for problem $P(l)$ with big multiplicity parameter $l \geq s-1$:

Corollary 3.5.6. *Let $\mathcal{A}$ be a polynomial time δ-approximation algorithm for the problem $P(s-1)$ and $F^{\mathcal{A}}$ be an objective value of solution provided by $\mathcal{A}$. Then for any $l \geq s-1$, there exists a polynomial time δ-approximation algorithm for $P(l)$.*

Proof. Let us remind that $\mathcal{A}$ is a δ-approximation algorithm if

$$\frac{F^{\mathcal{A}} - F(s-1)}{F(s-1)} \leq \delta.$$

Now, consider an arbitrary $l \geq s-1$. By Corollary 3.5.5,

$$\begin{aligned} F(l) &= F(s-1)\frac{s-1}{l} + F(T)\frac{l-s+1}{l} \\ &\geq \frac{F^{\mathcal{A}}}{1+\delta}\frac{s-1}{l} + F(T)\frac{l-s+1}{l} \\ &> \frac{1}{1+\delta}\left(F^{\mathcal{A}}\frac{s-1}{l} + F(T)\frac{l-s+1}{l}\right). \end{aligned}$$

Consider solution $x^{APPX} = x^{\mathcal{A}} + (l-s+1)x^T$ of $P(l)$ where $x^{\mathcal{A}}$ is a solution provided by $\mathcal{A}$. Clearly, x^{APPX} is feasible solution for $P(l)$ and its objective value equals

$$F^{APPX}(l) = F^{\mathcal{A}}\frac{s-1}{l} + F(T)\frac{l-s+1}{l}. \tag{3.42}$$

Therefore

$$F(l) > \frac{1}{1+\delta}F^{APPX}(l)$$

or equivalently

$$\frac{F^{APPX}(l) - F(l)}{F(l)} < \delta,$$

which completes the proof. □

Moreover, one can easily see that the approach provided by Corollary 3.5.5 is not only approximation preserving but also asymptotically optimal with respect to multiplicity parameter l:

Corollary 3.5.7.

$$\lim_{l \to +\infty} F^{APPX}(l) = F(T).$$

Proof. The statement straightforwardly follows from equality (3.42) and the fact that $F(T)$ is a lower bound on any feasible solution of any problem $P(l)$, $l \in \mathbb{N}$. □

Now, as we announced in Section 3.4, we are going to improve the upper bound on the *stabilization number* l^0:

Corollary 3.5.8. *If an instance is stable then the stabilization number* l^0 *is such that* $l^0 \leq s-1$, *and this bound is tight.*

Proof. According to Theorem 3.5.2, it holds for $l \geq s$ that $l \times F(l) = (l-1) \times F(l-1) + F(T)$ or equivalently $F(l-1) = (l \times F(l) - F(T))/(l-1)$. By stability, we have that there exists l such that $F(l) = F(T)$. Substituting $F(T)$ for $F(l)$ in the above equality we get $F(l-1) = F(T)$. Since by Theorem 3.5.2 the equality holds for for any $l \geq s$ we have $F(s-1) = F(T)$ and immediately $l^0 \leq s-1$.

It remains to show that there exist instances for which $l^0 = s-1$. Consider the following straightforward extension of Example 3.3.7.

Example 3.5.9. *Let $J = \{1, 2, \ldots, s\}$, $n_i = 1$ for any $i \in J$ and the distance matrix is*

$$[c_{i,j}]_{s \times s} = \begin{pmatrix} 1 & 1 & 1 & 1 & \ldots & 1 & 1 \\ 1 & 1 & a & a & \ldots & a & a \\ 1 & a & 1 & a & \ldots & a & a \\ \vdots & & & \ddots & & & \\ 1 & a & \ldots & a & 1 & a & a \\ 1 & a & \ldots & a & a & 1 & a \\ 1 & a & \ldots & a & a & a & 1 \end{pmatrix}$$

where $a > 1$.

- *For the transportation problem $F(T) = s$, $x^T_{i,j} = 0$ for $i \neq j$ and $x^T_{i,j} = 1$ for $i = j$. An optimal collection of cycles is*

$$\mathcal{C} = \{(1,(1,1)), (1,(2,2)), \ldots, (1,(s,s))\}.$$

- *For $l = 1$, an optimal solution is given by $x_{i,i+1} = 1$ for $i = 1, 2, \ldots, s-1$, $x_{s,0} = 1$, and $x_{i,j} = 0$, $i, j \in J$ otherwise. Thus, the value of an optimal solution is in this case $F(1) = 1 + (s-2)a + 1 = 2 + (s-2)a$.*

- *For $l = 2$, an optimal solution is given by $x_{1,2} = x_{2,2} = x_{2,1} = 1$, $x_{1,3} = 1$, $x_{i,i+1} = 1$ for $i = 3, 4, \ldots, s-1$, and $x_{s,0} = 1$, $x_{i,i} = 1$, $i = 3, 4, \ldots, s$, and $x_{i,j} = 0$, $i, j \in J$ otherwise. Thus, in this case the value of an optimal solution equals $F(2) = (3+1+(s-3)a+1+(s-2))/2 = ((s-3)a + s + 3)/2$,*

- *For any $1 < l < s-1$ an optimal solution $x^l_{i,j}$ $(i \in J,\ j \in J)$ is*

$$[x^l_{i,j}]_{s\times s} = \begin{pmatrix} 1 & 1 & \dots & 1 & 1^l & 0 & \dots & 0 & 0 \\ 1 & l-1 & & 0 & 0^l & 0 & \dots & 0 & 0 \\ \vdots & & \ddots & & \vdots & & \dots & & \vdots \\ 1 & 0 & & l-1 & 0^l & 0 & \dots & 0 & 0 \\ 0^l & 0^l & \dots & 0^l & l-1^l & 1^l & & 0^l & 0^l \\ 0 & 0 & \dots & 0 & 0^l & l-1 & \ddots & 0 & 0 \\ \vdots & & \dots & & \vdots & & \ddots & \ddots & \\ 0 & 0 & \dots & 0 & 0^l & 0 & & l-1 & 1 \\ 1 & 0 & \dots & 0 & 0^l & 0 & \dots & 0 & l-1 \end{pmatrix}$$

where the "l" superscripts the l-th row and l-th column of the matrix.

The value of this optimal solution is

$$\begin{aligned} F(l) &= \frac{(l-1)2+(l-1)(l-1)+1+(s-(l+1))a+(s-l)(l-1)+1}{l} \\ &= \frac{(l-1)(s+1)+2+(s-l-1)a}{l}. \end{aligned}$$

- *For $l = s-1$ we get $F(s-1) = s = F(T)$. An optimal solution is*

$$[x^{s-1}_{i,j}]_{s\times s} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & l-1 & & \\ \vdots & & \ddots & \\ 1 & & & l-1 \end{pmatrix}$$

Thus, for this instance $l^0 \le s-1$. We finish the proof by showing that $l^0 \ge s-1$. Indeed, for $l = s-2$ and $s \ge 3$,

$$\begin{aligned} F(s-2) &= \frac{((s-2)-1)(s+1)+2+(s-(s-2)-1)a}{s-2} \\ &= \frac{(s-3)(s+1)+2+a}{s-2} \\ &= \frac{s^2-2s-1+a}{s-2} \\ &= s+\frac{a-1}{s-2} \\ &> s, \end{aligned}$$

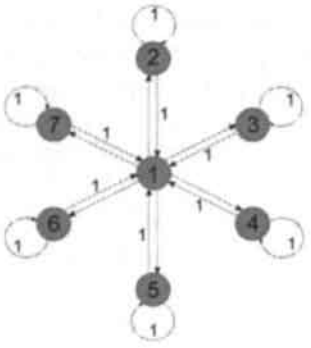

Figure 3.10: Structure of the optimal walk in Example 3.5.9 for $l \geq s-1$

which yields $l^0 > s-2$ *as required.* □

Example 3.5.9 also proves a final statement of this section:

Corollary 3.5.10. *There exist instances for which* $F(s-2) > F(s-1)$.

3.6 Summary and conclusions

This chapter deals with a very general high multiplicity sequencing problem, namely the high multiplicity travelling salesman problem (HMTSP). Our research is motivated by the applicability of the HMTSP to a variety of scheduling problems. Lately, high multiplicity scheduling problems have been widely studied, but very little is known about the impact of scheduling the jobs as specified by a minimal part set, as opposed to the more general scheduling problem of scheduling jobs in non-minimal, optimal, part sets. This chapter investigates how the cycle time, or length of a HMTSP tour varies with the multiplicities in the minimal part set. In particular, we investigate the behavior of the optimal solution, and the optimal solution value, when all quantities in the minimal part set are multiplied by a factor l. We show that the optimal solution value decreases when l increases. Moreover, we show that the ratio between the value of an optimal solution for $l = 1$ and for an arbitrary l can grow proportional to l.

Further, we have investigated whether there exists a finite l for which the optimal value cannot be improved upon. We have given a polynomial procedure that solves this problem in Section 3.4. In the same section we show that, if there exists finite l for which the optimal value is minimal, then this l can be bounded from above by $(s+1)^2/4$. In fact, we have shown that, if we restrict our encoding schemes to sequences which explicitly specify an order in which the cities are to be visited, then the bound $(s+1)^2/4$ can not be improved. However, in Section 3.5 we show that if the optimal solution is given in terms of a solution to an integer program, this bound can be improved to be $s-1$. Section 3.5 also shows that even if there is no finite l for which the optimal solution value is lowest attainable, the differences between the lengths of cyclic optimal schedules for l and $l+1$ are identical for all $l \geq s-1$.

The bounds on l establish the importance of encoding schemes for solutions, especially in the case of sequencing problems. Indeed, the results make clear that if the optimal solution is given in terms of an explicit sequence, then the tightest bound on l that can be derived is $O(s^2)$, whereas this bound can be improved to $O(s)$ when using a more compact encoding scheme, which is based upon another integer programming formulation.

The linear bound obtained in this chapter applies to all high multiplicity scheduling problems that can be formulated as a HMTSP. It will be interesting to see whether this result, as well as the ratio analysis, can be improved upon for specific problems.

Finally, we conjecture that the bound of Theorem 3.5.2 can be slightly improved:

Conjecture 3.6.1. *Theorem 3.5.2 holds even for* $l \geq (s-1)/\min_{i \in J} n_i$.

Chapter 4

Modelling and solving the periodic maintenance problem

4.1 Introduction

The planning and scheduling of preventive maintenance activities is often crucial for the cost-effectiveness of many large industrial organizations. For instance, manufacturing organizations that have highly sophisticated and complex machinery have long recognized that efforts spent on preventive maintenance can contribute significantly towards an efficient running of the organization. Also in service organizations (like medical facilities or governmental institutions), preventive maintenance is regarded as an important activity that can help to reach the organization's strategic goals. However, the costs associated with preventive maintenance can be significant: there are not only costs involved in the maintenance itself, also the costs of production losses during the maintenance have to be taken into account. Computerized Maintenance Management Systems (CMMS's) are becoming increasingly popular as a tool to increase machine-availability and more generally, to improve control over the maintenance activities. Software vendors (see, for instance, http://www.plant-maintenance.com/index.shtml) offer packages that usually include a scheduling module that suggests (among other things) when to service which unit (or machine). This decision is seen as a re-occurring event, i.e., it is expected that a schedule is of a cyclic nature, and hence will be executed repeatedly.

There is an extensive literature on preventive maintenance, see, e.g., [71, 102, 108]. However, approaches in literature usually are of a stochastic nature

where a probability distribution is used to describe the failure properties of some part (see, for instance, Gertsbakh and Gertsbakh [35]). In this work we take a different, completely deterministic, approach (see Giglio, Glaser, and Wagner, [37] for an early reference). More specifically, we deal with the problem of cyclically scheduling maintenance activities under a certain given cost-structure assuming a fixed cycle length. A precise description is given in the next subsection.

4.1.1 Problem description

We consider the following problem. There are a number of machines $M_i, i \in M = \{1, 2, \ldots, m\}$, and there is a time-interval $T = \{1, 2, \ldots, \mathrm{T}\}$ with $\mathrm{T} \geq m$. During each period of the time-interval T, at most one machine can be serviced. When machine M_i is serviced, a given, nonnegative, servicing cost of b_i is incurred, regardless of the period. A machine M_i that is not serviced during some period is in operation and incurs an operation cost of $j_i(t) \times a_i$, where a_i is a given positive integer, and where $j_i(t)$ is the number of periods elapsed since last servicing machine M_i, $i \in \{1, 2, \ldots, m\}$. Observe that we assume here that the operating costs of a machine increase linearly with the number of periods elapsed since last servicing that machine. The problem is now to determine a maintenance schedule, i.e., to decide for each period $t \in T$ which machine to service (if any), such that total servicing costs and operating costs are minimized.

There are good reasons to view such problems in a cyclic context. In such a context, it is assumed that the maintenance schedule will be executed repeatedly. Thus, in period $k \times \mathrm{T} + t$, $(k \in \mathbb{N},\ t \in T)$, the same machine that was serviced in period t will be serviced again. In addition, the cost will be considered in this infinite horizon context. Consequently, the cost of a maintenance schedule is calculated by summing over all $t \in T$ the total of the servicing costs incurred in period t and the operating costs incurred by machines which are not serviced in period t. These operating costs are defined in a cyclic context, i.e., the last maintenance service may lie in a previous execution of the maintenance schedule. We will refer to this problem as the *Periodic Maintenance Problem (PMP)*. Notice that in an optimal solution to PMP, each machine receives service at least once. Finally, we notice here explicitly that in PMP, T is considered to be an input parameter.

For ease of understanding, we now present a brief example.

Example 4.1.1. *Let* $T = 7$, $m = 3$ *and the set of machines is* $\{1,2,3\}$. *Further, let* $b_i = 1, i = 1,2,3$ *and let* $a_1 = a_2 = 10$ *and* $a_3 = 1$. *Consider the solution* $(1,2,1,2,1,2,3)$. *This sequence of maintenance services is to be read as follows: in the first period, we service machine 1, in the second period machine 2, et cetera, until we service in the seventh period machine 3. Then, this sequence of maintenance services is repeated, i.e., in the 8-th period we service machine 1 again, followed by machine 2 in period 9, and so on. The cost of this solution can be computed as follows. Since there is maintenance in each of the seven periods of* T, *and since all service costs* b_i *are equal to one, the total servicing costs equal 7. For the first machine the operating costs are incurred in periods 2, 4, 6 and 7. In periods 2, 4 and 6, these costs equal 10, and in period 7 these costs amount to 20. Thus, machine 1 has a total operating cost of 50. Similarly, it can be checked that machine 2 has operating costs of 20+0+10+0+10+0+10=50, and machine 3 has operating costs of 1+2+3+4+5+6=21. Thus, the total cost for this solution is 128. The reader can verify that the solution presented above is in fact optimal.*

Apart from the application sketched above, PMP and variants of PMP have real-life applications of various origins such as scheduling of maintenance services, multi-item replenishment of stock, and broadcasting of data messages over a communication channel. In particular, the problem where the cycle length is not given, but is instead a decision variable, has received much attention (see the references in Section 4.2). In the remainder, we refer to the variant of PMP where T is considered to be a decision variable, as the *Free Periodic Maintenance Problem (FPMP)*; we use T^* to denote the optimal cycle length in FPMP.

Our motivation for investigating PMP, rather than FPMP, is twofold. First of all, PMP is a practical problem. Especially in the context of constructing maintenance schedules, it is very natural to fix the cycle length to some constant such as 365, 52, 30, 7, 24 or 60. Indeed, an organization that implements a cyclic maintenance schedule will, for reasons of simplicity, ensure that the length of the cyclic schedule coincides with the size of a natural time-interval such as the number of days per year, or the number of weeks per year, or the number of days per week. Further, in many practical settings, it is desirable that the cycle length T is not too large. In fact, even for instances of modest size, for example, $m = 2$, $a_1 = 1$, $a_2 = a$, $b_1 = b_2 = 0$, the optimal cycle length T^* can be fairly large, for this case, see [7] where it is proved that $T^* \geq \lfloor \sqrt{2a} \rfloor$. Thus, one is interested in computing a cyclic

schedule with a cycle length that is bounded from above by some reasonably small (given) integer B. In such a case, one can find the optimal value of $T \leq B$ by solving the PMP for each possible value of T not exceeding B. In both cases, the task is to find a solution of some specific cycle length that may differ from the optimal length T^*. As far as we are aware, the PMP has not been studied before.

A second motivation for our work is that we are interested in solving instances of the problem to optimality. As we shall see in Section 4.2, apart from [7], most research has focused on complexity results, and approximation for FPMP. From this point of view, we further explore the area of solving instances to optimality by solving them for a fixed, but not necessarily optimal, T. In addition, our results provide insight regarding the effect of varying T on the actual schedule and its solution, i.e., we investigate the sensitivity of the solution with respect to the cycle length.

This chapter is organized as follows. In the next section we present a brief literature review. Section 4.3 discusses several models, and how they might be of use in solving the problem to optimality. Section 4.4 presents a branch-and-price algorithm that solves one of the models of Section 4.3 to optimality. In Section 4.5 we explain good performance of the branch-and-price algorithm showing the good quality of the lower bound used in this algorithm. In Section 4.5 we also introduce a randomized polynomial time approximation algorithm for PMP. In Section 4.6 we present computational results on instances with three to ten machines and with a number of periods ranging from three till one hundred. Section 4.7 contains the conclusions.

4.2 Literature review

Anily, Glass and Hassin [7] consider a special case of FPMP in a machine maintenance context, where $b_i = 0$ for all $i \in M$. They prove that there exists an optimal schedule that is cyclic. Further, they describe a network-flow based algorithm that has exponential complexity to solve the problem exactly. This approach allows them to solve instances with up to four machines exactly. In addition, the authors propose two lower bounds and a greedy heuristic, which performs very well. Notice, however, that in their problem setting, the cycle length is a decision variable, and therefore the solutions given by the heuristic may use a different cycle length than the cycle length of an optimal solution. The case with three machine and no maintenance costs is investigated in Anily, Glass and Hassin [6]. In this work

the authors introduced an algorithm solving certain instances of the problem to optimality and for the other instances they present a polynomial time approximation algorithm with a performance ratio of 1.0333.

Bar-Noy et al. [9], and Kenyon, Schabanel and Young [68] consider a generalized version of the FPMP where in each period at most M machines can be serviced. Their interest in the problem is motivated by applications that arise in broadcast scheduling. Bar-Noy et al. prove that FPMP is NP-hard. Further, they investigate lower bounds and propose a $\frac{9}{8}$-approximation algorithm. Kenyon, Schabanel and Young [68] present a polynomial-time approximation scheme for FPMP with bounded service costs. The version of the problem with non-identical service times is studied in Kenyon and Schabanel [67]. Recently, Schabanel [100] shows that the version of FPMP in which preemptions are allowed, is also NP-hard.

Brakerski et al. [11] consider the problem of encoding a solution in such a way that the next machine to be serviced can always be found quickly, given that all service activities performed up till now are known. Brauner et al. [13] address related scheduling problems that arises from compact encodings of solutions. Work on preventive maintenance using linear programming is described in De Ghellinck, Smeers and Souissi [36].

Let us briefly examine the PMP from a complexity viewpoint. First of all, notice that the input to PMP consists of $2m+1$ numbers (the a_i, b_i and T). Thus, an algorithm which has the parameter T present in its running-time is not a polynomial-time algorithm for PMP. In fact, all models we present in this chapter have (at least) a pseudo-polynomial number of variables. Second, the reduction in [9] shows that FPMP is NP-hard even when T^* is known. This implies indeed that PMP is NP-hard as well, since it may be the case that $T = T^*$.

4.3 Modelling PMP

In this section we describe three formulations for PMP. Subsection 4.3.1 gives a quadratic programming formulation, Subsection 4.3.2 describes an integer programming based formulation, and Subsection 4.3.3 presents a set partioning formulation.

4.3.1 A quadratic programming formulation

Here we introduce a compact and natural, but non-convex quadratic program modeling PMP with operational costs only, i.e., we first assume $b_i = 0$ for all $i \in M$. The model uses a variable $x_{i,t} \in \mathbb{Z}^+$, $i \in M$, $t \in T$, which represents the number of periods between the current period $t \in T$ and the last period before t when machine i has been serviced. Clearly, for any machine i, and any period t, the value of variable $x_{i,t+1}$ is obtained by either adding 1 to the value of $x_{i,t}$, or by setting it to 0. Setting the value of $x_{i,t+1}$ to 0 corresponds to servicing machine i in period $t+1$. PMP can now be formulated as follows:

$$\min_{x} \sum_{i \in M} \sum_{t \in T} a_i x_{i,t} \tag{4.1}$$

$$x_{i,t+1}(x_{i,t+1} - x_{i,t} - 1) = 0, \quad i \in M,\ t \in T \setminus \mathrm{T}; \tag{4.2}$$

$$x_{i,1}(x_{i,1} - x_{i,\mathrm{T}} - 1) = 0, \quad i \in M; \tag{4.3}$$

$$x_{i,t} + x_{k,t} \geq 1, \quad i \neq k,\ i \in M,\ k \in M, t \in T; \tag{4.4}$$

$$x_{i,t} \in \mathbb{Z}^+, \quad i \in M,\ t \in T. \tag{4.5}$$

Equations (4.2) and (4.3) ensure the required behavior of the $x_{i,t}$ variables, i.e., if $x_{i,t+1} \neq 0$ then $x_{i,t+1} = x_{i,t} + 1$. Equations (4.4) imply that no two machines can be served simultaneously. Notice that if for some machine i one of the associated variables is integral, (4.2) and (4.3) together imply that all other variables corresponding to machine i are integral as well.

Since most of the available software for solving quadratic programming problems only solve convex quadratic programs, we have not been able to solve problem instances through the formulation given above. Instead, we now linearize model (4.1)–(4.5) and take into account the servicing costs b_i:

$$\min_{x,y} \sum_{i \in M} \sum_{t \in T} (a_i x_{i,t} + b_i y_{i,t}) \tag{4.6}$$

$$x_{i,t+1} \geq x_{i,t} + 1 - N y_{i,t+1}, \quad i \in M,\ t \in T \setminus \mathrm{T}; \tag{4.7}$$

$$x_{i,1} \geq x_{i,\mathrm{T}} + 1 - N y_{i,1}, \quad i \in M; \tag{4.8}$$

$$\sum_{i \in M} y_{i,t} \leq 1, \quad t \in T; \tag{4.9}$$

$$x_{i,t} \in \mathbb{Z}^+, \quad i \in M,\ t \in T; \tag{4.10}$$

$$y_{i,t} \in \{0,1\}, \quad i \in M,\ t \in T, \tag{4.11}$$

where N is a sufficiently big number.

The binary variable $y_{i,t}$ simply takes on value 1 if we service the i-th machine in period t and 0 otherwise. The objective (4.6) minimizes the total costs that now consist of operating costs and servicing costs. The equations (4.7) and (4.8) enforce the variables $x_{i,t}$ to behave in the same way as in the previous model. According to (4.9) we cannot service more than one machine in a single period. Restrictions (4.10) and (4.11) are the integrality constraints. We refer to the formulation (4.6)–(4.11) as QP (for quadratic program).

We illustrate model (4.6)–(4.11) with the following example.

Example 4.3.1. *Let* T $= 7$, $m = 3$ *and the set of machines is* $\{1, 2, 3\}$. *A feasible solution of the formulation is depicted in Table 4.3.1.*

Period ($t \in T$):	1	2	3	4	5	6	7
Sequence of maintenance services (machines):	1	3	1	2	1	3	2
$y_{1,t}$ (service indicator):	1	0	1	0	1	0	0
$y_{2,t}$ (service indicator):	0	0	0	1	0	0	1
$y_{3,t}$ (service indicator):	0	1	0	0	0	1	0
$x_{1,t}$ (state):	0	1	0	1	0	1	2
$x_{2,t}$ (state):	1	2	3	0	1	2	0
$x_{3,t}$ (state):	2	0	1	2	3	0	1

Table 4.1: A feasible solution

Notice that formulation (4.6)–(4.11) involves a so-called *big N parameter* which renders the associated linear relaxation to be rather poor. For instance, by setting $y_{i,t} = 1/m$ and $x_{i,t} = 0$, $i \in M$, $t \in T$, we satisfy all constraints of the linear relaxation. The value of the objective function of this solution to the linear relaxation is equal to $\mathrm{T} \sum_{i \in M} b_i/m$, which is an arbitrary bad lower bound on the optimum. This explains the poor computational performance we obtained using the standard ILP-packages dealing with formulation (4.6)–(4.11), see Section 4.6.3.

Another weak point of this formulation is that we use the fact that the objective is to *minimize* the total operating and servicing costs. This means

that not every solution that satisfies (4.7)–(4.11) is a meaningful solution to PMP. Thus, to solve the problem under maximization or mixed min-max criteria we cannot use the linear model described above. Section 4.8 describes a somewhat unusual model that contains a complete description of the set of feasible solutions.

4.3.2 An integer programming formulation

We now present a formulation that contains $O(m \times \mathrm{T}^2)$ binary variables. We introduce a variable $x_i^{s,t}, i \in M, s, t \in T$, whose value equals 1 if machine i is serviced in period s, and serviced next (cyclically) in period $t+1$, and 0 otherwise. Notice that when s is the last service in T, we have that $t \leq s$, because of the cyclicity of the maintenance schedule. Using costs $c(s,t)$ defined as follows:

$$c(s,t) = \begin{cases} \frac{(t-s)(t-s+1)}{2} & \text{if } s \leq t; \\ \\ \frac{(T-s+t)(T-s+t+1)}{2} & \text{if } s > t, \end{cases}$$

the problem can be modeled as follows:

$$\min_{x} \sum_{i \in M} \sum_{s \in T} \sum_{t \in T} \left(a_i c(s,t) x_i^{s,t} + b_i x_i^{s,t} \right) \tag{4.12}$$

subject to

$$\sum_{i \in M} \sum_{s \in T} x_i^{s,t} \leq 1, \quad t \in T; \tag{4.13}$$

$$\sum_{s \in T} x_i^{s,t} = \sum_{s \in T} x_i^{t+1,s}, \quad i \in M, t \in T \setminus \mathrm{T}; \tag{4.14}$$

$$\sum_{s \in T} x_i^{s,T} = \sum_{s \in T} x_i^{1,s}, \quad i \in M; \tag{4.15}$$

$$\sum_{s \in T} \sum_{t \in T} x_i^{s,t} \geq 1, \quad i \in M; \tag{4.16}$$

$$x_i^{s,t} \in \{0,1\}, \quad i \in M, s \in T, t \in T. \tag{4.17}$$

Inequalities (4.13) express that in each period at most one machine can be serviced, equalities (4.14)–(4.15) imply that there is a next period in which a

machine will be serviced, inequalities (4.16) say that each machine is serviced at least once, and finally (4.17) are the integrality constraints.

Again, the LP relaxation of this formulation is rather poor. For example, setting $x_1^{t,t+1} = x_2^{t,t+1} = \ldots = x_m^{t,t+1} = \frac{1}{m}$ for all $t \in T \setminus \mathrm{T}$, $x_i^{\mathrm{T},1} = \frac{1}{m}$ for all $i \in M$, and all other variables equal to 0, yields a feasible solution with zero operating costs. Notice how this solution resembles the example demonstrating the poor behavior of the LP relaxation of (4.6)–(4.11). The LP relaxation is strengthened considerably when we replace (4.16) by the following constraints (for the validity of the inequalities see Theorem 4.3.2 below):

$$\sum_{s \le u} \sum_{t < s} x_i^{s,t} + \sum_{s \le u} \sum_{t \ge u} x_i^{s,t} + \sum_{t \ge u} \sum_{s > t} x_i^{s,t} = 1, \text{ for all } i \in M,\ 1 < u < \mathrm{T}; \quad (4.18)$$

$$\sum_{s > 1} \sum_{t < s} x_i^{s,t} + \sum_{s \le \mathrm{T}} x_i^{s,T} = 1, \text{ for all } i \in M; \quad (4.19)$$

$$\sum_{t < T} \sum_{s > t} x_i^{s,t} + \sum_{t \ge 1} x_i^{1,t} = 1, \text{ for all } i \in M. \quad (4.20)$$

Equalities (4.18)–(4.20) state that for every machine and for every period u, the sum of the variables corresponding to pairs (s,t) that contain period u, is one. Obviously, this rules out the solution given above. We refer to the strengthened formulation (4.12)–(4.15), (4.17) and (4.18)–(4.20) as the *flow formulation* (FF). In Section 4.6 we provide computational results, which show that FF yields promising computational results when solving it using state of the art standard software CPLEX 7.5.

We finish the section showing that the introduced equations (4.18)–(4.20) are valid inequalities for the integer linear program (4.12)–(4.17):

Theorem 4.3.2. *Equalities (4.18)–(4.20) are valid inequalities for any optimal solution of the integer linear program (4.12)–(4.17).*

Proof. We shall prove the validity of the constraint (4.18). The proof for the constraints (4.20) and (4.19) can be done in the similar way.

Consider any $1 < u < \mathrm{T}$. By inequality (4.16) and integrality constraint (4.17) for any machine $i \in M$ there exists a pair (s,t) such that $x_i^{s,t} = 1$. Allowing the cyclic intervals (i.e., the intervals starting at s, passing through T and then 1, and finishing in $t < s$), we notice that if $u \notin [s,t]$, then by flow conservation constraints (4.14) and (4.15) there must be a pair (s',t') such

that $x_i^{s',t'} = 1$ and $u \in [s', t']$. Thus, for any $1 < u < \mathrm{T}$ there is a pair (s', t') such that $x_i^{s',t'} = 1$ and $u \in [s', t']$. It implies

$$\sum_{s \le u} \sum_{t < s} x_i^{s,t} + \sum_{s \le u} \sum_{t \ge u} x_i^{s,t} + \sum_{t \ge u} \sum_{s > t} x_i^{s,t} \ge x_i^{s',t'} = 1. \tag{4.21}$$

Now, assume that there are at least two different pairs (s_1, t_1) and (s_2, t_2) such that $x_i^{s_1,t_1} = x_i^{s_2,t_2} = 1$ and $u \in [s_1, t_1]$ and $u \in [s_2, t_2]$. Then by the flow capacity constraint (4.13) there must be at least two different flows for the i-machine service. Assume that in the first flow i-machine is serviced at time moments $s_1^1 \le s_2^1 \le \ldots \le s_k^1$, $k < \mathrm{T}$, and in the second flow it is serviced at $s_1^2 \le s_2^2 \le \ldots \le s_l^2$, $l < \mathrm{T}$. Since these two flows are different there are no $1 \le i \le k$ and $1 \le j \le l$ such that $s_i = s_j$. Consider a new service sequence containing all time units from both flows, say $s_1' \le s_2' \le \ldots \le s_{k+l}'$. Since the number of i-machine services stays the same, the total maintenance cost also stays the same. On the other hand, since the intervals between the sequential services become smaller, the total operating cost may only decrease. Therefore, among the optimal solutions of the problem (4.12)–(4.17) we can consider only solutions with a unique flow. It implies that we can restrict ourselves to the case with only one pair (s, t) such that $x_i^{s,t} = 1$ and $u \in [s, t]$. Thus,

$$\sum_{s \le u} \sum_{t < s} x_i^{s,t} + \sum_{s \le u} \sum_{t \ge u} x_i^{s,t} + \sum_{t \ge u} \sum_{s > t} x_i^{s,t} \le 1, \tag{4.22}$$

that together with inequality (4.21) completes the proof. □

4.3.3 A set partitioning formulation

Yet another formulation, related to the previous one, concludes this modelling section.

Let S be the set of all nonempty subsets of T. Clearly, every $s \in S$ is a possible set of periods for servicing a machine $i \in M$. Let us call $s \in S$ a *service strategy* or simply *strategy*. For every pair consisting of a machine $i \in M$ and a strategy $s \in S$, we can compute the cost $c_{i,t}$ incurred when servicing machine i in the periods contained in s as follows: let p_s be the cardinality of s and let q_j, $j \in \{1, 2, \ldots, p_s\}$, be the distances between neighboring services in s. For example, if $\mathrm{T} = 7$ and $s = \{2, 4, 6\}$ then $p_s = 3$

and $q_1 = 4-2 = 2$, $q_2 = 6-4 = 2$, $q_3 = 7-6+2 = 3$. The total service and operating cost associated with machine $i \in M$ and strategy $s \in S$ is

$$c_{i,s} = b_i p_s + a_i \sum_{j=1}^{p_s} (q_j - 1) q_j / 2.$$

So, in the example above the total costs of servicing machine i using strategy s is $c_{i,s} = 3b_i + a_i + a_i + 3a_i = 3b_i + 5a_i$.

Now, we introduce a variable $x_{i,s}$ which has value 1 if machine $i \in M$ is serviced in the periods contained in $s \in S$, and 0 otherwise. This allows for the following Set Partitioning formulation (SP):

$$\min_x \sum_{i \in M} \sum_{s \in S} c_{i,s} x_{i,s} \tag{4.23}$$

subject to

$$\sum_{s \in S} x_{i,s} = 1, \qquad i \in M; \tag{4.24}$$

$$\sum_{i \in M} \sum_{s \in S : t \in s} x_{i,s} \leq 1, \qquad t \in T; \tag{4.25}$$

$$x_{i,s} \in \{0,1\}, \qquad i \in M,\ s \in S. \tag{4.26}$$

Constraints (4.24) imply that one strategy has to be selected for each machine, and constraints (4.25) ensure that no two strategies make use of the same period. Constraints (4.26) are the integrality constraints. Despite the exponential size of this integer linear program it has two very important properties. First, its linear relaxation (obtained by replacing (4.26) by $x_{i,s} \geq 0$ for all i, s) is solvable in time polynomial in m and T (see Section 4.4). Second, computational experiments show that the linear relaxation of this integer problem is quite strong. In the next section we show how to solve SP using a branch-and-price algorithm.

We conclude this section by showing that the LP relaxations of SP is stronger than LP relaxation of FF.

Theorem 4.3.3. *Let $v(FFLP)$, $v(SPLP)$ be optimal solutions of linear relaxations of FF and SP respectively, then $v(FFLP) \leq v(SPLP)$.*

Proof. Let $x^* = \{x_{i,s} : i \in M;\ s \in S\}$ be any solution to the LP relaxation of SP. Construct a solution $y^* = \{y_i^{u,v} : i \in M;\ u, v \in T\}$ to the relaxed FF as follows. For every $x_{i,s}$, and (u, v) where $u, v \in s$ and there is no $t \in s$ such that $u \leq t \leq v$, we set $y_i^{u,v} = x_{i,s}$. In addition, for every $x_{i,s}$, and (u, v) where u is the element in s with highest index, and v is the element in s with smallest index, we set $y_i^{u,v} = x_{i,s}$.

Now, let us show that this solution is feasible. The solution y^* satisfies the flow conservation constraints (4.14)–(4.15) from its construction. Similarly, constraint (4.25) and the feasibility of x^* implies that (4.13) is satisfied. Further, it follows from constraint (4.24) and the construction of y^* that (4.18)–(4.20) is satisfied. We leave it to the reader to verify that the objective function values of x^* and y^* are equal.

Thus, any solution of the LP relaxation of SP can be converted to a corresponding solution of LP relaxation of FF, that completes the proof. □

4.4 A branch-and-price algorithm for PMP

In this section we show how to solve SP using branch-and-price. In Subsection 4.4.1 we show how column generation can be used to solve the LP relaxation of (4.23)–(4.26) without enumerating all variables $x_{i,s}$. Next, in Subsection 4.4.2 we propose a branching scheme that keeps the structure of the problem intact. We refer to Barnhart et al. [10] for a general description of branch-and-price algorithms.

4.4.1 Column generation algorithm

The linear relaxation of SP, called $SPLP$, is obtained by replacing constraints (4.26) by $x_{i,s} \geq 0$ for all $i \in M$, $s \in S$. The corresponding dual problem (called SPD) is

$$\max_{u,v} \left(\sum_{i \in M} u_i + \sum_{t \in T} v_t \right) \tag{4.27}$$

subject to

$$u_i + \sum_{t \in s} v_t \leq c_{i,s}, \qquad i \in M,\ s \in S; \tag{4.28}$$

$$v_t \leq 0, \qquad t \in T. \tag{4.29}$$

The column generation procedure starts with finding a feasible solution for $SPLP$. To do that we can use, for example, a trivial integer solution where in the first T periods we service all machines one by one, and for all remaining periods we service only the machine with the largest coefficient a_i. So, in an initialization step, we generate the set of pairs $N = \{(i, s_i) : i \in M\}$ where s_i is the set of periods when we service machine $i \in M$. Let us restrict the column set of $SPLP$ to N and let us call the problems restricted to N as $SPLP(N)$ and $SPD(N)$ respectively.

Next, we find an optimal solution for $SPLP(N)$ and $SPD(N)$ using an LP-solver. Thus, we obtain a primal-dual pair of solutions $(x(N), (u(N), v(N))$. We can extend $x(N)$ to a solution of $SPLP$ by setting the remaining variables to zero. Establishing whether or not this extended solution is optimal for $SPLP$ can be done by analyzing the corresponding dual solution $(u(N), v(N))$. Optimality of $x(N)$ for $SPLP$ depends on the feasibility of $(u(N), v(N))$ in SPD. To verify whether all dual constraints are satisfied we have to solve the following pricing problem:

$$\text{Price: } \exists\, i \in M, s \in S \text{ such that } u_i + \sum_{t \in s} v_t > c_{i,s}? \tag{4.30}$$

If the dual solution $(u(N), v(N))$ satisfies all constraints of SPD, then $x(N)$ extended with zeros is an optimal solution of $SPLP$. If not, then we have found - by solving the pricing problem - a machine i and a strategy s whose reduced costs (c_{is} minus the left-hand side of the inequality (4.30)) are negative. Thus, bringing this variable into the basis will contribute to the objective function's value. Then we update N by adding this variable to it, and we iterate. The efficiency of this procedure depends to a large extent on the speed with which the pricing problem can be solved. We have the following theorem:

Theorem 4.4.1. *The pricing problem can be solved in $O(m\mathrm{T}^3)$ time.*

Proof. We prove that for each i we need to solve an all-pairs shortest path problem on a directed graph with $O(\mathrm{T})$ nodes. Since this problem can be solved in $O(\mathrm{T}^3)$ using the Floyd-Warshall algorithm (see Ahuja et al. [3]), the result follows.

Thus, let us now consider a specific machine i, and let us build the following graph $G = (V, A)$ with $V = T$ and $A = \{(p, q) : \ p \leq q, \ p, q \in V\}$.

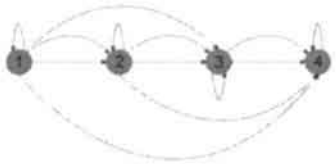

Figure 4.1: Graph G on 4 nodes

For each arc $(p,q) \in A$ we define the following costs $w(p,q)$:

$$w(p,q) = b_i + a_i \frac{(q-p)(q-p-1)}{2} - v_q \qquad \text{if} \quad p \neq q \quad \text{and}$$

$$w(p,p) = b_i + a_i \frac{\mathrm{T}(\mathrm{T}-1)}{2} - v_p.$$

This completes the construction of G. Notice that all costs w are nonnegative. Let us now establish a correspondence between a path P in G and a service strategy s for machine i. Indeed, consider any path $P = \{t_1, t_2, \ldots, t_k\}$ in G in case s contains $t_1, t_2, \ldots, t_k$, and loop (t_1, t_1) in case s consists only of t_1. We have the following

Claim: If there exists a loop (t_1, t_1) with cost less than $Q \equiv u_i$ or there exists a path in G from t_1 to t_k such that $t_1 \neq t_k$ with costs less than $Q \equiv u_i + v_{t_1} - b_i + a_i \sum_{t=t_k+1}^{T+t_1-1} (t - t_k)$ then the current solution is not optimal.

Argument: Consider the first case where there exists a loop (t_1, t_1) such that $w(t_1, t_1) < u_i$. Using definition of the costs we derive

$$w(t_1,t_1) = b_i + a_i \frac{\mathrm{T}(\mathrm{T}-1)}{2} - v_{t_1} = c_{i,s} - v_{t_1} = c_{is} - \sum_{t \in s} v_t < u_i.$$

Now, consider the second case where there exists a path in G from t_1 to t_k such that $t_1 \neq t_k$ with costs less than $Q \equiv u_i + v_{t_1} - b_i + a_i \sum_{t=t_k+1}^{T+t_1-1} (t - t_k)$. Notice that Q depends only on t_1 and t_k. Consider now the cost of a path $\{t_1, t_2, t_3, \ldots, t_{k-1}, t_k\}$ in V. Summing the appropriate coefficients w gives:

$$(k-1)b_i + a_i \sum_{l=1}^{k-1} \sum_{t=t_l+1}^{t_{l+1}-1} (t - t_l) - \sum_{l=2}^{k} v_{t_l}.$$

We now derive:

$$(k-1)b_i + a_i \sum_{l=1}^{k-1} \sum_{t=t_l+1}^{t_{l+1}-1} (t - t_l) - \sum_{l=2}^{k} v_{t_l} < Q \iff$$

$$kb_i + a_i \sum_{l=1}^{k} \sum_{t=t_l+1}^{t_{l+1}-1} (t - t_l) - \sum_{l=1}^{k} v_{t_l} < u_i \iff$$

$$c_{is} - \sum_{t \in s} v_t < u_i.$$

It follows that given the first and the last service period, computing a shortest path in G between the corresponding vertices determines whether there is a strategy to be added to the master problem. Hence, to solve the pricing problem for machine i we need to compute shortest paths between every pair of vertices in G. As mentioned above this can be done using Floyd-Warshall's algorithm in $O(\mathrm{T}^3)$ operations (see Ahuja et al. [3]) □

Corollary 4.4.2. *The problem SPLP can be solved in time polynomial in* m *and* T.

Proof. The proof of the corollary straightforwardly follows from Theorem 4.4.1 and the well-known theorem by Grötschel, Lovasz and Schrijver, see [47], stating

There exists a polynomial time algorithm for the separation problem for a family of polyhedra, if and only if there exists a polynomial time algorithm for the optimization problem for that family.

Since the pricing problem is nothing else but the separation problem for SPD we have that optimization problems SPD and $SPLP$ are solvable in time polynomial in m and T. □

In practice we did not use the approach by [47]. Instead of this we observe that the number of rows in SP ($SPLP$) is relatively small, so that we may try to apply a column generation algorithm to solve $SPLP$.

4.4.2 A branching scheme

To solve the original integer programming formulation SP let us introduce the following branching strategy. Notice that a traditional branching strategy that consists of setting a variable to 0 versus setting a variable to 1, would

not preserve the efficient solvability of the pricing problem (see Barnhart et al. [10]). Given a linear programming solution $x_{i,s}$, define $sum_i(t) = \sum_{s \in S: t \in s} x_{i,s}$ for $i \in M, t \in T$.

Lemma 4.4.3. *If the solution is fractional, i.e., if there exists a machine $i_0 \in M$ and a strategy $s \in S$ with $0 < x_{i_0,s} < 1$, then there exists a $t \in T$ such that $0 < sum_{i_0}(t) < 1$.*

Proof. Consider a machine $i_0 \in M$. Let $S(i_0)$ be the set of strategies s for which $0 < x_{i_0,s} < 1$. We say that strategy s_1 *contains* strategy s_2 if, for each period $t \in s_2$, we have that $t \in s_1$. Let $s_0 \in S(i_0)$ be a strategy that does not contain any other strategy from $S(i_0)$ (notice that such a strategy always exists). We argue by contradiction.

Assume that $S(i_0) \neq \emptyset$ and for all $t \in T$ the values $sum_{i_0}(t)$ are equal to either 0 or 1. This implies that $sum_{i_0}(t) = 1$ for all periods $t \in s_0$. Since, by constraint (4.25), $\sum_{s \in S} x_{i_0,s} = 1$, and since for each $t \in s_0$ we have that $sum_{i_0}(t) = \sum_{s \in S: t \in s} x_{i_0,s} = 1$, it follows that $x_{i_0,s} = 0$ for each strategy $s \in S$ that uses a period t not used by strategy s_0. Due to the fact that s_0 does not contain any strategy from $S(i_0)$, it follows that for each $s \in S(i_0) \setminus s_0$, there exists a period $t \in s$ such that $t \notin s_0$. Consequently, $x_{i_0,s} = 0$ for all $s \in S(i_0) \setminus s_0$, and $x_{i_0,s_0} = 1$, a contradiction to $S(i_0) \neq \emptyset$. □

Let us now describe how the branching scheme preserves the efficient pricing service strategies. Let the branching rule be simply to decide whether a period $t \in T$ is used in a service strategy for machine $i_0 \in M$ (branch 1) or not (branch 2). Considering branch 1, this has the following consequences for the pricing problem: each arc passing t, i.e., going from some $t_1 < t$ to some $t_2 > t$ is deleted from the graph and from now on for every child node of the branching tree machine i_0 is serviced in period t. Moreover, in the graphs associated with the other machines, we delete all arcs entering node t. So, no path will visit node t. Considering branch 2 is even easier: we simply delete from the graph all arcs entering t. Now, from Lemma 4.4.3 we can conclude that this branching strategy excludes the current fractional solution.

4.5 Integrality gap and approximation algorithms

4.5.1 Bounded integrality gap

In this subsection we show that in case $\mathrm{T} \geq 2m$ the integrality gap of SP is bounded from above by a constant.

First of all, let us construct a feasible maintenance schedule given a solution of $SPLP$. Suppose $x_{i,s}$, $i \in M$, $s \in S$, is an optimal solution of the linear program $SPLP$. Based on solution x for every machine $i \in M$ we define the probability:

$$p_i = \frac{\sum_{t \in T} \sum_{s \in S: t \in s} x_{i,s}}{\mathrm{T}}. \tag{4.31}$$

Note that, putting into consideration a new dummy-machine 0 with zero maintenance and operating costs, we can enforce constraint (4.25) to be tight, i.e.,

$$\sum_{i \in M} \sum_{s \in S: t \in s} x_{i,s} = 1, \quad t \in T. \tag{4.32}$$

This clearly implies that $\sum_{i \in M} p_i = 1$.

Now, at every odd time slot $2(i-1)+1$, $i \in M \setminus \{0\}$, we service machine i. Since we choose $\mathrm{T} \geq 2m$ and all non-dummy machines are serviced, this already gives us a feasible solution of SP. At all even time slots we service machines at random with probabilities p_i. Let Z_i, $i \in M$, be a random variable, which denotes the total cost incurred by machine i over interval T in the randomized schedule above and let us call the resulting schedule R.

Lemma 4.5.1. *Due to Bar-Noy et al, see* [9]*:*

$$EXP(Z_i) \leq 2\mathrm{T}a_i \frac{1-p_i}{p_i} + 2\mathrm{T}b_i p_i. \tag{4.33}$$

Proof. By linearity of expectations we can consider maintenance and operating costs separately. First, let us consider the maintenance cost. Let n_i be a random variable which is the number of maintenance services of machine i in schedule R. Notice that n_i is binomially distributed with parameter p_i for all even time-slots and has an additional service in the odd time slot $2(i-1)+1$. Thus, $EXP(n_i) = \lfloor \mathrm{T}/2 \rfloor p_i + 1 \leq \mathrm{T}p_i + 1$. By inequalities (4.24) and the

definition of probabilities p_i, $i \in M$, we have $1 \leq \mathrm{T}p_i$ for any $i \in M$, so $EXP(n_i) \leq 2\mathrm{T}b_i p_i$.

The result for expectation of the operating cost follows from Wald's identity [31]. Let $Y_i(j)$ be a random variable representing the operating cost occurring for the j-th pair of consecutive maintenances of the i-th machine in R. Here, we start counting the pairs from the first even time slot where we have maintenance of the i-th machine. We ignore the maintenance services in odd slots and assume that the upper bound on the last (n_i-th) maintenance interval is not limited by T but can be extended to any positive integer $\mathrm{T}' \geq \mathrm{T}$. Notice that these assumption can only increase the operating cost for the i-th machine, i.e.,

$$Oper.Cost_i \leq \sum_{j=1}^{n_i-1} Y_i(j).$$

Let us define an infinite set of random variables:

$$I_j = \begin{cases} 1 & \text{if } j = 1; \\ 1 & \text{if } j \geq 2 \text{ and } R \text{ contains at least } j \text{ maintenances of machine } i; \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, we can write

$$EXP(Oper.Cost_i) \leq EXP\left(\sum_{j=1}^{n_i-1} Y_i(j)\right) = EXP\left(\sum_{j=1}^{+\infty} I_j Y_i(j)\right).$$

Notice that I_j is dependent of $Y_i(1), Y_i(2), \ldots, Y_i(j-1)$ but it is independent of $Y_i(j), Y_i(j+1)$ and so on. Since $\sum_{j=1}^{+\infty} I_j Y_i(j)$ is finite and linearity of expectations it holds that

$$EXP\left(\sum_{j=1}^{+\infty} I_j Y_i(j)\right) = \sum_{j=1}^{+\infty} EXP\left(I_j Y_i(j)\right).$$

As we have noticed above I_j is independent of $Y_i(j)$. Therefore,

$$\sum_{j=1}^{+\infty} EXP\left(I_j Y_i(j)\right) = \sum_{j=1}^{+\infty} EXP(I_j) EXP(Y_i(j)).$$

Note that the $Y_i(j)$, $j \in \mathbb{N}$, are identically distributed random variables. Hence,

$$\begin{aligned}\sum_{j=1}^{+\infty} EXP(I_j)EXP(Y_i(j)) &= EXP(Y_i)\sum_{j=1}^{+\infty} EXP(I_j)\\ &= EXP(Y_i)EXP\left(\sum_{j=1}^{+\infty} I_j\right)\\ &= EXP(Y_i)EXP(n_i - 1)\\ &\leq EXP(Y_i)\,\frac{\mathrm{T}p_i}{2}.\end{aligned}$$

Notice that servicing machines only in the even time slots we have every two slots the operating cost increment of $4a_i$: if at time t we have paid operating cost $a_i s_t$ then at times $t+1$, $t+2$, $t+3$ we pay $a_i(s_t+1)$, $a_i(s_t+2)$, $a_i(s_t+3)$ respectively and the two time slots operating cost increment is $a_i(s_t+2) + a_i(s_t+3) - a_i(s_t+1) - a_i s_t = 4a_i$. Given the increment of $4a_i$, we can estimate the value $EXP(Y_i)$ via the geometric distribution with parameter p_i.

$$\begin{aligned}\frac{\mathrm{T}p_i}{2}\,EXP(Y_i) &\leq \frac{\mathrm{T}p_i}{2}(4a_i)\sum_{j=1}^{\infty}\frac{j(j-1)}{2}(1-p_i)^{j-1}p_i\\ &= 2\mathrm{T}p_i a_i\frac{(1-p_i)}{p_i^2}\\ &= 2\mathrm{T}a_i\frac{(1-p_i)}{p_i}.\end{aligned}$$

Finally, by linearity of expectations

$$\begin{aligned}EXP(Z_i) &= EXP(Maint.Cost_i) + EXP(Oper.Cost_i)\\ &\leq 2\mathrm{T}a_i\frac{1-p_i}{p_i} + 2\mathrm{T}b_i p_i,\end{aligned}$$

which completes the proof. □

To present an upper bound on the integrality gap we prove the following theorem.

Theorem 4.5.2.

$$EXP\left(\sum_{i\in M} Z_i\right) \leq 4v(SPLP). \tag{4.34}$$

Proof. For every $s \in S$ define $k_s = \sum_{t\in s} 1$ and notice that since S does not contain an empty set

$$\mathrm{T}p_i = \sum_{t\in T}\sum_{s\in S: t\in s} x_{i,s} = \sum_{s\in S} k_s x_{i,s} \geq 1, \quad i \in M. \tag{4.35}$$

By Lemma 4.5.1 and linearity of expectations

$$EXP\left(\sum_{i\in M} Z_i\right) = \sum_{i\in M} EXP(Z_i) \leq \sum_{i\in M}\left(2\mathrm{T}a_i\frac{1-p_i}{p_i} + 2\mathrm{T}b_i p_i\right).$$

By definition of probabilities p_i, $i \in M$, and definition of k_s, $s \in S$, this equals

$$\sum_{i\in M}\left(2(1-p_i)\mathrm{T}^2 a_i \frac{1}{\sum_{s\in S} k_s x_{i,s}} + 2\sum_{s\in S} b_i k_s x_{i,s}\right).$$

By Lemma 4.5.4 (see the end of this subsection), this is not more than

$$\begin{aligned}
&\sum_{i\in M}\left(2(1-p_i)\sum_{s\in S}\mathrm{T}^2 a_i \frac{x_{i,s}}{k_s} + 2\sum_{s\in S} b_i k_s x_{i,s}\right) \\
&= \sum_{i\in M}\left(4(1-p_i)\sum_{s\in S} a_i \frac{1}{2}\left(\frac{\mathrm{T}}{k_s}\right)^2 k_s x_{i,s} + 2\sum_{s\in S} b_i k_s x_{i,s}\right).
\end{aligned}$$

Note that in $SPLP$, for any machine $i \in M$, the total maintenance cost is $F_i^{MC} = \sum_{s\in S} b_i k_s x_{i,s}$, and the total operating cost is at least

$$F_i^{OC} \geq \sum_{s\in S}\left(a_i \frac{1}{2}\left(\frac{\mathrm{T}}{k_s}\right)^2 k_s - \frac{1}{2}\mathrm{T}a_i\right) x_{i,s}. \tag{4.36}$$

Inequality (4.36) is valid due to the fact that, given a number of services k_s, the *smooth* distribution of these services, where all the consecutive maintenances are equally distanced from each other, minimizes the operating cost,

see, e.g., ([7]). Thus,

$$\begin{aligned}
&\sum_{i\in M}\left(4(1-p_i)\sum_{s\in S}a_i\frac{1}{2}\left(\frac{\mathrm{T}}{k_s}\right)^2 k_s x_{i,s}+2\sum_{s\in S}b_i k_s x_{i,s}\right)\\
\leq\ &\sum_{i\in M}\left(4(1-p_i)F_i^{OC}\left(1+\frac{\sum_{s\in S}\mathrm{T}a_i x_{i,s}/2}{F_i^{OC}}\right)+2F_i^{MC}\right)\\
\leq\ &\sum_{i\in M}\left(4(1-p_i)F_i^{OC}\left(1+\frac{\sum_{s\in S}\mathrm{T}a_i x_{i,s}/2}{\sum_{s\in S}a_i\mathrm{T}^2x_{i,s}/2k_s-\sum_{s\in S}\mathrm{T}a_i x_{i,s}/2}\right)+2F_i^{MC}\right)\\
=\ &\sum_{i\in M}\left(4(1-p_i)F_i^{OC}\left(1+\frac{\sum_{s\in S}x_{i,s}}{\mathrm{T}\sum_{s\in S}x_{i,s}/k_s-\sum_{s\in S}x_{i,s}}\right)+2F_i^{MC}\right)\\
=\ &\sum_{i\in M}\left(4(1-p_i)F_i^{OC}\left(1+\frac{1}{\mathrm{T}\sum_{s\in S}x_{i,s}/k_s-1}\right)+2F_i^{MC}\right).
\end{aligned}$$

Using Lemma 4.5.4 we derive that this is at most

$$\sum_{i\in M}\left(4(1-p_i)F_i^{OC}\left(1+\frac{1}{\mathrm{T}/\sum_{s\in S}k_s x_{i,s}-1}\right)+2F_i^{MC}\right).$$

By equation (4.35) this equals

$$\begin{aligned}
&\sum_{i\in M}\left(4(1-p_i)F_i^{OC}\left(1+\frac{1}{1/p_i-1}\right)+2F_i^{MC}\right)\\
=\ &\sum_{i\in M}\left(4(1-p_i)F_i^{OC}\left(1+\frac{p_i}{1-p_i}\right)+2F_i^{MC}\right)\\
=\ &\sum_{i\in M}\left(4F_i^{OC}+2F_i^{MC}\right)\\
\leq\ &4v(SPLP),
\end{aligned}$$

which completes the proof. □

Now, let us show that based on the approach above the upper bound of 4 on the integrality gap can not be improved.

Theorem 4.5.3. *There exists an instance of the maintenance problem such that*

$$\lim_{\mathrm{T}\to\infty}\frac{EXP(\sum_{i\in M}Z_i)}{v(SPLP)}=4. \tag{4.37}$$

Proof. Take an instance of the maintenance problem with two machines, unit operating costs and no maintenance costs for both machines. It is not difficult to verify that $v(SPLP) = \mathrm{T}$ and $p_1 = p_2 = 1/2$. In expectation analysis (see the proof of Lemma 4.5.1) we derive $EXP(Z_i) = 2\mathrm{T} - o(\mathrm{T})$, $i \in \{1, 2\}$, which gives $EXP(Z_1 + Z_2) = 4\mathrm{T} - o(\mathrm{T})$, completing the proof. □

Lemma 4.5.4.

$$\frac{1}{\sum_{s \in S} k_s x_{i,s}} \leq \sum_{s \in S} \frac{x_{i,s}}{k_s}. \tag{4.38}$$

Proof. The inequality (4.5.4) holds if and only if

$$\left(\sum_{s \in S} k_s x_{i,s}\right)\left(\sum_{s \in S} \frac{x_{i,s}}{k_s}\right) \geq 1.$$

Notice

$$\left(\sum_{s \in S} k_s x_{i,s}\right)\left(\sum_{s \in S} \frac{x_{i,s}}{k_s}\right) = \sum_{s \in S} x_{i,s}^2 + \sum_{s \in S} \sum_{t \in S: t \neq s} \left(\frac{k_s}{n_t} + \frac{n_t}{k_s}\right) x_{i,s} x_{i,t}.$$

Since for any natural numbers k_s and n_t, it holds that $k_s/n_t + n_t/k_s \geq 2$, we derive

$$\begin{aligned} & \sum_{s \in S} x_{i,s}^2 + \sum_{s \in S} \sum_{t \in S: t \neq s} \left(\frac{k_s}{n_t} + \frac{n_t}{k_s}\right) x_{i,s} x_{i,t} \\ \geq & \sum_{s \in S} x_{i,s}^2 + \sum_{s \in S} \sum_{t \in S} 2 x_{i,s} x_{i,t} \\ = & \left(\sum_{s \in S} x_{i,s}\right)\left(\sum_{s \in S} x_{i,s}\right) = 1, \end{aligned}$$

which completes the proof. □

4.5.2 Deterministic approximation algorithm

In this subsection, instead of a derandomization of the randomized algorithm, we introduce a simple LP-based deterministic approximation algorithm for SP with the same performance ratio of 4. Given an optimal solution of the LP

relaxation of SP, take parameters p_i, $i \in M$, defined by equation (4.31). For every $i \in M$ find a number r_i such that $2^{r_i-1} < 1/p_i \leq 2^{r_i}$. Define $d_i = 2^{r_i}$. Order the set M by increasing d_is (without loss of generality we assume that $d_1 \geq d_2 \geq \ldots \geq d_m$). Schedule the maintenance services for machine 1 (the machine with the lowest number d_i) starting from time slot 1 in such a way that the distance between any two sequential services of that machine is $d_1 - 1$. So, we maintain machine 1 in every d_i time units. Similarly, schedule machine 2 (the machine with the second lowest number d_i) starting from the first non-occupied time slot in the slots distanced from each other on $d_2 - 1$, and so on. Since for every machine the distances between the sequential services are power-of-two numbers, at every time slot we schedule at most one machine. Since the distances between machine services are rounded up (to the nearest higher power-of-two number), and summation of all p_i's gives 1, we have enough space in interval $[1, T]$ to schedule all machines from M. Therefore, the resulting schedule is feasible. Now, using the same line of argument as in the proof of Theorem 4.5.2, we can show that the cost of the constructed feasible schedule is bounded from above also by $4v(SPLP)$. To explain this in a less formal way, notice that the distance rounded to the nearest higher power-of-two number can at most double the distances between the consecutive services of a machine. Doing this, the maintenance costs can be only decreased since we maintain machines less frequently. In turn, the operating costs can be increased no more than 4 times. Thus, the constructed schedule provides an integral solution with objective value of at most $4v(SPLP)$.

4.6 Computational results

In this section we present computational results for all LP models constructed in Section 4.3.

4.6.1 Technical details

All experimental results were obtained on computer AMD Athlon 2400 XP+/1GB RAM running Debian GNU/Linux 3.0 with kernel 2.4.18. All calculations were limited by 100000 branching nodes and by 10000 seconds CPU. To compute the optimal solutions for QP and FF we used the package ILOG OPL-Studio 3.5 using the CPLEX MIP Solver. The computational results for SP are obtained using aforementioned column

stances of PMP are easy to implement and easy to solve (the running time is small). We can observe also that the computation times for the other two formulations are much better than that of QP and quite similar to each other. In the remainder we concentrate on the solutions of SP and FF only.

4.6.4 The quality of the lower bound

Now, we focus on the general performance of the column generation algorithm for $SPLP$ and the branch-and-price algorithm for SP versus the LP based branch-and-bound algorithm that the CPLEX MIP Solver uses to solve FF. Again, we consider instances from [7] on four machines, without maintenance costs and the solution value is presented in terms of the average operating cost per period. As previously, we have chosen T to be the optimal schedule length as computed by [7]. The results are presented in Table 4.4.

Here, OPT is the value of the optimal solution of PMP instance. The computational results show that the lower bound provided by the linear programming relaxation is very good and can be obtained in very short time. Moreover, in most of the cases the solutions found for the relaxed problems are integer and therefore optimal for the original model. We observed also that the linear relaxation of SP is just slightly better than the linear relaxation of FF. Moreover, in general for both formulations it is required just few branching nodes to obtain an integer solution.

We observe also that even in case of positive integrality gap OPL-Studio can provide an integral solution for FF analyzing only the root of the search tree (see, for example, the instance $m = 4$, T $= 8$, $a = (10, 5, 5, 1), b = 0$, in Table 4.4). The reason for this is following. The OPL-Studio MIP-Solver is based on branch-and-cut algorithm creating a number of cuts (actually there are 9 types of different cuts) in every node of the search tree including the root. Sometimes it cuts the fractional solution right in the root.

Notice that for instance $m = 4$, $a = (30, 10, 10, 1)$, $b = (0, 0, 0, 0)$, at cycle length $T = 30$ we found solution with $OPT = 58.3333$ which is better than the solution of $OPT = 58.42$ reported in [7].

4.6.5 Symmetry

In order to further test the proposed solution approaches, we have composed symmetrical instances where $a_i = 1$ and $b_i = 0$ for all machines $i \in M$. Table 4.5 for $m = 3$ displays the computational results.

For all instances of this class, we considered the integrality gap between the optimal solution and the solution of the relaxed problems. For all these instances, the integrality gap in SP is zero and we conjecture that this holds for any numbers T and m. In contrast, $v(FFLP) = 3$ for all the instances.

We noticed also that in symmetric cases the algorithm based on SP performs better if we start with the *simple solution* as an initial set of columns in LP rather than with the *greedy solution*.

We conclude that these instances are rather difficult to solve, especially for the branch-and-price algorithm. Despite the fact that integrality gap for these instances is equal to zero, many branches are needed to prove optimality. Notice also that the computational results for the flow model are significantly better, despite the fact that it uses much more nodes in the search tree. Thus, we conclude that the column generation algorithm spends relatively much time on solving the relaxed instances.

4.6.6 Maintenance costs

Here we investigate the impact of introducing strictly positive maintenance costs. Table 4.6 shows that for different maintenance and fixed operating costs the computation time and the number of nodes in the search tree can vary significantly.

For the instances in Table 4.6 we used a part of the benchmarks from [7] on five machines. Since Anily et al did not report the optimal cycle length for $m = 5$, we took $T = 24$ for all the instances.

Since for all these instances the branch-and-price algorithm based on SP performs worse than a straightforward implementation of FF we do not report the results for SP.

4.6.7 Cases with many machines

Finally, we would like to investigate how the number of machines affects the performance of the algorithms. We took five instances with $m = 10$ introduced in [7] without maintenance costs and we define a relatively modest cycle length $T = 18$. In Table 4.7 we introduce the results.

For all the instances in that table we have noticed that the algorithm based on the SP formulation performs much better than OPL implementation of FF. For some instances, for example, with $a = (10, 10, 10, 10, 10, 10, 10, 10, 10, 1)$, we could not even solve the FF formulation in 12 hours. We

explain a better performance of the branch-and-price algorithm by the following reason: from the experiments we clearly see that in these cases the linear relaxation provided by SP is much stronger than the linear relaxation of FF.

4.7 Conclusions, open questions and further research

In this chapter we have proposed several models for a periodic maintenance scheduling problem that has applications in other areas as well. In contrast to prior research, our approach has been to fix the length of the period. This chapter proposes several natural mathematical programming formulations, most of which are integer linear programs. We have investigated the computational behavior of these problems when solving them by an LP based branch-and-cut algorithm. One of the formulations is a *set partitioning* formulation, that contains a number of variables that is exponential in the cycle length T. We have shown how this problem can be solved using a column generation approach, and how the pricing problem can be solved efficiently. This results in a branch-and-price algorithm. The computational results of this approach are comparable to the results obtained through another (*flow*) formulation.

From a theoretical viewpoint, this chapter indicates several directions for further research. The first issue concerns the quality of the LP relaxations of SP and FF. Is it possible to improve the bound of 4 on the ratio between the value of the optimal solution of SP and $v(SPLP)$? Is there an upper bound on the integrality gap for FF? Is it true that with $m = 2$ the integrality gap is always 0?

The second trend of questions concerns the randomized algorithm introduced in Section 4.5: how can we derandomize the algorithm? How can we adopt the randomized algorithm to the case with $T < 2m$? What if machine services will take place at all time slots (in the current version of the randomized algorithm we do not use the last $T/2 - m$ even slots which is almost a half of the interval T)?

From a more practical viewpoint, the chapter leaves open some questions about solving the LP relaxations. The computational results indicate that the column generation approach takes much more time to find an optimal solution for the LP relaxation of the *set partitioning* formulation, than it

takes CPLEX to find an optimal solution to the LP relaxation of FF. Thus, in order to improve the computational results, the following question arises. From all columns with negative reduced costs, which one(s) should be added?

Conversely, the branch-and-price algorithm usually requires less nodes in the search tree than the branch-and-cut algorithm based on the *flow formulation*. Thus, the question arises whether other branching strategies can accelerate the solution process for this formulation.

4.8 Complete description of feasible solutions

In this section we present yet another model. This model is somewhat unusual but provides a complete description of the feasible solutions. For simplicity we assume again that all maintenance costs are equal to 0 and we are dealing with operating costs only. Here, instead of a vector representation of the states ($x_{i,t},\ i \in M, t \in T$) that were used in Subsection 4.3.1, we use the number representation of the state. We introduce T variables $s_0, s_1, \ldots, s_T$, that are integer numbers in the T-base number system. Here, the number $s_t,\ t \in T$, corresponds to the service-state of the maintenance in the time-unit t. The idea behind this is simple: instead of the vector representation of the state $(x_{1,t}, x_{2,t}, \ldots, x_{m,t})$ we use the representation of the state like an integer number $s_t = x_{1,t}x_{2,t}\ldots x_{m,t}$ where the i-th digit (in the T-base number system) represents the number of time units after the last maintenance of i machine at time slot t. We call a number in the T-base number system by T-*number* and the cipher in it by *digit*. Clearly, T is the maximal possible state component for any machine.

The decision variables $y_{i,t} \in \{0,1\},\ i \in M,\ t \in T$, are the same as we used in Subsection 4.3.1.

Now, the most important part of the model is to properly enforce the changes of maintenance states in time. Consider a time-unit $t \in T$ and the corresponding state s_t. Assume that at the next time unit we service machine i. Let us determine

- Variable $v_{i,t},\ i \in M,\ t \in T$, is a T-number which is formed by first $(i-1)$ digits of the T-number s_t. Expressing $v_{i,t}$ in terms of s_t we have $s_t/\mathrm{T}^{m-i+1} \leq v_{i,t} \leq s_t/\mathrm{T}^{m-i+1} + 1$.

- Variable $u_{i,t},\ i \in M,\ t \in T$, is a T-number which is first i digits of the number s_t; $s_t/\mathrm{T}^{m-i} \leq u_{i,t} \leq s_t/\mathrm{T}^{m-i} + 1$.

- Variable $w_{i,t}$, $i \in M$, $t \in T$, is a T-number which is just an i-th digit of s_t. So, $w_{i,t} = \mathrm{T}v_{i,t} - u_{i,t}$.
- Constant Φ_i is an m-digital T-number having $(m-1)$ times 1-digits and one 0-digit in i-th position. For example, 7-digital Φ_3 number is 1101111.

Now, we can describe the changing of the state in time as follows

$$s_{t+1} = s_t + \Phi_i - \mathrm{T}^{i-1}w_{i,t}, \quad t \in T \setminus \mathrm{T};$$

$$s_1 = s_{\mathrm{T}} + \Phi_i - \mathrm{T}^{i-1}w_{i,\mathrm{T}}.$$

where i is a number of a machine that we want to be serviced at the time moment $t+1$.

Taking into account that we need to decide which machine will be serviced at the next time unit, we can describe the whole set of feasible solutions of the maintenance problem as the following integer linear program:

$$\sum_{i \in M} y_{i,t} = 1, \quad t \in T; \tag{4.39}$$

$$s_{t+1} = s_t + \sum_{i \in M} \Phi_i y_{i,t+1} - \sum_{i \in M} \mathrm{T}^{i-1}w_{i,t}, \quad t \in T \setminus \mathrm{T}; \tag{4.40}$$

$$s_1 = s_{\mathrm{T}} + \sum_{i \in M} \Phi_i y_{i,1} - \sum_{i \in M} \mathrm{T}^{i-1}w_{i,\mathrm{T}}; \tag{4.41}$$

$$w_{i,t} \le \mathrm{T}y_{i,t+1}, \quad i \in M,\ t \in T \setminus \mathrm{T}; \tag{4.42}$$

$$w_{i,\mathrm{T}} \le \mathrm{T}y_{i,1}, \quad i \in M; \tag{4.43}$$

$$w_{i,t} = \mathrm{T}v_{i,t} - u_{i,t}, \quad i \in M,\ t \in T; \tag{4.44}$$

$$s_t/\mathrm{T}^{m-i+1} - \mathrm{T}^i(1-y_{i,t+1}) \le v_{i,t} \le s_t/\mathrm{T}^{m-i+1} + 1, \quad i \in M,\ t \in T \setminus \mathrm{T}; \tag{4.45}$$

$$s_{\mathrm{T}}/\mathrm{T}^{m-i+1} - \mathrm{T}^i(1-y_{i,1}) \le v_{i,\mathrm{T}} \le s_{\mathrm{T}}/\mathrm{T}^{m-i+1} + 1, \quad i \in M; \tag{4.46}$$

$$s_t/\mathrm{T}^{m-i} - \mathrm{T}^i(1-y_{i,t+1}) \le u_{i,t} \le s_t/\mathrm{T}^{m-i} + 1, \quad i \in M,\ t \in T \setminus \mathrm{T}; \tag{4.47}$$

$$s_{\mathrm{T}}/\mathrm{T}^{m-i} - \mathrm{T}^i(1-y_{i,1}) \le u_{i,\mathrm{T}} \le s_{\mathrm{T}}/\mathrm{T}^{m-i} + 1, \quad i \in M; \tag{4.48}$$

$$y_{i,t} \in \{0,1\}, \quad i \in M,\ t \in T; \tag{4.49}$$

$$s_t,\ u_{i,t},\ v_{i,t},\ w_{i,t} \quad \text{are T-numbers, } i \in M,\ t \in T. \tag{4.50}$$

Although the integer linear program (4.39)–(4.50) completely describes the feasible region of the problem it still involves the big N parameter (here it is in fact T). By consequence, the presented reformulation has also (as well as QP) a poor linear relaxation.

4.9 Tables of computational results

T	a	OPT	QP-nodes	QP-time	FF-nodes	FF-time	SP-nodes	SP-time
3	1,1,1	3.0	14	1	1	1	1	1
3	2,1,1	4.0	9	1	1	1	1	1
3	2,2,1	5.0	13	1	1	1	1	1
4	5,1,1	5.5	20	1	1	1	1	1
4	5,2,1	7.0	38	1	1	1	1	1
5	5,5,1	10.0	70	1	1	1	1	1
4	10,1,1	8.0	29	1	1	1	1	1
4	10,2,1	9.5	37	1	1	1	1	1
6	10,5,1	13.3333	156	1	3	1	9	1
16	10,10,1	17.25	197040	114	1	1	1	1
8	30,1,1	14.5	194	1	1	1	1	1
17	30,2,1	17.2941	142837	89	1	1	33	2
8	30,5,1	22.25	437	1	2	1	1	1
9	30,10,1	28.4444	1169	1	33	1	15	1
13	30,30,1	42.9231	17099	9	1	1	1	1
10	50,1,1	19.0	351	1	1	1	1	1
21	50,2,1	22.6667	766220	604	1	1	1	1
10	50,5,1	29.5	1397	2	1	1	1	1
10	50,10,1	36.5	1377	1	1	1	1	1
15	50,30,1	55.0	44664	27	17	1	27	1
17	50,50,1	66.8235	184068	114	1	1	1	1

Table 4.3: Different PMP formulations

T	a	OPT	FF-nodes	FF-time	$v(FFLP)$	SP-nodes	SP-time	$v(SPLP)$
4	1,1,1,1	6.0	1	1	6.0	1	1	6.0
9	2,1,1,1	7.3333	1	1	7.3333	1	1	7.3333
10	2,2,1,1	8.8	1	1	8.8	1	1	8.8
15	2,2,2,1	10.4	1	1	10.4	1	1	10.4
6	5,1,1,1	10.0	1	1	10.0	1	1	10.0
16	5,2,1,1	11.75	1	1	11.75	1	1	11.75
22	5,2,2,1	13.7273	99	6	13.5	1	3	13.7273
6	5,5,1,1	15.0	1	1	15.0	1	1	15.0
6	5,5,2,1	17.5	1	1	17.5	1	1	17.5
24	5,5,5,1	22.25	1	2	22.25	15[a]	3[a]	22.25[a]
6	10,1,1,1	12.5	1	1	12.5	1	1	12.5
6	10,2,1,1	15.0	1	1	15.0	1	1	15.0
6	10,2,2,1	17.5	1	1	17.5	1	1	17.5
8	10,5,1,1	19.5	1	1	19.5	1	1	19.5
6	10,5,2,1	22.5	1	1	22.5	1	1	22.5
8	10,5,5,1	27.875	1	1	27.25	19	1	27.25
8	10,10,1,1	24.5	1	1	24.5	1	1	24.5
6	10,10,2,1	27.5	1	1	27.5	1	1	27.5
9	10,10,5,1	34.0	10	1	32.8889	15	1	32.8889
33	10,10,10,1	40.4545	3	9	40.4545	17[a]	17[a]	40.4545[a]
8	30,1,1,1	21.75	1	1	21.75	1	1	21.75
8	30,5,1,1	29.5	1	1	29.5	1	1	29.5
10	30,5,5,1	40.5	3	1	39.5	17	1	39.5
8	30,10,1,1	37.0	1	1	37.0	1	1	37.0
12	30,10,5,1	49.6667	88	2	48.0	23	1	48.0
30	30,10,10,1	58.3333	123	14	57.9231	51[a]	34[a]	58.3333[a]
26	30,30,1,1	55.8462	1	3	55.8462	1	3	55.8462
24	30,30,5,1	70.5	36	5	70.4231	19[a]	4[a]	70.5
14	30,30,10,1	81.5	20	1	80.4231	23	1	80.7857
19	30,30,30,1	108.4737	1	1	108.4737	1	1	108.4737

Table 4.4: Algorithms performances and lower bounds

T	*OPT*	*FF*-nodes	*FF*-time sec.	*SP*-nodes	*SP*-time sec.
50	3.04	57	33	29^s	699^s
51	3.0	1	13	1^s	21^s
52	3.0385	111	42	7^s	154^s
53	3.0377	75	40	7^s	247^s
54	3.0	1	14	1^s	32^s
55	3.0364	94	50	13^s	1325^s
56	3.0357	156	55	11^s	590^s
57	3.0	1	19	1^s	46^s
58	3.0345	69	52	1^s	366^s
59	3.0339	61	57	3^s	407^s
60	3.0	1	22	1^s	170^s
61	3.0328	104	66	13^s	3999^s
62	3.0323	107	75	13^s	4542^s
63	3.0	1	28	1^s	195^s
64	3.0313	61	77	5^s	1431^s
65	3.0308	80	94	5^s	2067^s
66	3.0	1	31	5^s	597^s
67	3.0299	135	102	3^s	723^s
68	3.0294	162	134	7^s	1346^s
69	3.0	1	46	1^s	601^s
70	3.0286	110	149	7^s	5150^s
80	3.025	182	258	5^s	2804^s
90	3.0	1	771	1^s	1093^s
100	3.02	217	1655	3^s	5189^s

Table 4.5: Symmetry case

a	b	OPT	FF-nodes	FF-time	$v(FFLP)$
5,1,1,1,1	0,0,0,0,0	15.0	1	3	15.0
5,1,1,1,1	5,1,1,1,1	17.3333	1	3	17.3333
5,1,1,1,1	10,10,10,10,10	25.0	1	3	25.0
5,1,1,1,1	30,10,5,2,1	27.0417	86	10	26.9333
5,5,1,1,1	0,0,0,0,0	21.9583	3289	20	21.75
5,5,1,1,1	5,5,1,1,1	25.4167	15858	66	25.1429
5,5,1,1,1	10,10,10,10,10	31.9583	2917	19	31.75
5,5,1,1,1	30,10,5,2,1	33.8333	469	9	33.7143
5,5,5,1,1	0,0,0,0,0	29.5	1	2	29.5
5,5,5,1,1	5,5,5,1,1	33.5	1	2	33.5
5,5,5,1,1	10,10,10,10,10	39.5	1	2	39.5
5,5,5,1,1	30,10,5,2,1	41.125	542	7	40.8214
5,5,5,5,1	0,0,0,0,0	40.375	59750	260	39.5
5,5,5,5,1	5,5,5,5,1	44.875	82879	357	44.10
5,5,5,5,1	10,10,10,10,10	50,375	85112	354	49.5
5,5,5,5,1	30,10,5,2,1	50.375	25289	127	49.35
10,5,1,1,1	0,0,0,0,0	26.75	1	7	26.75
10,5,1,1,1	10,5,1,1,1	32.125	310	38	31.8333
10,5,1,1,1	10,10,10,10,10	36.75	1	6	36.75
10,5,1,1,1	30,10,5,2,1	41.0	5394	169	40.4167
10,10,5,1,1	0,0,0,0,0	43.5	4515	208	42.9091
10,10,5,1,1	10,10,5,1,1	50.9583	27114	829	50.2
10,10,5,1,1	10,10,10,10,10	53.5	2375	83	52.9091
10,10,5,1,1	30,10,5,2,1	56.125	3223	180	55.3833
30,10,5,1,1	0,0,0,0,0	61.4167	1443	71	60.8462
30,10,5,1,1	30,10,5,1,1	77.4167	912	61	76.5909
30,10,5,1,1	10,10,10,10,10	71.4167	1152	69	70.8462
30,10,5,1,1	30,10,5,2,1	77.5	1042	67	76.6818
30,30,1,1,1	0,0,0,0,0	69.0	177	25	68.7692
30,30,1,1,1	30,30,1,1,1	91.75	61	18	91.6364
30,30,1,1,1	10,10,10,10,10	79.0	177	25	78.7692
30,30,1,1,1	30,10,5,2,1	84.6667	528	54	83.7436
30,30,30,1,1	0,0,0,0,0	129.5	24292	122	126.9474
30,30,30,1,1	30,30,30,1,1	155.875	13947	74	153.7647
30,30,30,1,1	10,10,10,10,10	139.5	32342	164	136.9474
30,30,30,1,1	30,10,5,2,1	142.7917	24652	152	139.3860
30,30,30,30,1	0,0,0,0,0	207.75	18793	89	204.0
30,30,30,30,1	30,30,30,30,1	236.5417	23764	120	232.7826
30,30,30,30,1	10,10,10,10,10	217.75	24370	112	214.0
30,30,30,30,1	30,10,5,2,1	218.2917	15191	67	214.5326

Table 4.6: Maintenance costs

a	OPT	FF-nodes	FF-time	$v(FFLP)$	SP-nodes	SP-time	$v(SPLP)$
1,1,1,1,1, 1,1,1,1,1	49.0	$\gg$ 100000	—	45.0	1^s	1^s	49.0^s
10,9,8,7,6, 5,4,3,2,1	232.0	$\gg$ 100000	—	225.76	1671	617	232.0
10,10,10,10,10, 10,10,10,10,1	413.5	$\gg$ 100000	—	393.5	3^s	2^s	413.5^s
100,1,1,1,1, 1,1,1,1,1	126.5	1	5	126.5	1	5	126.5
1000,1,1,1,1, 1,1,1,1,1	576.5	1	3	576.5	35	19	576.5

Table 4.7: Many machines

Chapter 5

High multiplicity supply chain scheduling problems

5.1 Introduction

In classical machine scheduling problems it is usual to consider scheduling problems in which there exist jobs and machines, and in some cases resources. In contemporary manufacturing environments however, there is an important class of scheduling problems for which this context is not satisfactory. Let us start by describing in more detail, the manufacturing environment we aim to investigate.

Now, that ERP systems are widespread among manufacturers of all sorts and sizes, most companies rely for their material requirements planning on an MRP system indeed, see, e.g., [8]. For convenience of discussion, we now briefly describe the basic functionality of an MRP system. The input of an MRP system consists of two sets of data:

1. The master production schedule (MPS), in which demand for each endproduct is satisfied. This demand may consist of a combination of booked orders and forecast.

2. The Bill of Materials (BOM). The BOM specifies for each of the endproducts of which and how many intermediates or raw materials it is manufactured (assembled, configured). In turn it specifies for the intermediates of which and how many other intermediates and/or raw materials they are composed, et cetera. In addition it contains the leadtime for each of the manufacturing steps, i.e., the time necessary

to execute an endproduct or intermediate from its constituents.

Based on the MPS and the BOM an MRP engine computes which raw materials and intermediate products are required to produce the endproducts in the MPS in time.

Usually the leadtimes amount to a period length of several weeks or months. In the meantime however, the demand may change. Not in a make-to-stock environment with long delivery times, but it has become customary to produce on a make-to-order basis, or have short delivery times. In such environments, the forecast may turn out to differ from actual demand, orders are cancelled, orders get changed, and rush orders are accepted. By consequence, the material requirements as estimated by the MRP engine may differ significantly from the actual requirements. The problem that subsequently arises is how to assign the materials and/or intermediates to endproducts so as to best meet actual demand, i.e., customer orders.

Without further specifying the problem, we first point out that several high multiplicity issues may play a role in this problem. First of all, notice that the unit of measurement is not a job, but a customer order. Usually, customer orders consist of multiple items, and hence any reasonable encoding explicit schedule consisting starting (and/or completion) times of all items is not polynomial in the input size of the problem. Another multiplicity issue arises, when deliveries follow a prespecified pattern. For example, a certain raw material is delivered every Tuesday in a fixed quantity. In such a case, in any reasonable encoding even the explicit delivery schedule is not polynomial in the input size.

In this chapter, we will study problems as sketched above. The informal problem description given above is based on a quite general manufacturing setting. In this chapter we restrict our focus to the last manufacturing step, the assembly line. Nowadays, this last step is made to order in most industries, and therefore the objective can be naturally related to customer demand. As customary, we model this assembly line by a single machine, and hence the analysis in the chapter investigates a single machine scheduling problem. This approach takes quite an abstract view on the processes feeding the assembly line, they are modelled by delivery of materials. However, understanding the nature of the problem in a single machine setting may also serve as a stepping stone for considering more complicated environments, such as flow shops.

5.2 Notations, definitions, and examples

In this chapter it is convenient to adopt the three-field scheduling notation $\alpha|\beta|\gamma$ described in [40]. We shall extend this notation by notions of multiplicities and raw materials. Let us recall the basics of the notation.

5.2.1 Notations

- The β-field represents the job type data. Given a set of job types $J = \{1, 2, \ldots, s\}$ we specify the following data options related to any job type:
  - a number n_j, which is a multiplicity of the job type $j \in J$, i.e., we have to complete n_j individual jobs of type j;
  - one or more processing times p_j or $p_{k,j}$ that jobs of type j have to spend on the various machines on which it requires processing (see also the description of the machine environment in α-field below);
  - a due date $d_{j,i}$, $i \in \{1, 2, \ldots, n_j\}$, by which the i-th job of type j should ideally be completed. Such a set of due dates we call also a *demand schedule*. Notice that this data can be compactly encoded, for example, encoding by $d_{j,i} = b_j i$ requires only one number b_j. If an explicit representation of due dates or any other job attributes for all individual jobs is not polynomial in the input size of the problem we speak of *high multiplicity in demand*, otherwise we speak of *single multiplicity in demand*. Clearly, if for every job type j we have that $n_j = 1$ then we deal with single multiplicity in demand. Due dates of individual jobs can be introduced also by aggregated demand over time: let $D_j(t)$ be the total number of individual jobs of type j which should ideally be completed before or at time t;
  - presence of one of the scripts $K, ddc, mddc, rm$ means that we have a set R of raw materials. Script $rm = K$ means that we have a fixed number of raw materials ($|R| \leq K$) and different types of jobs may require the same raw material. Script ddc corresponds to the case that we have one dedicated raw material for every job type, and $mddc$ means multiple dedicated raw materials per job type. If we use script rm then we assume that there are no restrictions on the number of raw materials, or on the usage of these

materials by different types of jobs. Generally, we assume that a unit of any raw material is not renewable and can be assigned to at most one individual job. Notice, that problem input must also contain

* $a_{j,r}$ that specifies the amount of raw material r required to complete one individual job of type j,
* a *supply schedule* $s_{r,t}$ specifying an additional amount of raw material r that becomes available at time $t-1$ (so, at time interval $[t-1,t]$ we already can use the $s_{r,t}$ additional units of raw material r to process a job). Here we notice again that the supply schedule can be compactly encoded, for instance, by the linear expression $s_{r,t} = b_r t$ with only one parameter b_r for every $r \in R$. Symmetrically to high multiplicity in demand we introduce *high multiplicity in supply*: if an explicit representation of all raw material unit deliveries is not polynomial in the input size then we speak of high multiplicity in supply. A supply schedule can be represented also by aggregated supply of raw materials over time: let $S_r(t)$ be the total amount of raw material $r \in R$ delivered to time moment t;

– a weight w_j indicates a relative importance of job type j.

- The field $\alpha = \alpha_1\alpha_2$ specifies the machine environment.
  - If $\alpha_1 = \circ$, where $\circ$ denote the empty symbol, then each individual job consists a single operation that can be processed on any machine;
  - If $\alpha_1 = F$ we have a flow shop. Here, each individual job of type $j \in J$ consists a chain of operations $(O_{1,j}, O_{2,j}, \ldots, O_{m,j})$, and $O_{i,j}$ has to be processed on machine i during $p_{k,j}$ time units;
  - $\alpha_2 = m$, $m \in \mathbb{N}$, means there are m machine available to process the jobs;
- The γ-field refers to the optimality criterion chosen. Given a schedule, we can compute for each individual job j
  - the completion time of individual job $C_{j,i}$, $i \in \{1,2,\ldots,n_j\}$, or simply C_j in case $n_j = 1$;
  - the lateness $L_{j,i} = C_{j,i} - d_{j,i}$ or L_j if $n_j = 1$;

- the tardiness $T_{j,i} = \max\{0, C_{j,i} - d_{j,i}\}$ or T_j if $n_j = 1$;
- the unit penalty $U_{j,i} = 1$ if $C_{j,i} > d_{j,i}$ and 0 otherwise. We use U_j if $n_j = 1$.

The optimality criteria we use involve the minimization of the following functions (γ-field):

- the makespan $C_{max} = \max_{j,1 \le i \le n_j} C_{j,i}$;
- the maximum lateness $L_{max} = \max_{j,1 \le i \le n_j} L_{j,i}$;
- the total weighted tardiness $\sum_{j \in J} \sum_{i=1}^{n_j} w_j T_{j,i}$;
- the weighted number of late jobs $\sum_{j \in J} \sum_{i=1}^{n_j} w_j U_{j,i}$.

In general we assume that each machine can process at most one job at a time and that each job can be processed on at most one machine at a time. Also preemptions of individual jobs are not allowed.

Remark 5.2.1. *Clearly, in case $rm \neq 0$ demand is satisfiable only if there is enough raw materials to process the jobs. Thus, throughout the chapter we always assume that*

$$\sum_{t=0}^{+\infty} s_{r,t} \geq \sum_{j \in J} n_j a_{j,r}, \quad r \in R. \tag{5.1}$$

5.2.2 Examples

Let us give some examples.

Example 5.2.2. *The problem $1|rm = 1, p_j = 1|C_{max}$: minimize the makespan on a single machine subject to (1) there are no restrictions on the number of individual jobs per job type; (2) there is one raw material without any restrictions on supply schedule and consumption vector a, (3) all jobs have unit processing times.*

Let us classify this problem. The input of the problem can be represented, for example, by the following set of attributes: $\{n_1, n_2, \ldots, n_s, a_1, a_2, \ldots, a_s, (t_1, s_1), (t_2, s_2), \ldots, (t_k, s_k)\}$ where n_j is the multiplicity of job type j, a_j is a consumption of raw material by one individual job of type j, t_τ is the time moment when additional s_τ units of raw material become available. The size

of the input is $\sum_{j\in J}(\log n_j + \log a_j) + \sum_{\tau=1}^{k}(\log t_\tau + \log s_\tau)$. *An explicit representation of consumptions for all individual jobs requires space* $\sum_{j\in J} a_j$ *which is exponential in the input size, therefore we deal with high multiplicity in demand. Similarly, explicit representation of every unit of supply requires exponential space* $\sum_{\tau=1}^{k} s_\tau$, *thus we have also high multiplicity in supply.*

Example 5.2.3. $F2|rm = 0|L_{max}$: *minimize the maximal lateness in a 2-machine flow shop for the case when (1) there are no restrictions on the number of individual jobs per job type; (2) jobs do not require any raw material; (3) there are no restrictions on processing requirements; and (4) there are no constraints on due dates.*

Clearly, this is a problem with single multiplicity in supply. To answer the question whether it is high multiplicity in demand we need additional information about the encoding of the demand schedule. For example, if the due dates are given explicitly for every individual job then the problem becomes single multiplicity in demand. On the other hand, if the due dates are compactly encoded, for instance, if all individual jobs have a common due date, then the encoding of explicit due dates requires exponential space. By (4) we do not have any restrictions on due dates, therefore in the problem $F2|rm = 0|L_{max}$ *due dates can be compactly encoded which, in turn, implies high multiplicity in demand.*

Example 5.2.4. $1|ddc, s_{j,t} = 1, p_j = 1|\sum_j \sum_{i=1}^{n_j} w_j T_{j,i}$: *minimize the total weighted tardiness on a single machine under the following restrictions: (1) There are no restrictions on the number of individual jobs per job type; (2) There are s raw materials where every material is dedicated to a certain job type; (3) There are no restrictions on consumption of the raw materials; (4) At every time moment one additional unit of any raw material becomes available; (5) All individual jobs have unit processing time; (6) There are no limitations on due dates.*

Here, because of (1) and (6) we have high multiplicity in demand. Notice, that the supply schedule can be compactly represented by the function $S_j(t) = \lfloor t \rfloor$, $j \in J$ *that requires simply a constant space. Therefore, explicit encoding of supply is not polynomial in the input size of the problem. Thus, we have high multiplicity in supply.*

Remark 5.2.5. *Further, dealing with high multiplicity we will focus basically on two compact encodings of the input. We will assume that the supply schedule can be represented*

- *either by a set of pairs*

$$\bigcup_{r\in R}\{(t_\tau, s_{r,\tau})|\ s_{r,\tau} > 0,\ \forall\tau\} \tag{5.2}$$

  where t_τ is a time moment when there is a strictly positive supply of $s_{r,\tau}$ units of raw material r;

- *or by a regular supply schedule of the form*

$$S_r(t) = c_r\lfloor t/\mathrm{T}_r\rfloor + b_r,\quad b_r, c_r, \mathrm{T}_r \in \mathbb{Z}^+,\ r \in R. \tag{5.3}$$

  Here we require that in the beginning we have b_r units of the r-th raw material available, and every T_r time units the raw material $r \in R$ is supplied in amount c_r.

Symmetrically, discussing high multiplicity in demand we shall focus on two types of demand schedules:

- *The first one is a description of due dates of individual jobs of type $j \in J$ by a set of pairs*

$$\bigcup_{j\in J}\{(t_\tau, n_{j,\tau})|\ n_{j,\tau} > 0,\ \forall\tau\} \tag{5.4}$$

  where t_τ is a time moment when additional $n_{j,\tau} > 0$ individual jobs of type j should ideally be completed.

- *The second one is a regular pattern of due dates defined by a polynomially computable mapping of the form*

$$D_j(t) = c_j\lfloor (t - \mathrm{T}_j^0)/\mathrm{T}_j\rfloor^+,\quad c_j, \mathrm{T}_j, \mathrm{T}_j^0 \in \mathbb{Z}^+,\ j \in J. \tag{5.5}$$

  Here, starting from T_j^0 every T_j time units c_j individual jobs of type $j \in J$ should ideally be completed.

Let us motivate restrictions on the compact input representation by the following reason. We can easily construct a problem where given compactly

encoded $S(t)$ we can not effectively solve even the equation $S(t) = 1$ unless $P = NP$. Consider the following scheduling problem of makespan minimization. Given is one raw material, one unit time job requiring only one unit of raw material, and an instance of the well known, see [34], *NP-complete SATISFIABILITY (SAT) problem (SAT: Given a set U of variables and a collection C of clauses over U, is there a satisfying truth assignment for C?). Let $S(t)$ be such that it takes value* 1 *if the binary representation of integer t is a satisfying truth assignment for the given instance of SAT. Then any effective algorithm for the constructed scheduling problem also effectively solves SAT problem.*

Moreover, we can construct even a class of undecidable scheduling problems. Take, for instance, the general Diophantine problem, "Given a polynomial with integer coefficients in k variables, does it have an integer solution?" Matijasevic and Robinson in [83] *show that this problem is undecidable even for $k = 13$. Now, we can construct an instance where the raw material becomes available only in the roots of a given polynomial. Thus, if the Diophantine problem is undecidable, the scheduling problem is undecidable as well.*

5.3 Basic high multiplicity problems: Complexity, algorithms, and approximations

Throughout this section we analyze the complexity of very basic scheduling problems. We prove all positive results, such as algorithms and approximations, in some very general high multiplicity setting. In turn, all negative results, like NP-hardness, we prove for special single multiplicity cases. Notice that for decision (recognition) versions of single multiplicity problems considering in this section, it is rather trivial to introduce solution certificates. It implies that having NP-hardness result for single multiplicity problem we immediately have NP-completeness result for the corresponding recognition problem.

5.3.1 $1|rm = 1, p_j = 1|C_{max}$

Consider the following simple algorithm $\mathcal{A}$. First, the algorithm finds the ascending order of values a_j. Without loss of generality assume that $a_1 \leq$

$a_2 \leq \ldots \leq a_s$. Then the algorithm schedules first all jobs of type 1 then all jobs of type 2, and so on.

Proposition 5.3.1. *The solution obtained by algorithm $\mathcal{A}$ is an optimal solution for* $1|rm = 1, p_j = 1|C_{max}$.

Proof. Here we use a standard interchanging argument. Assume that there is an optimal solution where an individual job of type i is an immediate successor of an individual job of type j and $a_i \leq a_j$. If $a_i \leq a_j$ then switching the places of these two individual jobs preserves sufficiency of the raw materials for all jobs, and therefore it preserves the feasibility of solution. Since all jobs have unit processing times the switching does not change the makespan. Repetitively applying this switching argument we obtain an optimal solution in which all jobs are scheduled in order of nondecreasing a_j. □

Now, we know that the schedule obtained by algorithm $\mathcal{A}$ is an optimal one. The question that remains is how to output this optimal schedule. If it is allowed to output the schedule as a sequence of jobs to be processed on the single machine then we are done and we can even compactly encode the resulting solution by a polynomial size mapping $\pi(j,i) = \sum_{j'<j} n_{j'} + i,\ j \in J,\ 1 \leq i \leq n_j$. We call such a mapping a sequence-oriented description of the output, see Chapter 2. Clearly, the value $\pi(j,i)$, denoting the ordering number of the i-th individual job of type j in the optimal job sequence, can be computed in time that is polynomial in the input size of the problem.

Now, assume that it is required to output the completion (starting) times of the individual jobs. The explicit representation of all completion times will be exponential in the input size of the problem. As we discussed in Chapter 2, see also [13], instead of explicit representation we can output, for example, a time-oriented or job-oriented description of the optimal schedule.

First, consider the case with a supply schedule representation given by equation (5.2), see Remark 5.2.5. Without loss of generality, we assume that $t_1 \leq t_2 \leq \ldots \leq t_k$ and $a_1 \leq a_2 \leq \ldots \leq a_s$. Let us schedule as many jobs of type 1 as possible in the time interval $[t_1, t_2]$. To do this compare the numbers $n_1 a_1$ and s_1. If $n_1 a_1 \leq s_1$ then let $n'_{1,1} = n_1$, otherwise $n'_{1,1} = \lfloor s_1/a_1 \rfloor$, and assign $n'_{1,1}$ individual jobs of type 1 to the interval $[t_1, t_2]$. Redefine $s_1 := s_1 - n'_1 a_1$ and $n_1 := n_1 - n'_{1,1}$, and continue with job type 2, 3, and so on. If the amount of raw material s_1 is not enough to schedule any remaining job in the interval $[t_1, t_2]$ let $s_2 := s_2 + s_1$ and consider the next

interval, $[t_2, t_3]$. Continue the procedure till the last interval, $[t_k, t_{k+1})$ (here $t_{k+1} = +\infty$). As a result we obtain a matrix $[n'_{j,q}]_{s\times k}$ where $n'_{j,q}$ is the number of individual jobs of type $j \in J$ assigned to the interval $[t_q, t_{q+1}]$, $1 \le q \le k$. Notice that matrix $[n'_{j,q}]_{s\times k}$ has a very special structure. Compact encoding of this matrix requires only $O(\max\{k, s\})$ space.

Now, let us define a time-oriented description where given a time moment t we output a triple (i, j, τ) such that in the schedule the i-th individual job of type j is the first individual job to be processed at or after time t and τ is starting time of that job. Without loss of generality, assume that $t_q \le t \le t_{q+1}$ for some $1 \le q \le k$. Define the time-oriented description as follows

$$G(t) = \begin{cases} (j,i,t), & \text{if } t \le t_q + \sum_{j' \le j} n'_{j',q}, \\ & \text{and } t \ge t_q + \sum_{j' < j} n'_{j',q}, \\ & \text{and } i = t - t_q - \sum_{j' < j} n'_{j',q} + \sum_{q' < q} n'_{j,q'} + 1; \\ (j,i,t_{q^*}), & \text{if } t > t_q + \sum_{j' \in J} n'_{j',q}, \\ & \text{and } q^* = \min\{q' > q \mid \exists j \in J : n'_{j,q^*} > 0\}, \\ & \text{and } j = \min\{j \in J \mid n'_{j,q^*} > 0\}, \\ & \text{and } i = \sum_{q' < q} n'_{j,q'} + 1; \\ (0,0,0), & \text{otherwise.} \end{cases}$$

Notice that the time-oriented description above requires space $O(\max\{k, s\})$. This description can be obtained in time $O(\max\{k, s\})$, and, given $t \in \mathbb{R}$, we can compute the triple corresponding to t also in time $O(\max\{k, s\})$.

In the case of a supply schedule representation given by equation (5.3), see Remark 5.2.5, we notice that the optimal makespan C_{max} is easy to compute. First, calculate the earliest time moment at which there is enough raw material to process all jobs, $t^* = \lceil T(\sum_{j\in J} n_j a_j - b)/c \rceil$. Then calculate the number of individual jobs of type $j \in J$ that are scheduled before time t^* as follows:

$$n'_j = \begin{cases} n_j, & \text{if } \sum_{j' \le j} n_{j'} a_{j'} \le c\lfloor t^*/T \rfloor - c + b; \\ 0, & \text{if } \sum_{j' < j} n_{j'} a_{j'} \ge c\lfloor t^*/T \rfloor - c + b; \\ \left\lfloor \frac{c\lfloor t^*/T \rfloor - c + b - \sum_{j' < j} n_{j'} a_{j'}}{a_j} \right\rfloor, & \text{otherwise.} \end{cases}$$

Then $C_{max} = t^* + \sum_{j\in J}(n_j - n'_j)$. Finally we output the following job-oriented description of an optimal schedule where all idle time is gathered in

the beginning of the schedule:

$$C_{j,i} = C_{max} - \sum_{j'>j} n_{j'} - (n_j - i).$$

Here, computation of the completion time of any individual job takes only $O(s)$ time. The required space for the job-oriented description is also at most $O(s)$.

5.3.2 $1|n_j = 1, rm = 1|C_{max}$

In this section we show that the extension of the problem $1|rm = 1, p_j = 1|C_{max}$ to the case with arbitrary processing times is difficult to solve even in case of single multiplicities in demand.

Theorem 5.3.2. *The problem $1|n_j = 1, rm = 1|C_{max}$ is strongly NP-hard and the corresponding recognition version of the problem is strongly NP-complete.*

Proof. Let us reduce the well known, see, e.g., [34], strongly NP-complete problem 3-PARTITION to $1|n_j = 1, rm = 1|C_{max}$.

3-PARTITION: Given a set E of $3m$ elements, a bound $B \in \mathbb{Z}^+$, and size $s(e) \in \mathbb{Z}^+$ for each $e \in E$ such that $B/4 < s(e) < B/2$ and $\sum_{e \in E} s(e) = mB$, can E be partitioned into m disjoint sets $E_1, E_2, \ldots, E_m$ such that for $1 \leq i \leq m$, $\sum_{e \in E_i} s(e) = B$?

Define
$J = E$;
$a_e = s(e),\ e \in E$;
$p_e = s(e),\ e \in E$;
$s_t = B$ if $t \equiv 1(\mod B)$ and $s_t = 0$ otherwise.

We claim there is a required partition of set E if and only if there is a schedule for the constructed instance of $1|n_j = 1, rm = 1|C_{max}$ with makespan $C_{max} = mB$.

Assume that in the constructed instance of $1|n_j = 1, rm = 1|C_{max}$ there is a schedule of length mB. Since the total workload of all jobs from E is mB and the schedule length is also mB we conclude that in the schedule there

are no idle times. Consider the interval $[1, B]$. Since $B/4 < s(e) < B/2$ for any $e \in E$ and in the schedule there are no idle times, exactly three jobs $E_1 = \{e_{1,1}, e_{1,2}, e_{1,3}\}$ with total workload $\sum_{e \in E_1} p_e = \sum_{e \in E_1} s(e) \geq B$ start in the interval $[1, B]$. By definition of raw material consumptions and the raw material constraint, we deduce that $\sum_{e \in E_1} a_e = \sum_{e \in E_1} s(e) \leq B$. Thus, $\sum_{e \in E_1} s(e) = B$. Applying the same arguments to intervals $[B + 1, 2B], [2B + 1, 3B], \ldots, [(m-1)B + 1, mB]$ we find that $E_1, E_2, \ldots, E_m$ is required partition.

If there is a partition then it is trivial to construct a feasible schedule of the length $C_{max} = mB$: just schedule first all the jobs from E_1, then schedule all the jobs from E_2, and so on. Since at a time $t \equiv 1 \pmod{B}$ there is enough raw material to schedule any triple E_i, $1 \leq i \leq m$ and the total workload on the triple E_i is B, we get a feasible schedule without idle times. It implies the makespan $C_{max} = mB$.

The recognition version of $1|n_j = 1, rm = 1|C_{max}$ is clearly in NP, and therefore it is strongly NP-complete. □

5.3.3 $1|rm = 1, s_t = 1|C_{max}$

Consider the high multiplicity in demand problem with regular unit supply of raw material. Here, at each time moment one additional unit of raw material becomes available, i.e., $s_t = 1$ or equivalently $S(t) = \lfloor t \rfloor$.

Let us reduce the problem to the flow shop problem $F2|rm = 0|C_{max}$. We construct the instance of the flow shop as follows. The set of jobs stays the same. Take an arbitrary job type $j \in J$. Let the first operation $O_{1,j}$ be the collection of the required raw materials. This operation take time $p_{1,j} = a_j$. Let the second operation $O_{2,j}$ be the processing of the job on the machine. It takes time p_j. Clearly, the constructed flow shop correctly models $1|rm = 1, s_t = 1|C_{max}$.

Proposition 5.3.3. *The problem $1|rm = 1, s_t = 1|C_{max}$ can be solved in time $O(s \log s)$.*

Proof. The proof straightforwardly follows from the reduction above and the fact that Johnson's algorithm, see [63], provides an optimal solution for $F2|n_j = 1, rm = 0|C_{max}$ in time $O(s \log s)$.

To prove the theorem it remains to manage high multiplicities in demand. Let us remind Johnson's algorithm. First schedule the jobs with $p_{1,j} \leq p_{2,j}$

in order of nondecreasing $p_{1,j}$, and then schedule the remaining jobs in order of nonincreasing $p_{2,j}$. Note that this algorithm creates an order on types but not on the individual jobs. Therefore, we can compute it in polynomial time in the input size. It implies that there is a polynomial time algorithm that output a polynomial time and polynomial size sequence-oriented description of the optimal schedule.

Now, let us construct a polynomial size job-oriented description of the schedule such that given $j \in J$ and $1 \leq i \leq n_j$ it outputs the completion time $C_{j,i}$ in polynomial time. Without loss of generality, assume that the optimal order of types by the Johnson's algorithm is $1 \prec 2 \prec \ldots \prec s$. Consider the following procedure.

Initialization. Let $l := 1$, $\alpha_l = 0$ and $\beta_l = 0$.

Basic step.
If $l = j$ then define and output (and STOP)

$$C_{j,i} = \max\{\alpha_j + ia_j + p_j, \beta_j + ip_j\}, \tag{5.6}$$

otherwise let
$l := l + 1$,
$\alpha_l := \alpha_{l-1} + a_{l-1}n_{l-1}$,
$\beta_l := \max\{\alpha_{l-1} + n_{l-1}a_{l-1} + p_{l-1}, \beta_{l-1} + n_{l-1}p_{l-1}\}$,
and repeat the basic step.

To verify correctness of the procedure, consider the l-th basic step of it. Assume that we collect raw materials for the jobs of types $1, 2, \ldots, l-1$ till time α_l and process these jobs till time β_l. So, starting from α_l we can collect the raw materials for the jobs of type l (the first operation in the corresponding flow shop) and starting from β_l we can process the jobs of type l if there is enough the raw materials (the second operation in the flow shop can start only after the first operation is complete). In the initial step, we have $\alpha_l = 0$ and $\beta_l = 0$. Notice that the processing of the i-th individual job of type l cannot start earlier than time $\alpha_l + ia_j$ since we do not have enough raw materials to process i jobs of type l. Also it can not start earlier than time $\beta_l + (i-1)p_l$ since we have to complete $i-1$ jobs of that type before we start the i-th individual job. Therefore, the maximum of these two values plus p_j is a lower bound on the completion time of the i-th individual job of type l. Now, let us show that this bound is reachable. Clearly, for any step l we have $\alpha_l \leq \beta_l$. If $a_l \leq p_l$ then for

any $i \in \mathbb{Z}^+$ it holds that $\alpha_l + ia_j \leq \beta_l + (i-1)p_l$ and we can schedule all jobs of type l without idle times which implies $C_{l,i} = \beta_l + ip_l$. If $a_l > p_l$ then idle times are possible. In this case idle time can appear only because of lack of raw materials. This means that immediately after we collect the required raw materials we can start the processing of the job. It implies $C_{l,i} = \alpha_l + ia_l + p_l$. These observations explain equation (5.6) and calculation of C_{l,n_l}. It remains to define the values α_{l+1} and β_{l+1} for the next basic step of the procedure. $\alpha_{l+1} = \alpha_l + a_l n_l$ is because we can collect raw materials without any idle times, and $\beta_{l+1} = \max\{\alpha_l + n_l a_l + p_l, \beta_l + n_l p_l\}$ is because it is the completion time of the n_l-th job of type l. So, in $O(s)$ time the procedure correctly output the completion time of any individual job in the optimal schedule computed by Johnson's algorithm. Here, the optimal objective value is simply $C_{\max} = C_{s,n_s}$.

Notice that the most expensive operation of the algorithm is to compute an order by the Johnson's rule. Therefore the running time of the algorithm is $O(s \log s)$, which completes the proof. □

5.3.4 $1|n_j = 1, rm = 2, p_j = 1|C_{max}$

In this section we show that the extension of the problem $1|rm = 1, p_j = 1|C_{max}$ to the case with two raw materials is difficult to solve even in case of single multiplicities in demand.

Theorem 5.3.4. *The problem $1|n_j = 1, rm = 2, p_j = 1|C_{max}$ is strongly NP-hard, and the recognition version of $1|n_j = 1, rm = 2, p_j = 1|C_{max}$ is strongly NP-complete.*

Proof. Due to Holthuijsen and Van de Klundert [60]: Let us reduce again 3-PARTITION to $1|n_j = 1, rm = 2, p_j = 1|C_{max}$.

We construct the reduction as follows. Define
$J = E$;
$a_{e,1} = s(e),\ e \in E$;
$a_{e,2} = B - s(e),\ e \in E$;
$s_{1,t} = B$ if $t \equiv 1(\mod 3)$ and $s_{1,t} = 0$ otherwise;
$s_{2,t} = 2B$ if $t \equiv 1(\mod 3)$ and $s_{2,t} = 0$ otherwise.

We claim that E can be partitioned as required if and only if there is a schedule for the constructed instance of $1|n_j = 1, rm = 2, p_j = 1|C_{max}$ with makespan $C_{max} = 3m$.

First, assume that in the constructed instance of $1|n_j = 1, rm = 2, p_j = 1|C_{max}$ there is a schedule of length $3m$. Since we have $3m$ unit time jobs and the makespan is also $3m$ we know that at each time unit we process one job. Let E_i consist three jobs $e_{i,1}$, $e_{i,2}$ and $e_{i,3}$ which are scheduled at time units $3i-2$, $3i-1$ and $3i$ where $1 \leq i \leq m$. Consider the first triple E_1. By definition of a and by constraint on the first raw material we have $\sum_{e \in E_1} s(e) = \sum_{e \in E_1} a_{e,1} \leq B$. By restriction on the second raw material $3B - \sum_{e \in E_1} s(e) = \sum_{e \in E_1} (B - s(e)) = \sum_{e \in E_1} a_{e,2} \leq 2B$. Therefore, $\sum_{e \in E_1} s(e) \geq B$, and immediately $\sum_{e \in E_1} s(e) = B$. Continuing further for E_2, E_3, and so on, we observe that $E_1, E_2, \ldots, E_m$ is the required partition.

Now, assume that partition exists. Clearly, scheduling triples $E_1, E_2, \ldots, E_m$ one-by-one without idle times we get a feasible schedule with makespan $3m$. It remains to notice that the reduction is polynomial in the input size of 3-PARTITION.

NP-completeness of the recognition problem follows straightforwardly from the hardness proof above and the fact that the recognition problem is in NP. □

5.3.5 $1|ddc|C_{max}$

Consider a trivial scheduling problem $1|n_j = 1, ddc|C_{max}$. Here, dedicated raw materials play the role of ordinary job release dates (a release date is the earliest time moment at which processing of the job can be started). For the single multiplicity problem $1|n_j = 1, ddc|C_{max}$ the following trivial algorithm provides an optimal solution: schedule the jobs in order of nondecreasing release dates. However, in the high multiplicity case, the explicit representation of an order of individual jobs is exponential in the input size of the problem. Therefore, we have to encode this order more compactly.

As mentioned in Remark 5.2.5 we consider two descriptions of supply schedules. One is a representation by equation (5.2). Since the raw materials are dedicated, we may say that the supply schedule is defined by set of pairs $\bigcup_{j \in J} \{(\tau, s_{j,\tau}) |\ s_{j,t} > 0\}$. Let T be the set of time moments when there is a strictly positive supply of the raw material for at least one of the job types and let $k = |T|$. In time $O(k \log k)$ (which is polynomial in the input size)

one can arrange the elements of T in non-decreasing order. Without loss of generality, assume that $T = \{t_1, t_2, \ldots, t_k\}$ and $t_1 \leq t_2 \leq \ldots \leq t_k$. Consider $n'_{j,t}$ which is the number of j-th type jobs released before or at time $t \in T$. We can compute this number by $n'_{j,t} = \lfloor \sum_{\tau \leq t} s_{j,\tau}/a_j \rfloor$ in linear time in the input size. If for every $j \in J$ we compute the numbers $n'_{j,t}$, $t \in T$, then we can output the sequence-oriented description of the schedule:

Input. $j \in J$ and $1 \leq i \leq n_j$.

Output. $\pi(j,i)$ which is an ordering number of i-th individual job of type j in the optimal schedule.

Step 1. Assuming $t_0 = 0$ find $\tau \in \{1, 2, \ldots, k\}$ such that $n'_{j,t_{\tau-1}} < i \leq n'_{j,t_\tau}$.

Step 2. Define and output (and STOP)

$$\pi(j,i) = i + \sum_{j'<j} n_{j',t_\tau}. \tag{5.7}$$

It is not difficult to see that this encodes the individual jobs in order of non-decreasing release dates and therefore this is a compact sequence-oriented description of the optimal schedule. A similar technique with the assignment of the job batches to a polynomial number of time intervals was used also by Hochbaum and Shamir in [58] to prove the existence of a strongly polynomial algorithms for problems $1|p_j = 1, d_{j,i} = d_j| \sum_{i=1}^{n_j} w_j T_{j,i}$ and $1|p_j = 1, d_{j,i} = d_j| \sum_{i=1}^{n_j} w_j T_{j,i}$.

Now, let us construct a compact (polynomial in the input size) job-oriented description of the optimal solution. Given a job type $j \in J$ and the replication number $1 \leq i \leq n_j$, we calculate the completion time $C_{j,i}$ by the following procedure.

Initialization. Let $l = 1$ and $t^* = t_1$.

Basic step. If $\sum_{k'<l} n'_{j,t_{k'}} < i \leq \sum_{k' \leq l} n'_{j,t_{k'}}$ then define and output (and STOP)

$$C_{j,i} = t^* + \sum_{j'<j} p_{j'} n_{j',t_l} + p_j (i - \sum_{k'<l} n_{j,t_{k'}}), \tag{5.8}$$

otherwise let $l := l + 1$, $t^* := \max\{t_l, t^* + \sum_{j' \in J} p_{j'} n_{j',t_{l-1}}\}$ and repeat the basic step.

Proposition 5.3.5. *Given $j \in J$ and $1 \leq i \leq n_j$, the procedure above in polynomial time correctly outputs the optimal completion time $C_{j,i}$ (in particular, $C_{max} = \max_{j \in J} C_{j,n_j}$ is obtainable in polynomial time).*

Proof. Let us analyze this procedure. Call a batch of the jobs not released by time t_{l-1} and released by time t_l the l-batch. At the l-th basic step of the procedure either the i-th individual job of type j belongs to the l-batch or it does not. Assume that $t^* \geq t_l$ is a time when the l-batch starts to process. If the i-th individual job of type j belongs to the batch, we obtain the required completion time by summing up t^* and the processing times of the predecessors of that job. If the individual job i of type j does not belong to the batch, we go to the $(l+1)$-batch and the earliest starting time for the new batch is exactly $t^* := \max\{t_{l+1}, t^* + \sum_{j' \in J} p_{j'} n_{j',l}\}$. This implies the correctness of the output.

The procedure requires space $O(sk \log n_{\max})$, where $n_{\max} = \max_{j \in J} n_j$, which is polynomial in the input size of the problem. The running time of the procedure, given $j \in J$ and $1 \leq i \leq n_j$, is also $O(sk \log n_{\max})$. □

Consider the second case, when the supply schedules are represented by mappings (5.3): $S_j(t) = c_j \lfloor t/\mathrm{T_j} \rfloor + b_j$, $b_j, c_j, \mathrm{T_j} \in \mathbb{Z}^+$, $j \in J$. For this case we have the following theorem.

Theorem 5.3.6. *There exists a polynomial time algorithm that outputs a polynomial time and polynomial size sequence-oriented description of the optimal schedule for $1|ddc|C_{max}$.*

Proof. Consider an optimal job processing sequence corresponding to the non-decreasing order of job release dates, call it *Earliest Release Date* order or simply ERD order. To be more specific, let this sequence be such that

- an individual job i of type j precedes individual job i' of the same type if $i < i'$;
- an individual job i of type j having release date t precedes an individual job i' of type j' having the same release date if $j < j'$.

Consider an arbitrary job type $j \in J$. Let us denote the i-th individual job of type j by (j, i). Let us count how many individual jobs of type $k \in J$ are preceding the job (j, i) in the optimal solution specified above. By definition, exactly $i - 1$ jobs of type j precede job (j, i).

The release date of the job (j,i) is the minimal number t satisfying the inequality $c_j\lfloor t/\mathrm{T}_j\rfloor + b_j \geq ia_j$. It implies that $t = \lceil \mathrm{T}_j(a_j i - b_j)/c_j\rceil$. Notice that we can effectively compute this release date. By time t, the aggregated supply of the k-th raw material equals $c_k\lfloor t/\mathrm{T}_k\rfloor + b_k$ and in the specified solution we have

$$n'_k = \left\lfloor \frac{c_k\lfloor t/\mathrm{T}_k\rfloor + b_k}{a_k} \right\rfloor$$

released jobs of type k.

Now, if the release date of job (k,n'_k) is not equal to t then all n'_k jobs of type k must precede (j,i)-job. If the release date of job (k,n'_k) and that of job (j,i) are equal to t then in case $k<j$ we have n'_k preceding jobs, and in case $j>k$ we have $n'_k - n''_k$ preceding jobs where n''_k is the number of jobs of k type released exactly at moment t:

$$n''_k = \left\lfloor \frac{c_k\lfloor t/\mathrm{T}_k\rfloor + b_k}{a_k} \right\rfloor - \left\lfloor \frac{c_k\lfloor t/\mathrm{T}_k\rfloor - c_k + b_k}{a_k} \right\rfloor.$$

Summing up the numbers of preceding jobs over all types and adding 1 gives the ordering number of job (j,i) in the optimal sequence.

The described procedure to determine the number of predecessors is polynomial in the input size as required. □

Consider the reverse problem: what is the type of k-th individual job in the order? To answer this question we prove another theorem.

Theorem 5.3.7. *There exists a polynomial time procedure computing the type of the k-th individual job in ERD order (optimal job processing sequence for problem $1|ddc|C_{max}$).*

Proof. Let $\pi(j,i)$ be the output of the procedure described in Theorem 5.3.6 for an input $j \in J$ and $1 \leq i \leq n_j$. Consider arbitrary $j \in J$. Doing bisection search over $1 \leq i \leq n_j$ we can find a number i such that $\pi(j,i) < k < \pi(j,i+1)$ or $\pi(j,i) = k$. Notice that there always exists a unique pair (j,i) for which $\pi(j,i) = k$. The procedure outputs $j \in J$ for which $\pi(j,i) = k$. It remains to notice that the running time of the procedure is polynomial in the input size. It is true since for every $j \in J$ bisection search takes at most $O(\log n_j)$ iterations of computing $\pi(j,i)$. Computing $\pi(j,i)$ takes $O(s)$ time which implies that the running time of the whole procedure is $O(s^2 \log n_{\max})$, which is indeed polynomial in the input length. □

Remark 5.3.8. *Notice that for the case with a regular supply schedule we construct only a sequence-oriented description of the optimal solution but not a job- or a time-oriented description as in* [13] *or in Chapter 2. Moreover, it is not so obvious that, given a compact representation of an optimal sequence, we can compute the completion times of individual jobs and, in particular,* C_{max}.

Remark 5.3.9. *In Theorems 5.3.6 and 5.3.7 we have introduced a technique for a polynomial space and polynomially computable description of the optimal job sequence in* $1|ddc|C_{max}$. *In this remark, we would like to substantially generalize this technique.*

Consider a high multiplicity scheduling problem with the set of job types J *and job multiplicities* n_j, $j \in J$. *Suppose that a feasible solution of the problem can be represented as just a sequence of individual jobs (high multiplicity sequencing problem). Then to compactly encode a solution it is sufficient to present a polynomial space and polynomial time procedure such that*

- *Given* (j_1, i_1)*-job and* (j_2, i_2)*-job, the procedure outputs either* $\pi(j_1, i_1) < \pi(j_2, i_2)$ *or* $\pi(j_1, i_1) > \pi(j_2, i_2)$;
- *The procedure guarantees that the transitivity property of a linear order is satisfied, i.e.,* $\pi(j_1, i_1) < \pi(j_2, i_2)$ *and* $\pi(j_2, i_2) < \pi(j_3, i_3)$ *imply* $\pi(j_1, i_1) < \pi(j_3, i_3)$ *for any individual jobs* (j_1, i_1), (j_2, i_2), *and* (j_3, i_3).

Indeed, if the two mentioned conditions are satisfied, the procedure encodes a unique linear order (a unique sequence).

An advantage of this description is that we do not require to explicitly compute the ordering numbers of the jobs. We require only that the procedure has to be able to specify the order between any two individual jobs. A disadvantage of this encoding is that in general we do not know how to translate in polynomial time the sequence description into the job- or time-oriented description. As a consequence, given such a sequence description, we do not know how to compute the objective value of the solution.

5.3.6 An approximation algorithm for $1|rm|C_{max}$

In this section we introduce an approximation algorithm for $1|rm|C_{max}$ with worst case performance ratio of 2. Notice, that this is a high multiplicity

(in both, demand and supply) problem without any restriction on number of raw materials, supply schedule and consumption of the raw materials.

Consider the following very simple algorithm $\mathcal{A}_1$. Compute the time moment

$$t^0 = \min \{ t \mid \sum_{\tau \le t} s_{r,\tau} \ge \sum_{j \in J} n_j a_{j,r} \text{ for any } r \in R \} \qquad (5.9)$$

when we receive the last required supply of raw materials to process all jobs. Starting from time moment t^0 algorithm $\mathcal{A}_1$ schedules the jobs simply type-by-type: first all individual jobs of one type, then another, and so on. For definitiveness one can schedule the jobs in order of increasing a_j/p_j (actually one can choose any order of types or even any schedule without idle times). Let $C_{max}(\mathcal{A}_1)$ be the makespan of the solution obtained by this algorithm and C^*_{max} be the optimal makespan. Then we have the following theorem.

Theorem 5.3.10. *$C_{max}(\mathcal{A})/C^*_{max} \le 2$ and the ratio of 2 is tight.*

Proof. Clearly, $C^*_{max} \ge t^0 + 1$ and $C^*_{max} \ge \sum_{j \in J} n_j p_j$. Since after time moment t^0 the algorithm $\mathcal{A}_1$ schedules all jobs without idle times we have $C_{max}(\mathcal{A}_1) = t^0 + \sum_{j \in J} n_j p_j \le 2C^*_{max}$.

It remains to show that the ratio of 2 is tight. Consider the case where we have only two individual jobs. Let the first job have the processing time p and it does not require any raw material. The second job has unit processing time and requires one unit of raw material, which becomes available at time moment $p+1$. For this instance $C^*_{max} = p+1$ and $C_{max}(\mathcal{A}_1) = 2p+1$ which yields $\lim_{p \to \infty} C_{max}(\mathcal{A})/C^*_{max} = \lim_{p \to \infty}(2p+1)/(p+1) = 2$. □

5.3.7 $1|rm = 0|L_{max}$

For problem $1|n_j = 1, rm = 0|L_{max}$, Jackson [62] shows that an optimal solution is obtained in $O(s \log s)$ time by sequencing jobs in non-decreasing order of their due dates. Traditionally, this method is called *Earliest Due Date* or simply EDD rule, and the corresponding order the EDD order.

In Section 5.3.5, we have observed how high multiplicity can be handled in the case that an optimal solution is obtained by sequencing jobs in non-decreasing order of their release dates. Thus, straightforwardly applying the techniques from Section 5.3.5 we derive the following results. Take any instance of $1|rm = 0|L_{max}$ satisfying the conditions described in Remark 5.2.5. We claim

Theorem 5.3.11. *There exists a polynomial time algorithm that outputs a polynomial time and polynomial size sequence-oriented description of the optimal schedule for* $1|rm = 0|L_{max}$.

Theorem 5.3.12. *There exists a polynomial time procedure computing the type of k-th individual job in EDD order (optimal job processing sequence for* $1|rm = 0|L_{max}$*).*

In contrast to Section 5.3.5, in this case we can even construct a polynomial size and polynomially computable job-oriented description of an optimal schedule. However, it is still not clear how to compute the maximum lateness L_{max} in polynomial time in the case that the due dates are regular and given by a very compact mapping (5.5), see Remark 5.2.5. Even if a job-oriented description of an optimal solution is already given, it is not clear how to compute the maximum lateness.

5.3.8 $1|rm = 1, p_j = 1|L_{max}$

In this section we assume that the supply schedule and the job due dates are described by representations (5.2) and (5.4) respectively. We introduce a polynomial time and polynomial space algorithm $\mathcal{A}$ answering the question whether there exists a schedule for $1|rm = 1, p_j = 1|L_{max}$ on this class of instances such that $L_{max} = 0$. Moreover, if such a schedule exists the algorithm outputs a job-oriented description of the schedule. The algorithm is an extension of the algorithm by [60] solving in polynomial time the corresponding version of single multiplicity problem $1|n_j = 1, rm = 1, p_j = 1|L_{max}$.

The basic idea of the algorithm is following. Notice that we have a polynomial number of time moments where we have strictly positive additional supply or strictly positive additional demand (the due dates of individual jobs). Consider any two sequential time moments from this set. Clearly, in the interval between such moments neither supply, nor demand changes. Consider the last such interval. Schedule the demand (individual jobs) of this interval backwards (from the latest time moment to the earliest one) in non-increasing order of raw material consumptions. Assign the remaining part of the demand to the demand of the previous interval and proceed recursively. Check sufficiency of the raw material supply and sufficiency of the schedule length for the remaining jobs (jobs which are not scheduled yet) on every

step of the procedure. In Proposition 5.3.13 we prove that this procedure correctly solves the problem.

More formally we define the algorithm $\mathcal{A}$:

Input:

1. A set of pairs $\{(\tau_i, s(\tau_i))|\ i \in I\}$, where τ_i is i-th time moment when additional strictly positive amount $s(\tau_i)$ of raw material becomes available.

2. A set of pairs $\bigcup_{j\in J}\{(d_k, n_j(d_k))|\ k \in K\}$, where d_k is k-th time moment when $n_j(d_k)$ individual jobs have their due dates and at least one of this numbers is strictly positive. Notice, $\sum_{k\in K} n_j(d_k) = n_j$.

3. A set of consumption requirements $\{a_j|\ j \in J\}$. Without loss of generality we may assume that $a_1 \geq a_2 \geq \ldots \geq a_s$.

Output:

Given an instance of $1|rm = 1, p_j = 1|L_{max}$, if there is no schedule such that $L_{max} = 0$ then the algorithm stops with answer "NO". If there is a schedule with zero maximum lateness then the algorithm outputs "YES" and a certificate such that, given a job type $j \in J$ and a number $1 \leq i \leq n_j$, it in polynomial time computes the starting/completion time of (j,i)-job in the schedule.

Preliminary steps:

1. Create a set of triples $\{(t_l, S_l, n_{j,l} :\ l \in L = I \cup K,\ j \in J\}$, where t_l, $l \in L$, is a time moment, S_l is aggregated supply to this time moment, and $n_{j,l}$ individual jobs of type $j \in J$ have their due date.

2. Find the lowest number j_0 from J (the job type with highest consumption of raw material) such that $n_{j_0,|L|} > 0$. Define $n'_{j,|L|+1} = 1$ if $j = j_0$ and $n'_{j,|L|+1} = 0$ otherwise. Redefine $n_{j,|L|} := n_{j,|L|} - n'_{j,|L|+1}$, $j \in J$.

3. Let $l := |L|$.

Basic steps:

1. Check whether $\sum_{j\in J}\sum_{l'\le l} n_{j,l'} \le t_l$. If not then STOP: the length of the interval $[0, t_l]$ is too short to process the jobs in time.

2. Check whether $\sum_{j\in J}\sum_{l'\le l} a_j n_{j,l'} \le S_l$. If not then STOP: the aggregated supply to moment t_l is not sufficient to process the jobs in time.

3. If $l = 1$ then STOP: there is a schedule with zero maximum lateness.

4. Consider an interval $[t_{l-1}, t_l]$. Find $\min\{t_{l-1} - t_l, \sum_{j\in J} n_{j,l}\}$ individual jobs with the highest raw material consumption assuming not allowing to take more than $n_{j,l}$ jobs of type $j \in J$. Let $n'_{j,l}$ be the number of such jobs of type j. Redefine $n_{j,l-1} := n_{j,l-1} + n_{j,l} - n'_{j,l}$, $j \in J$, let $l := l - 1$, and repeat the basic steps.

Job-oriented schedule:

Input: $j \in J$, and $1 \le i \le n_j$.

Output: $C_{j,i}$.

Computation of $C_{j,i}$:

1. Find a number $l \in L$ such that

$$\sum_{l'<l} n'_{j,l'} < i \le \sum_{l'\le l} n'_{j,l'}. \tag{5.10}$$

2. Output the number

$$C_{j,i} = t_l - \sum_{j'>j} n'_{j',l} + n_j - \sum_{l'>l} n'_{j,l'} - i. \tag{5.11}$$

Let us clarify equation (5.11). According to algorithm $\mathcal{A}$, within interval $[t_{l-1}, t_l]$ we schedule $n'_{j',l}$ individual jobs of type j'. Moreover, we schedule individual jobs backwards starting from time moment t_l in order of nonincreasing of j' (by assumption, $a_1 \le a_2 \le \ldots \le a_s$). This means that in the interval $[t_{l-1}, t_l]$ job (j, i) has $n'_{j',l}$ jobs-successors of type j' if $j' > j$, and 0 successors of type j' otherwise. Now, let us count how many successors of type j has job (j, i) on interval $[t_{l-1}, t_l]$. By time t_l we have to schedule

$n_j - \sum_{l'>l} n'_{j,l'}$ jobs of type j. This implies that after the i-th job of type j we have to schedule $n_j - \sum_{l'>l} n'_{j,l'} - i$ jobs of type j in the interval $[t_{l-1}, t_l]$. Summing up the unit processing times of all successors of job (j,i) in the interval $[t_{l-1}, t_l]$ we derive equation (5.11).

Proposition 5.3.13. *Given an instance of* $1|rm = 1, p_j = 1|L_{max}$, *the algorithm* $\mathcal{A}$ *in* $O(s|L|^2)$ *correctly answers the question whether there exists a schedule with zero maximum lateness.*

Proof. First let us show that the algorithm correctly answers the question. To prove it we use the following inductive argument. Consider the l-th basic step of the algorithm. In this stage we are solving the reduced problem $P(l)$ with only $\sum_{l' \leq l} n_{j,l'}$ individual jobs of type $j \in J$ where $n_{j,l'}$ jobs of type j have the due dates $t_{l'}$, $1 \leq l' \leq l$. Notice that quantities $n_{j,l'}$, $j \in J$, $l' \leq l$, are redefined by the algorithm. Now, the inductive argument is following. For $P(l)$ there exists a schedule with zero maximum lateness if and only if (1) there are enough raw materials to finish all individual jobs in time; (2) the length of the planning interval $[0, t_l]$ is long enough to complete all jobs in time; (3) $P(l-1)$ has a schedule with zero maximum lateness.

The basis of induction with $l = 1$ is trivial since we only have to check whether the amount of raw materials and the length of $[0, t_1]$ are large enough to complete the jobs.

Suppose (1)–(3) holds then we simply take the zero maximum lateness schedule for $P(l-1)$ and in interval $[t_{l-1}, t_l]$ schedule the remaining $n'_{j,l}$, $j \in J$, jobs. Since (1) and (2) hold and all remaining jobs, $n'_{j,l}$, $j \in J$, have the same due date t_l, the resulting schedule has also zero maximum lateness.

Now, suppose in $P(l)$ there exists a schedule with zero maximum lateness. Then (1) and (2) are obviously satisfied and it remains to show that $P(l-1)$ has a zero maximum lateness schedule. Clearly, in time interval $[t_{l-1}, t_l]$ there are only individual jobs having due date t_l; otherwise it would not be a schedule for $P(l)$ with zero maximum lateness. Since from time t_{l-1} we have enough raw materials to complete all jobs we assign to the interval $[t_{l-1}, t_l]$ the most expensive (raw material consuming) jobs having the due date t_l. Since all the jobs have unit processing time we can either completely fill in the interval or fill it in with all jobs having the due date t_l. This implies that there is a zero maximum lateness schedule for the reduced problem with changed number of jobs having due date t_{l-1}: $n_{j,l-1} := n_{j,l-1} + n_{j,l} - n'_{j,l}$, $j \in J$. Notice that this is exactly the problem $P(l-1)$.

Estimating the running time of the algorithm $\mathcal{A}$ we notice that the most expensive step is the fourth basic step. It requires $O(s|L|)$ time. The number of basic steps is $|L|$, which implies the total running time of the algorithm is $O(s|L|^2)$. □

Proposition 5.3.14. *The job-oriented description output by $\mathcal{A}$ correctly defines a schedule with zero maximum lateness. For any individual job the optimal completion time can be obtained in time $O(s|L| + |L| \log |L|)$.*

Proof. The first statement of the proposition is a straightforward consequence of Proposition 5.3.13.

To estimate the running time of computing $C_{j,i}$, $j \in J$, $1 \leq i \leq n_j$, in the job-oriented description we notice that calculation of the corresponding l at the first step takes $O(|L| \log |L|)$ time and calculation of $C_{j,i}$ afterwards takes $O(s|L|)$ time. □

Notice that increasing due dates by a constant L_{max} for all individual jobs we enforce zero maximum lateness. Any other increment of due dates which is strictly less than L_{max} does not provide zero maximum lateness. Notice also that there is an upper bound $t_{|L|} + \sum_{j \in J} n_j$ on $L_{\max}$ which is polynomial in the input size of the problem. Therefore, combining the standard binary search over values of L_{max} with the algorithm $\mathcal{A}$ we can in polynomial time obtain an optimal solution for $1|rm = 1, p_j = 1|L_{max}$. Thus, we have a theorem.

Theorem 5.3.15. *The problem $1|rm = 1, p_j = 1|L_{max}$ is polynomially solvable, i.e., in polynomial time we compute $L_{\max}$ and output a polynomial size polynomially computable job-oriented description of an optimal schedule.*

5.3.9 $1|n_j = 1, rm = 1, s_t = 1|L_{max}$

In Section 5.3.3 we observed the close relation (equivalency) of the problems $1|rm = 1, s_t = 1|C_{max}$ and $F2|rm = 0|C_{max}$. Applying exactly the same reduction, one can show that $F2|n_j = 1, rm = 0|L_{max}$ is reducible to $1|n_j = 1, rm = 1, s_t = 1|L_{max}$. By Lenstra, Rinnooy Kan, and Brucker [81] the recognition version of $F2|n_j = 1, rm = 0|L_{max}$ is strongly NP-complete. Therefore, the recognition version of $1|n_j = 1, rm = 1, s_t = 1|L_{max}$ is strongly NP-complete as well.

5.3.10 $1|n_j = 1, rm = 2, p_j = 1|L_{max}$

In Section 5.3.4 we show that $1|n_j = 1, rm = 2, p_j = 1|C_{max}$ is strongly NP-hard and its recognition version is NP-complete. Using this result let us briefly show that the recognition version of $1|n_j = 1, rm = 2, p_j = 1|L_{max}$ is also strongly NP-complete.

Consider the recognition version of the problem $1|n_j = 1, rm = 2, p_j = 1|C_{max}$ where we are looking for a feasible schedule with makespan at most C. Let us reduce this problem to the recognition version of $1|n_j = 1, rm = 2, p_j = 1|L_{max}$. Given an instance of the recognition version of $1|n_j = 1, rm = 2, p_j = 1|C_{max}$, define $d_j = C, j \in J$, and consider the given instance with objective to minimize the maximum lateness. Clearly, there is a feasible schedule with makespan at most C if and only if there is a feasible schedule with zero maximum lateness. Since recognition version of $1|n_j = 1, rm = 2, p_j = 1|L_{max}$ is clearly in NP, we conclude that this problem is strongly NP-complete.

5.3.11 $1|n_j = 1, ddc|L_{max}$

Since dedicated supply schedules are generalizations of the static release dates, and minimization of maximum lateness on one machine with release dates is strongly NP-hard, see [81], we conclude that $1|n_j = 1, ddc|L_{max}$ is strongly NP-hard as well.

5.3.12 An approximation algorithm for $1|rm, p_j = 1|L_{max}$

Holthuijsen, and Van de Klundert in [60] show that the EDD rule provides a schedule for $1|n_j = 1, rm, p_j = 1|L_{max}$ with a worst-case ratio of 2. Combining this result with Theorems 5.3.11 and 5.3.12, we claim that we can construct in polynomial time a polynomial space description of a schedule for $1|n_j = 1, rm, p_j = 1|L_{max}$ with a worst-case ratio of 2.

5.3.13 $1|n_j = 1, rm = 1, s_t = 1, p_j = 1, d_j = D|\sum_j w_j U_j$

Despite the very restrictive setting and simplicity of many parameters, the problem is difficult to solve. It clearly has a number theory flavor. To illustrate this we show that the problem is a special case of the well known KNAPSACK problem.

Let x_j, $j \in J$, be a variable indicating whether job j is late or not. We write the following integer linear program being the KNAPSACK formulation and modelling our scheduling problem:

$$\max_{x} \sum_{j \in J} w_j x_j; \tag{5.12}$$

$$\sum_{j \in J} a_j x_j \leq D; \tag{5.13}$$

$$x_j \in \{0,1\}, \quad j \in J. \tag{5.14}$$

The objective function (5.12) and the integrality constraints (5.14) are clear. Let us clarify the knapsack constraint (5.13). Since we have just one additional unit of raw material at a time we can collect to time moment D at most D units of the material. So, the knapsack constraint in the model is clearly valid since for the set of jobs which are not late, we can not use more units of raw material than we have up to moment D. Sufficiency of the knapsack constraint follows from the fact that, given a feasible solution of the KNAPSACK problem, we can easily construct the schedule of the jobs which are not late. To do this (1) take a set of the jobs corresponding to the variables of the KNAPSACK problem with value 1, and arrange this set in non-increasing order of a_j; (2) schedule the first job of the order in time interval $[D-1, D]$, the second one in interval $[D-2, D-1]$, and so on. By the knapsack constraint and by the condition that all jobs have unit processing times we have that all jobs corresponding to the variables of the KNAPSACK with value 1 can be scheduled before time D.

Since the KNAPSACK problem is NP-hard, see [34], the problem $1|n_j = 1, rm = 1, s_t = 1, p_j = 1, d_j = D|\sum_j w_j U_j$ is NP-hard as well. It is well known also, see, e.g., [93], that the KNAPSACK problem can be solved in pseudopolynomial time by a straightforward dynamic programming algorithm. This implies that the problem $1|n_j = 1, rm = 1, s_t = 1, p_j = 1, d_j = D|\sum_j w_j U_j$ also can be solved in pseudopolynomial time. Moreover, we can use the dynamic programming algorithm to derive a FPTAS for the problem (the detailed FPTAS for KNAPSACK can be found in [93], and for the general approach to derive FPTAS given a dynamic programming algorithm see [110]).

To manage the high multiplicities in demand we have to slightly modify the integer linear program (5.12)–(5.14). If instead of binary variables $x_j \in \{0,1\}, j \in J$, we consider integer variables $x_j \in \mathbb{Z}^+$ from the range $0 \leq$

$x_j \leq n_j$, then new integer linear program correctly describes our scheduling problem with high multiplicities in demand. Here, integer variable x_j, $j \in J$, indicates how many individual jobs of type j are not late. The problem is known as INTEGER KNAPSACK, see [34].

5.3.14 $1|n_j = 1, rm = 1, s_t = 1, p_j = 1|\sum_j w_j T_j$

In this section we show that minimization of the total weighted tardiness is also difficult even in very restricted setting: $1|n_j = 1, rm = 1, s_t = 1, p_j = 1|\sum_j w_j T_j$. First, we notice that collecting the required raw materials (together with processing) takes for the job $j \in J$ time a_j, and there is an optimal solution in which during this period there is no other job that collects the raw materials. Now, it is easy to reduce the strongly NP-hard problem $1|n_j = 1, rm = 0|\sum_j w_j T_j$, see [78] and [81], to the problem $1|n_j = 1, rm = 1, s_t = 1, p_j = 1|\sum_j w_j T_j$.

Define

- $J = J'$ where J' is the set of jobs given in $1|n_j = 1, rm = 0|\sum_j w_j T_j$;
- $a_j = p'_j,\ w_j = w'_j,\ d_j = d'_j,\ j \in J$ where p'_j, w'_j, d'_j are processing time, weight and due date of job $j \in J'$ respectively.

Now, let job $j \in J'$ in $1|n_j = 1, rm = 0|\sum_j w_j T_j$ has completion time C_j if and only if j has the completion time C_j also in $1|n_j = 1, rm = 1, s_t = 1, p_j = 1|\sum_j w_j T_j$. From the observation above it follows that every feasible schedule for $1|n_j = 1, rm = 0|\sum_j w_j T_j$ corresponds to a feasible schedule in $1|n_j = 1, rm = 1, s_t = 1, p_j = 1|\sum_j w_j T_j$ and vice versa. The objective values of the corresponding solutions are the same, and therefore we have a complete polynomial reduction.

5.4 Models and problems for periodic versions

In Remark 5.2.5 we have introduced two descriptions of input data in high multiplicity scheduling problems. One is based on the explicit representation of data in time moments when something is changing, for example, at moments when the raw materials are delivered or at due dates of individual

jobs. The other one is a regular supply schedule where we receive identical amount of raw material every so many time units. Symmetrically, we have introduced a regular demand schedule where the due dates for a job type appear regularly, once a certain period, for the same amount of individual jobs. In practice, however, we often observe a combination of these two descriptions. Namely, in a fixed time interval we have an irregular supply/demand but the interval with this irregular setting appears repetitively (regularly) many times (for instance, we can think of delivery of raw materials at Tuesdays and Thursdays every week during one year). Moreover, from production planning perspectives we are interested in a cyclic production policy having a cycle length equal to the length of the repeating supply/demand interval.

In this section we introduce a generic algorithm that solves a variety of such problems to optimality. Consider the problem $1|ddc, p_j = 1|\sum_{j\in J}\sum_{i=1}^{n_j} w_j T_{j,i}$. To simplify further explanations we construct an algorithm for this relatively simple and non-periodic problem and then we show how to extend the algorithm for the problems we are interested in.

5.4.1 Basic ideas of the algorithm

The announced algorithm is based on the same branch-and-price approach as in Chapter 4.

First, let us formulate an integer linear program model for the problem. We introduce the time horizon $T = \{1, 2, \ldots, \mathrm{T}\}$, and the sets S_j, $j \in J$, of all feasible schedules of individual jobs of type j. Let the variable $x_{j,s}$ have value one if individual jobs of type $j \in J$ are produced in the unit time slots contained in $s \in S_j$ and zero otherwise. This leads to the following *set partitioning* formulation:

$$\min_{x} \sum_{j\in J}\sum_{s\in S_j} c_{j,s}x_{j,s} \tag{5.15}$$

subject to

$$\sum_{s\in S_j} x_{j,s} = 1, \qquad j \in J; \tag{5.16}$$

$$\sum_{j\in J}\sum_{s\in S_j : t\in s} x_{j,s} \le 1, \qquad t \in T; \tag{5.17}$$

$$x_{j,s} \in \{0,1\}, \qquad j \in J,\ s \in S_j. \tag{5.18}$$

Here $c_{j,s}$, $j \in J, s \in S_j$, is the contribution of schedule s for jobs of type j to the objective function. Because of equation (5.16) we can choose only one feasible schedule for every job type. Equation (5.17) prohibits the processing of more than one individual job at a time. Notice, that this program just slightly differs from the set partitioning formulation considered in Chapter 4. The only difference is that we have additional restrictions on the sets S_j, $j \in J$. These restrictions are simply requirements that any $s \in S_j$ consists of exactly n_j ones and in the remaining positions of this vector we have zeros.

In the next theorem we show that despite the exponential size of the program, its linear relaxation is solvable in time polynomial in s and T (notice that because of the high multiplicity, T is not polynomial in the input size of the problem).

Theorem 5.4.1. *The linear relaxation of the problem (5.15)–(5.18) is solvable in time polynomial in s and* T.

Proof. Consider the dual problem

$$\max_{u,v} \left(\sum_{j \in J} u_j + \sum_{t \in \{1,2,\ldots,T\}} q_t \right) \tag{5.19}$$

subject to

$$u_j + \sum_{t \in s} q_t \leq c_{j,s}, \quad j \in J,\ s \in S_j; \tag{5.20}$$

$$q_t \leq 0, \quad t \in T. \tag{5.21}$$

The corresponding *pricing problem* (to be shown polynomially solvable) boils down to the following question:

$$\exists\, j \in J, s \in S_j \text{ such that } u_j + \sum_{t \in s} q_t > c_{j,s}?$$

Since in the *pricing problem* different job types are not related to each other, we can decompose the problem into $|J|$ independent subproblems. Consider the subproblem for job type $j \in J$. Construct a directed graph $G = (V, A)$ with a vertex set

$$V = \{v_{0,0}\} \cup \{v_{i,t} : \ 0 \leq i \leq n_j,\ t \in T\}$$

and an arc set

$$A = \{(v_{i,t}, v_{i,t+1}) : \forall i, t\} \cup \{(v_{i,t}, v_{i+1,t+1}) : \forall i, t\}.$$

Calculate the number $D_j(t)$ of individual jobs of type j with the due dates not exceeding time $t \in T$. Define the length of arc $e = (v_{i,t}, v_{i,t+1})$ as

$$\omega(e) = \frac{u_j}{\mathrm{T}} - w_j(D_j(t+1) - i)^+,$$

and the length of arc $e = (v_{i,t}, v_{i+1,t+1})$ as

$$\omega(e) = \begin{cases} \frac{u_j}{\mathrm{T}} + q_{t+1} - w_j(D_j(t+1) - i - 1)^+, & \text{if } S_j(t+1) \geq (i+1)a_j; \\ -\infty, & \text{otherwise.} \end{cases}$$

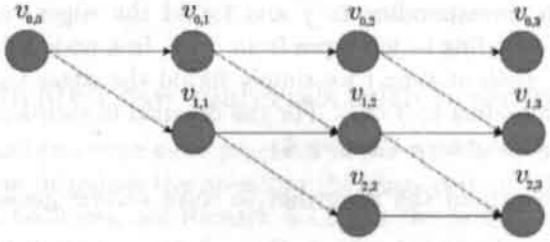

Figure 5.1: Graph G with job multiplicity 2 and $\mathrm{T} = 3$

To clarify the intuition behind this graph let us say that vertex $v_{i,t}$ corresponds to the state of the schedule in time t when i individual jobs of type j have been processed. The arcs correspond to possible evolutions of the states. In principal, every state has only two opportunities for evolution: either in the next time unit we process an individual job of type j or we do not.

Now, let us find in G the longest path from $v_{0,0}$ to $v_{n_j,\mathrm{T}}$. These two vertices correspond to the initial and the final states of any feasible schedule, respectively. Notice that the edge weights are defined in such a way that the length of the longest path from $v_{0,0}$ to $v_{n_j,\mathrm{T}}$ is equal to the maximum value of $u_j + \sum_{t \in s} q_t - c_{j,s}$ over all $s \in S_j$. Take this maximum value and compare it with 0. If it is positive then the pair (j, s) associated with the longest path is

a solution of the *pricing problem*. If the value is non-positive then the answer to the question of the *pricing problem* is negative.

The number of vertices in G is $O(\mathrm{T}^2)$. Therefore calculating the longest path in G takes $O(\mathrm{T}^4)$, see [3]. Thus, the *pricing problem* is solvable in time $O(s\mathrm{T}^4)$. If the *pricing problem* is polynomially solvable then we can find a violated inequality in the dual problem in polynomial time. □

Having a polynomially solvable linear relaxation we can straightforwardly construct an LP-based branch-and-price algorithm that solves the integer program to optimality. As it was described in Chapter 4, the linear program outputs a fractional solution if and only if there exists a pair (j,t) such that $\sum_{s\in S_j:t\in s} x_{j,s}$ is fractional. Now, in the case that we have a fractional solution we introduce two branches: either an individual job of type j is processed at time unit t or it is not. This branching rule eliminates the possibility for $\sum_{s\in S_j:t\in s} x_{j,s}$ to be fractional. In a node of the search tree where we decide to process an individual job of type j at time t let us forbid all edges $(v_{i,t-1}, v_{i,t})$ in the graph corresponding to j and forbid the edges $(v_{i,t-1}, v_{i+1,t})$ in all graphs corresponding to job types from $J \setminus j$. In a node where we decide not to process a j-job at time t we simply forbid the edges $(v_{i,t-1}, v_{i+1,t})$ in the graph corresponding to j only. For the detailed branch-and-price algorithm and more comments see Chapter 4.

Below we extend the algorithm to solve several generalizations of the problem.

Remark 5.4.2. *Notice that the algorithm is fairly universal and applicable to many scheduling problems. It can be adopted to a wide variety of objectives and even to combinations of objectives. For example, the problem can be solved also for total weighted tardiness and earliness, lateness, sum of weighted completion times, etc. To do this we should just redefine in G the lengths of the arcs in the an appropriate way.*

Remark 5.4.3. *Here, we would like to emphasize that the algorithm is very well suitable for high multiplicity problems. Intuitively it is clear that in high multiplicity problems we can consider a schedule of identical jobs as a whole since any permutation of these jobs does not change the objective. This is exactly what the algorithm does: it schedules individual jobs not one-by-one but all the jobs of the same type simultaneously.*

5.4.2 Multiple dedicated raw materials

Consider the problem $1|mddc, p_j = 1|\sum_{j\in J}\sum_{i=1}^{n_j} w_j T_{j,i}$. In this case, in the primal ILP, (5.15)–(5.18), only the structures of sets S_j, $j \in J$ change. The ILP itself, its dual and the *pricing problem* do not change at all.

To take into account changes in the sets S_j, $j \in J$, we only have to take care of sufficiency of the raw materials to process the jobs. In this case *pricing problem* can be modelled as the longest path problem for the same graph G as in the previous section with the arc lengths modified as follows. Redefine the length of an arc $e = (v_{i,t}, v_{i+1,t+1})$ by

$$\omega(e) = \begin{cases} -\infty, & \text{if there exists } r \in R_j \text{ such} \\ & \text{that } S_r(t+1) < (i+1)a_{j,r}; \\ \frac{u_j}{\Gamma} + v_{t+1} - w_j(D_j(t+1) - i - 1)^+, & \text{otherwise.} \end{cases}$$

5.4.3 Dedicated raw materials with inventory costs

To make the problem even more practical in the version with dedicated raw materials we can introduce the inventory (holding) costs in addition to, e.g., total weighted tardiness, see Remark 5.4.2. In the primal ILP we have to change the coefficients $c_{j,s}$, $j \in J, s \in S_j$, in the objective function adding the corresponding contributions of the inventory costs. In the *pricing problem*, to take the inventory costs into account we modify graph G in the following way. We split every vertex $v_{i,t}$ into two vertices $v'_{i,t}$ and $v''_{i,t}$ adjacent by an arc $(v'_{i,t}, v''_{i,t})$. All incoming arcs into $v_{i,t}$ redirect on $v'_{i,t}$ and all outgoing arcs direct from $v''_{i,t}$.

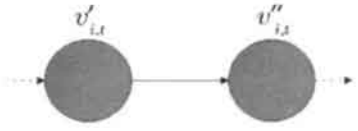

Figure 5.2: Inventory

Keep the old arc lengths the same and define the length of new arcs, $(v'_{i,t}, v''_{i,t})$, $1 \le i \le n_j$, $t \in T$, as a negative value of the total inventory cost

at time t:

$$-\sum_{r\in R_j} g_{r,t}\left(S_r(t) - ia_{j,r}\right),$$

where $g_{r,t}$ is the cost of holding one unit of $r \in R_j$ in inventory during the period $t \in T$. Now, the length of the longest path from $v'_{0,0}$ to $v''_{n_j,\mathrm{T}}$ contains the contribution of the inventory costs to the objective function.

5.4.4 Periodic scheduling

Now, let us discuss an extension of the algorithm for periodic scheduling problems. For simplicity again take the problem $1|ddc, p_j = 1|\sum_{j\in J}\sum_{i=1}^{n_j} w_j T_{j,i}$ but from now on in a periodic context. Given the supply and demand schedules in interval $T = \{1, 2, \ldots, \mathrm{T}\}$. Now, we assume that in the long run in every T time units supply/demand will be repeated in the same quantity, i.e., $s_{r,t} = s_{r,t+\mathrm{T}}$ for any $r \in R$ and $t \in \mathbb{Z}$, and $d_{j,t} = d_{j,t+\mathrm{T}}$ for any $j \in J$ and $t \in \mathbb{Z}$. It is required to find a schedule of individual jobs with period length T which minimizes a certain long run criterium.

To create a periodic schedule we suggest to consider all possible initial (the same as final) states and the corresponding state evolutions. To do this let us introduce an extended set of vertices and a new set of arcs.

$$\begin{aligned} V = \; & \left\{v_{0,0,s} : 0 \le s \le \textstyle\sum_{t\in T} s_{j,t}\right\} \bigcup \\ & \left\{v_{i,t,s} : \; 0 \le i \le n_j;\; t \in T;\; 0 \le s \le \textstyle\sum_{t\in T} s_{j,t}\right\}, \\ A = \; & \left\{(v_{i,t,s}, v_{i,t+1,s+s_{j,t+1}}) : \forall i, t, s\right\} \bigcup \\ & \left\{(v_{i,t,s}, v_{i+1,t+1,s+s_{i,t+1}-a_i}) : \forall i, t, s\right\}. \end{aligned}$$

Now, define the lengths of arcs in the same way as we have done in Section 5.4.1 and find the longest paths between all pairs of vertices $v_{0,0,s}$ and $v_{n_j,\mathrm{T},s}$, where $0 \le s \le \sum_{t\in T} s_{j,t}$. The longest path overall $j \in J$, provides a solution for the *pricing problem* in the periodic version.

One can easily see that in the periodic version graph G is not polynomially encoded in terms of T and $\log \max_{r\in R, t\in T} s_{r,t}$: the number of vertices in the graph is $O(\mathrm{T}^{|R|+1}(\max_{r\in R,t\in T} s_{r,t})^{|R|})$. On the other hand, if the number $\max_{r\in R,t\in T} s_{r,t}$ is bounded from above by a polynomial function on the input

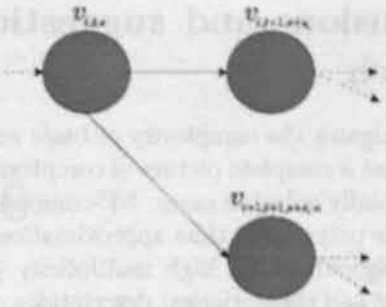

Figure 5.3: Evolution graph with additional supply s_t and consumption a

size, and L is bounded from above by a constant then the encoding of the graph remains polynomial in s and T. In this case all results obtained above are valid for the periodic version also.

Surprisingly, we can easily apply the algorithm to a generalization of the periodic version where raw materials can be used by jobs of different types. Suppose, two or more job types require the same raw material. Additionally we assume that (1) there are no inventory costs and we are free to choose inventory for the initial/final state of the schedule; (2) the total periodic supply of a raw material equals the total periodic demand on this material

$$\sum_{t \in T} s_{r,t} = \sum_{j \in J} n_j a_{j,r}, \quad r \in R. \tag{5.22}$$

By condition (5.22) any chosen initial/final inventory becomes renewable in every cycle. With large enough and renewable initial inventory any schedule satisfying demand becomes feasible with respect to raw materials. Thus, resources (raw materials) and their supply schedules are not restricting factors in the periodic problem. It means we can allow the sharing resource types by different types of jobs. Notice that this claim is valid only in case of no inventory costs and free choice of the initial state.

5.5 Conclusions and suggestions for further research

This chapter investigates the complexity of basic supply chain management problems and present a complete picture of complexity results including algorithms for polynomially solvable cases, NP-completeness proofs for difficult problems, and some polynomial time approximation algorithms.

Dealing with algorithms for high multiplicity problems, we apply notions of job-oriented and time-oriented descriptions of schedule introduced in Chapter 2. We extend this with the notion of a sequence-oriented description of a schedule. We motivate this extension by showing that for several high multiplicity problems we can construct in polynomial time a polynomially encoded sequence-oriented optimal schedule while we have not been able to construct job- or time-oriented one. We present also some cases where in polynomial time we can construct a polynomially encoded optimal solution using a job-oriented description but are unable to compute the optimal objective value.

Finally, we introduce the following list of open questions to this chapter:

1. Can we obtain in polynomial time a polynomial size and polynomially computable job-oriented description for problem $1|ddc|C_{\max}$ with regular supply given as in Remark 5.2.5?

2. How to compute the value of $L_{\max}$ in the problem $1|rm = 0|L_{\max}$ with regular demand given as in Remark 5.2.5?

3. How to solve the problem $1|rm = 1, p_j = 1|L_{\max}$ in case of regular demand given by a compact mapping, see Remark 5.2.5?

4. What is the optimal cycle length for periodic versions of the various scheduling problems? In particular, like in Chapter 3 we are asking the question: if T is the length of the supply-demand cycle then can an optimal solution with period $l \times \mathrm{T}, l \in \mathbb{Z}^+$, be better than the optimal solution with period T?

Chapter 6

Project scheduling with irregular costs: Complexity, approximability, and algorithms

6.1 Introduction

Due to its practical importance, the discrete time-cost tradeoff problem for project networks has been studied in various contexts by many researchers over the last fifty years; see Kelley & Walker [66] for an early reference. The modern treatment of this problem started with the dynamic programming approaches of Hindelang & Muth [53] and Robinson [97], and with an enumeration algorithm by Harvey & Patterson [50]. An up-to-date overview on the discrete time-cost tradeoff problem is Chapter 4 of the survey by Brucker, Drexl, Möhring, Neumann & Pesch [16]. In this chapter, we look at a generalization of the classical discrete time-cost tradeoff problem where the costs depend on the exact starting *and* completion times of the activities.

6.1.1 Statement of the problem

Formally, we consider instances that are called *projects* and that consist of a finite set $\mathcal{A} = \{A_1, \ldots, A_n\}$ of *activities* together with a partial order $\prec$ on $\mathcal{A}$. All activities are available for processing at time zero, and they

must be completed before a global project deadline T. Hence, the set of possible starting and completion times of the activities is $\{0, 1, \ldots, T\}$. The set of intervals over $\{0, 1, \ldots, T\}$ (the so-called *realizations* of the activities) is denoted by $\mathcal{R} = \{(x, y) \mid 0 \leq x \leq y \leq T\}$. For every activity A_j, there is a corresponding cost function $c_j : \mathcal{R} \to \mathbb{R}^+ \cup \{\pm\infty\}$ that specifies for every realization $(x, y) \in \mathcal{R}$ a non-negative cost $c_j(x, y)$ that is incurred if the activity is started at time x and completed at time y. A *realization* of the project is an assignment of the activities in $\mathcal{A}$ to the intervals in $\mathcal{R}$. A realization is *feasible* if it obeys the precedence constraints: For any A_i and A_j with $A_i \prec A_j$, activity A_j is not started before activity A_i has been completed. The cost of a realization is the sum of the costs of all activities in this realization. The goal is to find a feasible realization of minimum cost. This problem is called *min-cost* project scheduling with irregular costs, or min-cost PSIC for short.

A closely related problem is *max-profit* project scheduling with irregular costs, or max-profit PSIC for short. Instead of a cost functions c_j for activity A_j, here we have profit functions $p_j : \mathcal{R} \to \mathbb{R}^+ \cup \{\pm\infty\}$ that specify for every realization of A_j the resulting profit. The goal is to find a feasible realization of maximum profit. Such a profit may, for instance, represent the cost reduction for the project, if a deadline is stretched and an activity becomes less urgent. Clearly, the min-cost and the max-profit version are polynomial time equivalent: The transformations $c_j := \text{const}_1 - p_j$ and $p_j := \text{const}_2 - c_j$ with sufficiently large constants const_1 and const_2 translate one version into the other. However, the two versions seem to behave quite differently with respect to their approximability.

6.1.2 Special cases and related problems

Various special cases arise if the cost and profit functions satisfy additional properties. A cost function c is *monotone*, if $[x_1, y_1] \subseteq [x_2, y_2]$ implies $c(x_1, y_1) \geq c(x_2, y_2)$. A profit function p is *monotone*, if $[x_1, y_1] \subseteq [x_2, y_2]$ implies $p(x_1, y_1) \leq p(x_2, y_2)$. The intuition behind these concepts is that short and quick executions should be more expensive than long and slow executions. It is readily seen that the general version of PSIC is equivalent to the monotone version with respect to computational complexity and approximability.

Another interesting special case arises, if $y_1 - x_1 = y_2 - x_2$ implies $c(x_1, y_1) = c(x_2, y_2)$ and $p(x_1, y_1) = p(x_2, y_2)$. In this special case, the cost

and the profit of an activity only depend on the length of its realization. This special case actually is equivalent to the DEADLINE problem for the discrete time-cost tradeoff problem: The deadline T is hard, and the goal is to assign lengths to activities such that the overall cost is minimized. Only recently, De, Dunne, Gosh & Wells [24] proved that this problem is NP-hard in the strong sense. Skutella in [103] gives some positive approximability results, and Deineko & Woeginger [26] give some in-approximability results for bicriteria versions. All negative results in this chapter are proved for the DEADLINE problem, the weakest variant of PSIC. All positive results in this chapter are proved for the most general version of PSIC.

In another special case, for every activity A_j there is a number L_j such that $c_j(x, y) < \infty$ if and only if $y - x = L_j$. In other words, activity A_j must be realized by an interval of length exactly L_j. This special case is classical project scheduling with fixed processing times. Chang & Edmonds [18] proved that this case is polynomial time equivalent to the min-cut problem in graphs; hence, this case is polynomially solvable. Project scheduling with fixed processing times and some of its variants were also studied by Maniezzo & Mingozzi [82] and by Möhring, Schulz, Stork & Uetz [91].

6.1.3 Results

We derive several positive and negative statements on the complexity and the approximability of min-cost and max-profit PSIC for several natural classes of precedence constraints. Our results are the following:

1. Interval orders (Section 6.2). The min-cost and the max-profit version of the DEADLINE problem (and of their PSIC generalizations) are NP-hard and in-approximable even for interval orders. We establish a close (approximation preserving) connection of the min-cost DEADLINE problem to minimum vertex cover and of the max-profit DEADLINE problem to maximum independent set. All in-approximability results for these graph problems carry over to the DEADLINE problems. As an immediate consequence, unless P=NP the min-cost DEADLINE problem can not have a polynomial time approximation algorithm with worst case ratio strictly better than 7/6. This is quite an improvement over an earlier in-approximability result of Deineko & Woeginger [26] that only established APX-hardness for this problem.

2. Orders of bounded height (Section 6.3). If the height of the precedence constraints is bounded by 2, then the DEADLINE problems and its PSIC generalizations are NP-hard and in-approximable. However, if the height of the precedence constraints is bounded by 1, then min-cost and max-profit PSIC both can be solved in polynomial time. The main idea is to translate these project scheduling problems into a maximum weight independent set problem in an underlying vertex-weighted bipartite graph.

3. Orders of bounded width (Section 6.4). If the width of the precedence constraints is bounded by some fixed constant d, then min-cost and max-profit PSIC both can be solved in polynomial time $O(n^d T^{2d+1})$. The algorithm is based on simple dynamic programming over the time axis, but the details are somewhat messy.

4. Series parallel orders (Section 6.5). For series parallel precedence constraints, min-cost and max-profit PSIC can be solved in polynomial time $O(nT^3)$ by dynamic programming. This result builds on the approaches of Frank, Frisch, van Slyke & Chou [20] and Rothfarb, Frank, Rosenbaum, Steiglitz & Kleitman [33] for the classical discrete time-cost tradeoff problem, and extends them to the more general problems max-profit and min-cost PSIC.

5. Finally in Section 6.6, we discuss how the complexity of min-cost and max-profit PSIC depends on the encoding of the input. We present an example of PSIC with two activities A and B, with the precedence constraint $A \prec B$, and with (very) specially defined cost/profit functions. For this example, even the DEADLINE problem is NP-hard.

6.1.4 Technical remarks

For costs and profits we allow any values from $\mathbb{R}^+ \cup \{\pm\infty\}$, that is the non-negative numbers together with plus/minus infinity. This should be seen as a useful and simple convention for specifing the input: Whenever a cost equals $+\infty$ or a profit equals $-\infty$, then the corresponding realization is forbidden. Of course this convention leads to instances that do not have any feasible realization with finite cost or profit, but these instances are easily recognized and singled out in polynomial time by the following greedy algorithm: *"In*

every step, select a yet unrealized activity A for which all predecessors have already been realized. Choose for A the realization (x, y) of finite cost (respectively, finite profit) with smalles value y." This algorithm gets stuck if and only if there is no project realization of finite cost (respectively, finite profit).

Hence, throughout the chapter we will restrict ourselves to instances that allow at least one realization in which all costs (respectively, all profits) are non-negative reals. A more compact representation of the input only specifies those realizations of activities that have finite costs/profits.

6.2 Interval orders

In this section we will derive a number of negative results for problem PSIC under interval orders. An *interval order* on a set $\mathcal{A} = \{A_1, \ldots, A_n\}$ is specified by a set of n intervals $I_1, \ldots, I_n$ along the real line. Then $A_i \rightarrow A_j$ holds if and only if the interval I_i lies completely to the left of the interval I_j, or if the right endpoint of I_i coincides with the left endpoint of I_j. See, e.g., Möhring [90].

The central proof in this section will be done by a reduction from the NP-hard INDEPENDENT SET problem in graphs; see Garey & Johnson [34]: Given a graph $G = (V, E)$ and a bound z, does G contain an independent set (a set that does not induce any edges) of cardinality z? Without loss of generality, we assume that $V = \{1, \ldots, q\}$.

We construct a project with deadline $T = 3q$ for max-profit PSIC. This project contains the activities listed below. For every activity A, we define a so-called *crucial* interval $I(A)$ that will be used to specify the interval order.

- For every vertex $i \in V$, there is a corresponding activity A_i. If A_i is realized by an interval of length zero, then its profit is $-\infty$; for an interval of length 1 or 2 the profit is 0, and for any longer realization the profit is 1. The crucial interval $I(A_i)$ for A_i is $[3i - 3, 3i]$.

- For every edge $\langle i, j \rangle \in E$ with $i < j$, there is a corresponding activity $A_{i,j}$. If $A_{i,j}$ is realized by an interval of length $3j - 3i - 2$ or more then its profit is 0, and for shorter intervals its profit is $-\infty$. The crucial interval $I(A_{i,j})$ is $[3i, 3j - 3]$.

- For $t = 0, \ldots, q$ there are so-called *blocking* activities B_t and C_t. If they are executed for at least $3t$ time units, then they bring profit 0,

and for shorter intervals they bring profit $-\infty$. The crucial intervals for them are $I(B_t) = [0, 3t]$ and $I(C_t) = [3q - 3t, 3q]$.

The precedence constraints among these activities are defined as follows: For activities X and Y, $X \prec Y$ holds if and only if the crucial interval $I(X)$ lies completely to the left of the crucial interval $I(Y)$, or if the right endpoint of $I(X)$ coincides with the left endpoint of $I(Y)$. Note that this yields an interval order on the activities. Moreover, for every edge $\langle i, j \rangle \in E$ with $i < j$ this implies $A_i \prec A_{i,j} \prec A_j$.

Lemma 6.2.1. *If the graph G has an independent set W, then the constructed project has a feasible realization with profit $|W|$.*

Proof. Let $W \subseteq V$ denote the independent set of cardinality z. If $i \in W$, then process activity A_i with profit 1 during $[3i - 3, 3i]$. If $i \notin W$, then process it with profit 0 during $[3i - 2, 3i - 1]$. All other activities are processed at profit 0: Every blocking activity is processed during its crucial interval. For an edge $\langle i, j \rangle \in E$ with $i < j$ and $i \notin W$, process activity $A_{i,j}$ during $[3i - 1, 3j - 3]$; this puts $A_{i,j}$ after A_i and before A_j exactly as imposed by the precedence constraints. For an edge $\langle i, j \rangle \in E$ with $i < j$ and $i \in W$, process activity $A_{i,j}$ during $[3i, 3j - 2]$. Since $i \in W$, its neighbor j cannot be also in W; hence A_j is processed during $[3j - 2, 3j - 1]$ and after $A_{i,j}$, exactly as imposed by $A_i \prec A_{i,j} \prec A_j$.

Since in this realization activity A_i brings profit 1 if and only if $i \in W$, this realization has profit $|W|$. Moreover, it can be verified that all precedence constraints indeed are satisfied. □

Lemma 6.2.2. *If the constructed project has a feasible realization with profit $p \geq 1$, then the graph G has an independent set W with $|W| = p$.*

Proof. We first establish three simple claims on such a feasible project realization. The first claim is that (in any feasible realization with positive profit) the processing of every blocking activity must exactly occupy its crucial interval. Indeed, consider the activities B_t and C_{q-t} with their crucial intervals $I(B_t) = [0, 3t]$ and $I(C_t) = [3t, 3q]$. Since the total profit is positive, B_t is processed for at least $3t$ and C_{3q-t} is processed for at least $3q - 3t$ time units. Since B_t is a predecessor of C_{q-t}, they together cover the whole time horizon $[0, 3q]$; this fixes them in their crucial intervals.

The second claim is that every activity A_i is processed somewhere within its crucial time interval $[3i-3, 3i]$. By our first claim activity B_{i-1} completes at time $3i-3$ and activity C_{q-i} starts at time $3i$. Since $B_{i-1} \prec A_i \prec C_{q-i}$, activity A_i cannot start before time $3i-3$ and cannot end after time $3i$.

The third claim is that there exist exactly p activities A_i that exactly occupy their crucial intervals. By construction of the project all the profit results from the activities A_i, and A_i brings positive profit only in case it is executed for at least three time units. By our second claim, A_i cannot be executed for more than three time units. Hence, each activity A_i that brings positive profit occupies its crucial interval $[3i-3, 3i]$.

Now, we are ready to prove the statement in the lemma. Consider the set $W \subseteq V$ that contains vertex i if and only if A_i occupies its crucial interval $[3i-3, 3i]$. We claim that W is an independent set. Suppose otherwise, and consider $i, j \in W$ with $i < j$ and $\langle i, j \rangle \in E$. Then A_i occupies $[3i-3, 3i]$, and A_j occupies $[3j-3, 3j]$, and $A_i \prec A_{i,j} \prec A_j$ holds. Hence, $A_{i,j}$ is processed during the $3j-3i-3$ time units between $3i$ and $3j-3$. But in this case its profit is $-\infty$, and we get the desired contradiction. Hence, W is an independent set, and by our third claim $|W| = p$. □

Theorem 6.2.3. *Max-profit project scheduling with irregular costs is NP-hard even for interval order precedence constraints. For any $\varepsilon > 0$, the existence of a polynomial time approximation algorithm for max-profit PSIC for projects with n activities*

- *with worst case ratio $O(n^{1/4-\varepsilon})$ implies P=NP,*
- *with worst case ratio $O(n^{1/2-\varepsilon})$ implies ZPP=NP.*

Proof. NP-hardness follows from the Lemmas 6.2.1 and 6.2.2. The constructed reduction preserves objective values. It translates graph instances with independent sets of size z into project instances with realizations of profit z, and thus it is approximation preserving in the strongest possible sense. For a graph with q vertices, the corresponding project consists of $O(q^2)$ activities. Håstad [51] proved that the clique problem in n-vertex graphs (and hence also the independent set problem in the complement of n-vertex graphs) cannot have a polynomial time approximation algorithm with worst case guarantee $O(n^{1/2-\varepsilon})$ unless P=NP, and it cannot have a polynomial time approximation algorithm with worst case guarantee $O(n^{1-\varepsilon})$ unless ZPP=NP. Since the blow-up in our construction is only quadratic, the theorem follows. □

In the VERTEX COVER problem, the goal is to find a minimum cardinality vertex cover (a subset of the vertices that touches every edge) for a given input graph. Note that vertex covers are the complements of independent sets. We denote by τ_{VC} the approximability threshold for the vertex cover problem, i.e., the infimum of the worst case ratios over all polynomial time approximation algorithms for this problem. Håstad [51] proved that $\tau_{VC} \geq 7/6$ unless P=NP, and it is widely believed that $\tau_{VC} = 2$.

Theorem 6.2.4. *Min-cost project scheduling with irregular costs is NP-hard even for interval order precedence constraints. The existence of a polynomial time approximation algorithm for min-cost PSIC with worst case ratio better than τ_{VC} would imply P=NP.*

Proof. By a slight modification of the above construction. For activities $A_{i,j}$ and for blocking activities, we replace low profit $-\infty$ by high cost ∞, and the neutral profit 0 by the neutral cost 0. For activities A_i, we replace low profit $-\infty$ by high cost ∞, profit 0 by cost 1, and profit 1 by cost 0. It can be shown that there exists a realization of cost c for the constructed project, if and only if there exists an independent set of size $q - c$ for the graph, if and only if there exists a vertex cover of size c for the graph. Hence, this reduction preserves objective values. □

Corollary 6.2.5. *For the discrete time/cost tradeoff problem, the existence of a polynomial time approximation algorithm with worst case ratio better than τ_{VC} for the DEADLINE problem would imply P=NP.* □

6.3 Orders of bounded height

In this section we will derive a positive result for the project scheduling problem with irregular costs under orders of bounded height. The *height* of an ordered set is the number of elements in the longest chain minus one. Precedence constraints of height 1 are sometimes also called *bipartite* precedence constraints; see, e.g., Möhring [90].

Theorem 6.3.1. *Max-profit and min-cost project scheduling with irregular costs are NP-hard and APX-hard even when restricted to precedence constraints of height two.*

Proof. Deineko & Woeginger [26] establish APX-hardness for the min-cost DEADLINE version of the discrete time/cost tradeoff problem. Their reduction produces instances of height 2 for min-cost PSIC, and it is straightforward to adapt the construction to max-cost PSIC. □

In the rest of this section we will concentrate on the max-profit PSIC for precedence constraints of height 1, and we will derive a polynomial time algorithm for it. Consider such an instance where all profits are either $-\infty$ or non-negative, and classify the activities into two types. The *A-activities* $A_1, \ldots, A_a$ do not have any predecessors, and the *B-activities* $B_1, \ldots, B_b$ do not have any successors. The only precedence constraints are of the type $A_i \rightarrow B_j$, that is from A-activities to B-activities. We start with a preprocessing phase that simplifies this instance somewhat.

- If there exists some activity that neither has a predecessor nor a successor, it is completely independent from the rest of the instance. We process this activity at the maximum possible profit, and remove it from the instance. From now on we assume that each activity has at least one predecessor or successor, and that consequently the partition into A-activities and B-activities is unique.

- We remove all realizations with profit $-\infty$ from the instance.

- Assume that there is an A-activity A_i with profit function p_i, and that there are two realizations (x, y) and (u, v) for it with $y \leq v$ and $p_i(x, y) \geq p_i(u, v)$. Then the realization (x, y) imposes less restrictions on the successors of A_i and at the same time it comes at a higher profit; so we may disregard this realization (u, v) for A_i. By a symmetric argument, we may clean up the realizations of any B-activity B_j.

- Assume that $A_i \prec B_j$ and that there exists a realization (x, y) of A_i that collides with all surviving realizations of B_j (that is, the endpoint y lies strictly to the right of all possible starting points of B_j). Then we remove realization (x, y) for A_i, since it will always collide with the realization of B_j. Symmetrically, we clean up the realizations of the B-activities.

Lemma 6.3.2. *(i) The original instance has a realization with profit p if and only if the preprocessed instance has a realization with profit p.*

(ii) The surviving realizations for A_i can be enumerated as (x_i^1, y_i^1) , ..., $(x_i^{a(i)}, y_i^{a(i)})$ such that they are ordered by strictly increasing right endpoint and simultaneously by strictly increasing profit for A_i. Similarly, the surviving realizations for B_j can be enumerated as $(u_j^1, v_j^1), \ldots, (u_j^{b(j)}, v_j^{b(j)})$ such that they are ordered by strictly decreasing left endpoint and simultaneously by strictly increasing profit for B_j.

(iii) If the original instance has a realization with non-negative profit, then for every activity A_i (respectively, B_j) there exists a realization in the preprocessed instance that does not collide with any realization of a successor of A_i (respectively, of a predecessor of B_j).

Proof. Statements (i) and (ii) are clear from the preprocessing. To see (iii), consider the realization (x_i^1, y_i^1) that has the smallest left endpoint over all realizations of A_i. Suppose that it collides with some realization (u_j^ℓ, u_j^ℓ) of some successor B_j of A_i. Then this realization of B_j collides with *all* realizations of A_i and would have been removed in the last step of the preprocessing. □

From now on we assume that the conditions in (iii) in Lemma 6.3.2 are satisfied. We translate the preprocessed instance into a bipartite graph with weights on the vertices. The max-profit problem will boil down to finding an independent set of maximum weight in this bipartite graph.

- For every realization (x_i^k, y_i^k) of A_i with profit function p_i, there is a corresponding vertex A_i^k in the bipartite graph. If $k = 1$, then the weight of A_i^k equals $p_i(x_i^1, y_i^1)$. If $k \geq 2$, then the weight of A_i^k equals $p_i(x_i^k, y_i^k) - p_i(x_i^{k-1}, y_i^{k-1})$. Note that all weights are non-negative and that the weight of the first k realizations of A_i equals $p_i(x_i^k, y_i^k)$.

- Symmetrically, the bipartite graph contains for every realization (u_j^ℓ, u_j^ℓ) of activity B_j a corresponding vertex B_j^ℓ. The (non-negative) weights of the vertices B_j^ℓ are defined symmetrically to those of the vertices A_i^k.

- Finally, we put an edge between A_i^k and B_j^ℓ if and only if $A_i \prec B_j$ holds and if the interval $[x_i^k, y_i^k]$ does not lie completely to the left of the interval $[u_j^\ell, u_j^\ell]$.

Lemma 6.3.3. *The profit p of the most profitable realization of the preprocessed project equals the weight of the maximum weighted independent set in the bipartite graph.*

Proof. (Only if) Consider the most profitable realization, and consider the following set S of vertices. If activity A_i is realized as (x_i^k, y_i^k), then put the vertices $A_i^1, A_i^2, \ldots, A_i^k$ into S. The weight of these k vertices equals the profit $p_i(x_i^k, y_i^k)$ of realization (x_i^k, y_i^k). If B_j is realized as (u_j^ℓ, u_j^ℓ), then put the vertices $B_j^1, \ldots, B_j^\ell$ into S. The weight of these ℓ vertices equals the profit of the realization of B_j. By construction, the total weight of S equals the total profit p of the considered realization. Moreover, the set S is independent: If in S some A_i^s was adjacent to B_j^t, then $A_i \prec B_j$ and A_i^k and B_j^ℓ would be adjacent. But this would yield a collision in the execution of A_i and B_j, and the realization would be infeasible.

(If) Consider an independent set S of maximum weight in the bipartite graph. For an activity A_i, consider the intersection of S with $\{A_i^1, \ldots, A_i^{a(i)}\}$. By Lemma 6.3.2.(iii), this intersection is non-empty. Let k denote the largest index such that A_i^k is in S. Since the neighborhood of $A_i^1, \ldots, A_i^{k-1}$ is a subset of the neighborhood of vertex A_i^k, also these $k-1$ vertices are contained in S. Then we realize activity A_i by (x_i^k, y_i^k); the resulting profit $p_i(x_i^k, y_i^k)$ equals the total weight of the vertices $A_i^1, \ldots, A_i^k$ in S. For activity B_j, we symmetrically compute a realization that is based on the maximum index ℓ for which B_j^ℓ is in S. Since A_i^k and B_j^ℓ are not incident in the bipartite graph, the chosen realizations of A_i and B_j do not collide. Hence, this realization is feasible. By construction, the total profit equals the total weight of S. □

Theorem 6.3.4. *Max-profit and min-cost project scheduling with irregular costs are polynomially solvable when restricted to precedence constraints of height one.*

Proof. By Lemma 6.3.3, these problems are polynomial time equivalent to finding a maximum weight independent set in a bipartite graph with non-negative vertex weights. Maximum weight independent set in bipartite graphs is well-known to be polynomially solvable by max-flow min-cut techniques; see Ahuja, Magnanti & Orlin [3]. □

6.4 Orders of bounded width

In this section, we will show that if the width of the precedence constraints is bounded by some fixed constant d, then max-profit PSIC is solvable in polynomial time. For technical reasons, we assume throughout this section that all realizations of length 0 have profit $-\infty$ and hence are forbidden;

all our arguments would also go through without this assumption, but the presentation would become more complicated.

In an ordered set, two elements A_i and A_j are called *in-comparable* if neither A_i is a predecessor of A_j nor A_j is a predecessor of A_i. A set of tasks is an *anti-chain*, if its elements are pairwise in-comparable. The *width* of the order is the cardinality of its largest anti-chain. A well-known theorem of Dilworth [27] states that if the width of an ordered set with n elements equals d, then this set can be partitioned into d totally ordered chains $C_1, \ldots, C_d$. Moreover, it is straightforward to compute such a chain partition in $O(n^d)$ time.

For a given instance of max-profit PSIC of width d, we first compute a chain partition $C_1, \ldots, C_d$, and we denote the number of activities in chain C_j by n_j $(j = 1, \ldots, d)$. Now, let us consider some feasible realization of the project, and let us look at some fixed moment $t + \frac{1}{2}$ in time with $0 \leq t \leq T$. As the chain C_j is totally ordered, at time $t + \frac{1}{2}$, at most one of its activities is under execution. Chain C_j is called *in-active* at time $t + \frac{1}{2}$ if none of its activities is under execution, and otherwise it is *active* at time $t + \frac{1}{2}$.

Definition 6.4.1. *For a feasible realization, the* snapshot *S taken at time $t + \frac{1}{2}$ with $0 \leq t \leq T$ contains the following information:*

(S1) For every chain C_j, one bit of information that specifies whether C_j is active or in-active.

(S2) For every in-active chain C_j, a number IN_j with $0 \leq IN_j \leq n_j$ that specifies the last activity in C_j that was executed before time $t + \frac{1}{2}$. If no activity has been executed so far, then $IN_j = 0$.

(S3) For every active chain C_j, a number ACT_j with $1 \leq ACT_j \leq n_j$ that specifies the current activity of C_j. Moreover, the starting time x_j of the current activity with $0 \leq x_j \leq T - 1$.

For the data in (S1) there are at most 2^d possibilities, for all the numbers IN_j and ACT_j in (S2) and (S3) there are at most $O(n^d)$ possibilities, and for all the starting times in (S3) there are at most $O(T^d)$ possibilities. Since d is a fixed constant, this yields that there are at most $O(n^d T^d)$ snapshots at time $t + \frac{1}{2}$.

Definition 6.4.2. *For any t with $0 \leq t \leq T$ and for any possible snapshot S, we denote by $F[t; S]$ the maximum possible profit that can be earned on*

activities completing before time $t+\frac{1}{2}$ in a feasible project realization whose snapshot at time $t+\frac{1}{2}$ equals S.

If no such feasible realization exists, then $F[t;S]=-\infty$.

We compute all these values $F[t;S]$ by a dynamic programming approach that works through them by increasing t. The initial cases with $t=0$ are trivial, since $F[0;S]$ can only take the values 0 (if there exists a feasible realization with snapshot S at time $\frac{1}{2}$) or $-\infty$ (otherwise). To compute $F[t;S]$ for $t \geq 1$, we check all possibilities for a compatible predecessor snapshot S' at time $t-\frac{1}{2}$ in the following way by considering all the chains separately (the data from snapshots S and S' is represented by un-primed and by primed variables, respectively):

- Chain C_j might be active in S' and in-active in S. Then $\mathrm{IN}_j = \mathrm{ACT}'_j$. The additional profit comes from realizing the ACT'_j-th activity in chain C_j from time x'_j to time t.
- Chain C_j might be in-active in S' and active in S. Then $\mathrm{ACT}_j = \mathrm{IN}'_j+1$ and $x_j = t$. No additional profit is generated.
- Chain C_j might be in-active in S' and S. Then $\mathrm{IN}_j = \mathrm{IN}'_j$. Since no activity can simultaneously be started and completed at time t, no additional profit is generated.
- Chain C_j might be active in S' and S. There are two cases: If the same activity is executed at time $t-\frac{1}{2}$ and at time $t+\frac{1}{2}$, then $\mathrm{ACT}_j = \mathrm{ACT}'_j$ and $x_j = x'_j$, and no additional profit is generated. And if the executed activities at times $t-\frac{1}{2}$ and $t+\frac{1}{2}$ are distinct, then $\mathrm{ACT}_j = \mathrm{ACT}'_j+1$ and $x_j = t$ must hold. The additional profit comes from realizing the ACT'_j-th activity in C_j from time x'_j to time t.

If snapshots S and S' are of this form for all d chains, then we say that S' is a *predecessor* of S. Moreover, we denote the total additionally generated profit over all the chains by $\mathrm{profit}(S',S)$. It can be verified that any snapshot S at time $t+\frac{1}{2}$ has at most $O(T^d)$ predecessors at time $t-\frac{1}{2}$. Then the value $F[t;S]$ can be computed as

$$F[t;S] := \max\{F[t-1;S'] + \mathrm{profit}(S',S) \mid S' \text{ is a predecessor of } S\}. \tag{6.1}$$

In the end, the solution to the instance of max-profit PSIC can be found in $F[T;S^*]$ where S^* is the snapshot at time $T+\frac{1}{2}$ where all chains are in-active

and where $\mathrm{IN}_j = n_j$ holds for $j = 1, \ldots, d$. The time complexity of this dynamic programming algorithm is $O(n^d T^{2d+1})$: Since there are $O(n^d T^d)$ snapshots at time $t + \frac{1}{2}$, we altogether compute $O(n^d T^{d+1})$ values $F[t; S]$. Each value can be computed in $O(T^d)$ time by checking all predecessors in (6.1). By storing appropriate auxiliary information and by performing some backtracking, one can also explicitly compute the optimal feasible realization while increasing the running time only by a constant factor. Since these are standard techniques, we do not elaborate on them.

Theorem 6.4.3. *Max-profit and min-cost project scheduling with irregular costs are polynomially solvable in $O(n^d T^{2d+1})$ time when restricted to precedence constraints of width bounded by the fixed constant d.* □

6.5 Series parallel orders

Precedence constraints are called *series parallel* if (i) they contain a single vertex, or (ii) they form the series composition of two series parallel order, or (iii) they form the parallel composition of two series parallel orders. Only orders that can be constructed via rules (i)–(iii) are series parallel. Here the *series composition* of two orders $(V_1, \prec_1)$ and $(V_2, \prec_2)$ with $V_1 \cap V_2 = \emptyset$ is the order that results from taking their union and making all elements in V_1 predecessors of all elements in V_2, whereas the *parallel composition* of $(V_1, \prec_1)$ and $(V_2, \prec_2)$ simply is their disjoint union. Series parallel precedence constraints are a proper generalization of tree precedence constraints; see, e.g., Möhring [90].

It is well known that a series parallel order can be decomposed in polynomial time into its atomic parts according to the series and parallel compositions. Essentially, such a decomposition corresponds to a rooted, ordered, binary tree where all interior vertices are labelled by s or p (series or parallel composition) and where all leaves correspond to single elements of the order. We associate with every interior vertex v of the decomposition tree the series parallel order $\mathrm{SP}(v)$ that is induced by the leaves of the subtree below v. Note that for the root vertex *root* of the decomposition tree, the corresponding order $\mathrm{SP}(root)$ is the whole ordered set.

Our goal is to design a polynomial time algorithm for max-profit PSIC with series parallel precedence constraints. The usual tool for dealing with series parallel structures is dynamic programming.

Definition 6.5.1. *For a vertex v in the decomposition tree, and for integers x and y with $0 \leq x \leq y \leq T$, we denote by $F[v; x, y]$ the maximum possible profit that can be earned on the activities in SP(v), subject to the condition that all these activities are executed somewhere during the time interval $[x, y]$ such that they obey the precedence constraints.*

If no such feasible realization exists, then $F[v; x, y] = -\infty$.

We compute all these values $F[v; x, y]$ by a dynamic programming approach that starts in the leaves of the decomposition tree, and then moves upwards towards the root.

- If v is a leaf, the order SP(v) consists of a single activity A, and $F[v; x, y]$ is easily computed.
- If v is a p vertex with left child v_1 and right child v_2, then $F[v; x, y] := F[v_1; x, y] + F[v_2; x, y]$
- If v is an s vertex with left child v_1 and right child v_2, then $F[v; x, y] := \max\{F[v_1; x, z] + F[v_2; z, y] : x \leq z \leq y\}$

In the end, the solution to the instance of max-profit PSIC can be found in $F[root; 0, T]$. The time complexity of this dynamic programming algorithm is $O(nT^3)$: To compute the values $F[v; x, y]$ for the $O(nT^2)$ leaves, it is sufficient to look once at every possible realization of every activity; this altogether costs $O(nT^2)$ time. And for the inner vertices v, the corresponding $O(nT^2)$ values can be computed in $O(T)$ time per value. By standard techniques, one can also explicitly compute the optimal feasible realization while increasing the running time only by a constant factor.

Theorem 6.5.2. *Max-profit and min-cost project scheduling with irregular costs are polynomially solvable in $O(nT^3)$ time when restricted to series parallel precedence constraints.* □

6.6 PSIC with compactly encoded inputs

In all the sections above, we assumed that the cost and profit functions are specified *pointwise*, that is, that the input lists for every possible realization $(x, y) \in \mathcal{R}$ of every project the corresponding non-negative cost, respectively the corresponding non-negative profit. In this section, we briefly discuss the

variant where the cost and profit functions can be encoded *compactly* via a fast oracle algorithm.

We present a pathological example for the min-cost version of this variant; a pathological example for the max-profit version can be derived in a similar fashion.

Theorem 6.6.1. *The special case of the DEADLINE problem with only two activities $A \prec B$ and with compactly encoded cost functions is NP-hard in the ordinary sense.*

Proof. The proof is done by a reduction from the NP-hard THREE-SATISFIABILITY problem; see Garey & Johnson [34]: Given a collection $C = \{c_1, c_2, \ldots, c_m\}$ of clauses over a finite set $U = \{x_1, x_2, \ldots, x_n\}$ of logical variables such that every clause contains exactly three literals, does there exist a truth assignment for U that satisfies all the clauses in C?

With every n-bit integer F with bits $f_1, f_2, \ldots, f_n$, we associate a corresponding truth assignment for the variables $x_1, x_2, \ldots, x_n$ that sets x_k =TRUE if $f_k = 1$, and x_k =FALSE if $f_k = 0$. Consider the following instance of the DEADLINE problem with time horizon $T = 2^n$, and with two activities A and B where $A \prec B$:

- If activity A is realized at a length of ℓ with $0 \leq \ell \leq T$, then the resulting cost $c_A(\ell)$ equals $2T - 2\ell$ if the true assignment corresponding to ℓ satisfies the given THREE-SATISFIABILITY instance, and otherwise the cost equals $2T - 2j + 1$.

- For any ℓ with $0 \leq \ell \leq T$, the cost $c_B(\ell)$ of realizing activity B at a length of ℓ equals $2T - 2j$.

Note that the defined cost functions are strictly decreasing in ℓ. The cost function c_A is compactly encoded via the clause set C, and for any given value ℓ it can be evaluated in polynomial time. If there is a satisfying truth assignment, then the optimal cost is $2T$. If there is no satisfying truth assignment, then the optimal cost is $2T + 1$. □

Bibliography

[1] I. Adler, and R.D.C. Monteiro, *A geometric view of parametric linear programming*, Algorithmica **8** (1992), 161–176.

[2] A. Agnetis, *No-wait flowshop scheduling with large lot size*, Annals of Operations Research **70** (1997), 415–438.

[3] T. Ahuja, F. Magnanti, and D. Orlin, *Network flows*, Wiley Interscience, San Francisco, 1994.

[4] H. Aigbedo, and Y. Monden, *A parametric procedure for multiecriterion sequence scheduling for just-in-time mixed-model assembly lines*, International Journal of Production Research **35** (1997), 2543–2564.

[5] M. Van den Akker, *Lp-based solution methods for single-machine sceduling problems*, Ph.D. thesis, Technische Universiteit Eindhoven, Eindhoven, Netherlands, 1994.

[6] S. Anily, C.A. Glass, and R. Hassin, *Scheduling of maintenance services to three machines*, Annals of Operations Research **86** (1999), 375–391.

[7] S. Anily, C.A. Glass, and R. Hassin, *The scheduling of maintenance service*, Discrete Applied Mathematics **82** (1998), 27–42.

[8] N.J. Aquilano, R.B. Chase, and F.R. Jacobs, *Operations management for competitive advantage*, McGraw-Hill/Irwin, New York, NY, 2001.

[9] A. Bar-Noy, R. Bhatia, J.S. Naor, and B. Schiber, *Minimizing service and operation costs of periodic scheduling*, Mathematics of Operations Research **27** (2002), 518–544.

[10] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance, *Branch-and-price: Column generation for solving huge integer programs*, Operations Research **46** (1998), 316–329.

[11] Z. Brakerski, V. Dreizin, and B. Patt-Shamir, *Dispatching in perfectly-periodic schedules*, Working paper, Tel-Aviv University, Tel-Aviv, Israel, 2001.

[12] N. Brauner, and Y. Crama, *Facts and questions about the maximum deviation just-in-time scheduling problem*, Working paper GEMME 0013, University of Liège, Liège, Belgium, 2001.

[13] N. Brauner, Y. Crama, A. Grigoriev, and J. Van de Klundert, *On the complexity of high multiplicty scheduling problems*, Report 35, Laboratoire Leibniz-IMAG, Grenoble, France, 2001.

[14] N. Brauner, G. Finke, and W. Kubiak, *Complexity of one-cycle robotic flow-shops*, Working paper GEMME 0001, University of Liège, Liège, Belgium, 2000.

[15] N. Brauner, V. Jost, and W. Kubiak, *On symmetric Fraenkel's and small deviations conjectures*, Report 54, Laboratoire Leibniz-IMAG, Grenoble, France, 2002.

[16] P. Brucker, A. Drexl, R.H. Möhring, K. Neumann, and E. Pesch, *Resource-constrained project scheduling: Notation, classification, models, and methods*, European Journal of Operational Research **112** (1999), 3–41.

[17] P. Brucker, M.Y. Kovalyov, Y.M. Shafransky, and F. Werner, *Parallel Machine Deadline Batch Scheduling*, Annals of Operations Research **83** (1998), 23–40.

[18] G.J. Chang, and J. Edmonds, *The poset scheduling problem*, Order **2** (1985), 113–118.

[19] B. Chen, C.N. Potts, and G.J. Woeginger, *A review of machine scheduling: Complexity, algorithms and approximability*. Handbook of Combinatorial Optimization (Volume 3) (Editors: D.-Z. Du, and P. Pardalos), Kluwer Academic Publishers, 1998, 21–169.

[20] W.S. Chou, H. Frank, I.T. Frisch, and R. van Slyke, *Optimal design of centralized computer networks*, Networks **1** (1970), 43–58.

[21] J.J. Clifford, and M.E. Posner, *High multiplicity in earliness-tardiness scheduling*, Operations Research **48** (2000), 788–800.

[22] J.J. Clifford, and M.E. Posner, *Parallel machine scheduling with high multiplicity*, Mathematical Programming **89** (2001), 359–383.

[23] S.S. Cosmadakis, and C.H. Papadimitriou, *The traveling salesman problem with many visits to few cities*, SIAM Journal on Computing **13** (1984), 99–108.

[24] P. De, E.J. Dunne, J.B. Gosh, and C.E. Wells, *Complexity of the discrete time-cost tradeoff problem for project networks*, Operations Research **45** (1997), 302–306.

[25] P. De, E.J. Dunne, J.B. Gosh, and C.E. Wells, *The discrete time-cost tradeoff problem revisited*, European Journal of Operational Research **81** (1995), 225–238.

[26] V.G. Deineko, and G.J. Woeginger, *Hardness of approximation of the discrete time-cost tradeoff problem*, Operations Research Letters **29** (2001), 207–210.

[27] R.P. Dilworth, *A decomposition theorem for partially ordered sets*, Annals of Mathematics **51** (1950), 161–166.

[28] A. Drexl, and A. Kimms, *Sequencing JIT mixed-model assembly lines under station-load and part-usage constraints*, Management Science, **47** (2001), 480–491.

[29] M.E. Dyer, *The complexity of vertex enumeration methods*, Mathematics of Operations Research **8** (1983), 381–402.

[30] H.A. Eiselt, M. Gendreau, and G. Laporte, *Arc routing problems, part i: The chinese postman problem*, Operations Research **43** (1995), 231–243.

[31] W. Feller, *An introduction to the theory of probability and its applications*, John Wiley and Sons, 1967.

[32] G. Finke, *Scheduling on batching machines*, In Proceedings of the Sixth Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP 2003) (Aussois, France), 2003, p. 16.

[33] H. Frank, D.J. Kleitman, D.M. Rosenbaum, B. Rothfarb, and K. Steiglitz, *Optimal design of offshore natural-gas pipeline systems*, Operations Research **18** (1970), 992–1020.

[34] M.R. Garey, and D.S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, Freeman and Company, San Francisco, 1979.

[35] E. Gertsbakh, and I.B. Gertsbakh, *Reliability theory with applications to preventive maintenance*, Springer Verlag, Heidelberg, 2000.

[36] G. de Ghellinck, Y. Smeers, and M. Souissi, *Preventive maintenance optimization using linear programming*, In Proceedings of the 4th W. G. -7. 6 Working Conference on Optimization-Based Computer-Aided Modelling and Design (Noisy-le-Grand, France), 1996, 41.1–41.4.

[37] R.J. Giglio, R.G. Glaser, and H.M. Wagner, *Preventive maintenance scheduling by mathematical programming*, Management Science **10** (1964), 316–334.

[38] P.C. Gilmore, and R.E. Gomory, *Sequencing a one-state variable machine: A solvable case of the traveling salesman problem*, Journal of Operations Research Society of America **12** (1964), 655–679.

[39] M. Gouw, *Supply chain scheduling with bounded suppy chain solutions*, Master Thesis, Faculty of Economics and Business Administration, University of Maastricht, The Netherlands, 1998.

[40] 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 Operations Research **5** (1979), 287–326.

[41] F. Granot, and J. Skorin-Kapov, *On polynomial solvability of the high multiplicity total weighted tardiness problem*, Discrete Applied Mathematics **41** (1993), 139–146.

[42] H.J. Greenberg, *Rim sensitivity analysis from an interior solution*, World Wide Web, http://www-math.cudenver.edu/~ hgreenbe, 1997.

[43] A. Grigoriev, and J. Van de Klundert, *Throughput rate optimization in high multiplicity sequencing problem*, METEOR Research Memoranda RM/01/006, Maastricht University, Maastricht, Netherlands, 2001.

[44] A. Grigoriev, J. Van de Klundert, and F.C.R. Spieksma, *Modelling and solving periodic maintenance problem*, Working paper, Maastricht University, Maastricht, Netherlands, 2002.

[45] A. Grigoriev, and V. Strusevich, *Dedicated parallel machines scheduling with a single speeding-up resource*, In Proceedings of the Sixth Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP 2003) (Aussois, France), 2003, 131–132.

[46] A. Grigoriev, and G.J. Woeginger, *Project scheduling with irregular costs: Complexity, approximability, and algorithms*, In Proceedings of the 13th International Symposium on Algorithms and Computations (ISAAC 2002), 2002, 381–390.

[47] M. Grötschel, L. Lovasz, and A. Schrijver, *The ellipsoid method and its consequences in combinatorial optimization*, Combinatorica **1** (1981), 169–197.

[48] N.G. Hall, H. Kamoun, and C. Sriskandarajah, *Scheduling in robotic cells: Classification, two and three machine cells*, Operations Research **45** (1997), 421–439.

[49] N.G. Hall, H. Kamoun, and C. Sriskandarajah, *Scheduling in robotic cells: Heuristics and cell design*, Operations Research **47** (1999), 821–835.

[50] R.T. Harvey, and J.H. Patterson, *An implicit enumeration algorithm for the time/cost tradeoff problem in project network analysis*, Foundations of Control Engineering **4** (1979), 107–117.

[51] J. Håstad, *Clique is hard to approximate within* $n^{1-\epsilon}$, Acta Mathematica **182** (1999), 105–142.

[52] J. Håstad, *Some optimal inapproximability results*, In Proceedings of the 29th ACM Symposium on the Theory of Computing (STOC 1997), 1997, 1–10.

[53] T.J. Hindelang, and J.F. Muth, *A dynamic programming algorithm for decision CPM networks*, Operations Research **27** (1979), 225–241.

[54] K.L. Hitz, *Scheduling of flexible flow shops. ii*, Report LIDS-TH-1063, Laboratory of Information and Decision Systems, MIT, Cambridge, Massachusets, USA, 1979.

[55] K.L. Hitz, *Scheduling of flexible flow shops. ii*, Report LIDS-R-879, Laboratory of Information and Decision Systems, MIT, Cambridge, Massachusets, USA, 1980.

[56] D.S. Hochbaum (edit.), *Approximation algorithms for NP-hard problems*, PWS Publishing Company, 1997.

[57] D.S. Hochbaum, and R. Shamir, *Minimizing the number of tardy job units under release time constraints*, Discrete Applied Mathematics **28** (1990), 45–57.

[58] D.S. Hochbaum, and R. Shamir, *Strongly polynomial algorithms for the high multiplicity scheduling problem*, Operations Research **39** (1991), 648–653.

[59] D.S. Hochbaum, R. Shamir, and J.G. Shanthikumar, *A polynomial algorithm for an integer quadratic nonseparable transportation problem*, Mathematical Programming **55** (1992), 359–376.

[60] M. Holthuijsen, *Scheduling-problemen met raw-materials-constraints*, Master Thesis, Faculty of General Sciences, University of Maastricht, The Netherlands, 2003.

[61] J. Hurink, and S. Knust, *Flow shop problems with transportation times and a single robot*, Osnabrücker Schriften zur Mathematik 201, Universität Osnabrück, Osnabrück, Germany, 1998.

[62] J.R. Jackson, *Scheduling a production line to minimize maximum tardiness*, Research Report 43, Management Science Research Project, University of California, Los Angeles, USA, 1955.

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

[64] D.S. Johnson, C.H. Papadimitriou, and M. Yannakakis, *On generating all maximal independent sets*, Information Processing Letters **27** (1988), 119–123.

[65] P. Kanellakis, and C.H. Papadimitriou, *Flow-shop scheduling with limited temporary storage*, Journal of the ACM **27** (1980), 533–549.

[66] J.E. Kelley, and M.R. Walker, *Critical path planning and scheduling: An introduction*, Mauchly Associates Inc., Ambler, PA, 1959.

[67] C. Kenyon, and N. Schabanel, *The data broadcast problem with non-uniform transmission times*, Research Report 2001-43, Laboratoire de l'Informatique du Parallélisme, École Normale Supérieure de Lyon, France, 2000.

[68] C. Kenyon, N. Schabanel, and N. Young, *Polynomial-time approximation scheme for data broadcast*, In Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing (STOC 2000) (Portland, Oregon), 2000, pp. 659-666.

[69] J. Van de Klundert, *A note on the high multiplicity traveling salesman problem*, Working paper, Maastricht University, Maastricht, Netherlands, 1995.

[70] J. Van de Klundert, private communications.

[71] B.L. Kralj, and R. Petrović, *Optimal preventive maintenance scheduling of thermal generating units in power systems - A survey of problem formulations and solution methods* European Journal of Operational Research **35** (1988), 1–15.

[72] W. Kubiak, and S.P. Sethi, *A note on "level schedules for mixed-model assembly lines in just-in-time production systems"*, Management Science **37** (1991), 121–122.

[73] W. Kubiak, *A pseudo-polynomial algorithm for a two-machine no-wait job-shop scheduling problem*, European Journal of Operatinal Research **43** (1989), 267–270.

[74] W. Kubiak, and S.P. Sethi, *Optimal just-in-time schedules for flexible transfer lines*, International Journal of Flexible Manufacturing Systems **6** (1994), 137–154.

[75] W. Kubiak, S. Sethi, and C. Sriskandarajah, *An efficient algorithm for a shop problem*, Annals of Operations Research **57** (1995), 203–216.

[76] W. Kubiak, G. Steiner, and J.S. Yeomans, *Optimal level schedules for mixed-model, multi-level just-in-time assembly systems*, Annals of Operations Research **69** (1997), 241–259.

[77] W. Kubiak, and W. Timkovsky, *Total completion time minimization in two-machine job shops with unit-time operations*, European Journal of Operational Research **94** (1996), 310–320.

[78] E.L. Lawler, *A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness*, Annals of Operations Research **1** (1977), 331-342.

[79] E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, *Generating all maximal independent sets: NP-hardness and polynomial-time algorithms*, SIAM Journal on Computing **9** (1980), 558–565.

[80] H.W. Lenstra (Jr.), *Integer programming with a fixed number of variables*, Mathematics of Operations Research **8** (1983), 538–547.

[81] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker, *Complexity of machine scheduling problems*, Annals of Operations Research **1** (1977), 343–362.

[82] V. Maniezzo, and A. Mingozzi, *The project scheduling problem with irregular starting time costs*, Operations Research Letters **25** (1999), 175–182.

[83] Y. Matijasevic, and J. Robinson, *Reduction of an arbitrary Diophantine equation to one in 13 unknown*, Acta Arithmetica **27** (1975), 521–553.

[84] S.T. McCormick, M.L. Pinedo, S. Shenker, and B. Wolf, *Sequencing in an Assembly Line with Blocking to Minimize Cycle Time*, Operations Research **37** (1989), 925–935.

[85] S.T. McCormick, S.R. Smallwood, and F.C.R. Spieksma, *Polynomial algorithms for multiprocessor scheduling problems with a small number of job lengths*, Faculty of Commerce Working Paper 93-MSC-008, University of British Columbia, Vancouver, BC, Canada, 1993.

[86] R. McNaughton, *Scheduling with deadlines and loss functions*, Management Science **6** (1959), 1–12.

[87] S. Mehrotra, and R.D.C. Monteiro, *A general parametric analysis approach and its implications to sensitivity analysis in interior point methods*, Mathematical Programming **72** (1996), 65–82.

[88] M. Middendorf, and V.G. Timkovsky, *On scheduling cycle shops: classification, complexity and approximation*, Journal of Scheduling **5** (2002), 135–169

[89] J. Miltenburg, *Level schedules for mixed-model assembly lines in just-in-time production systems*, Management Science **35** (1989), 192–207.

[90] R.H. Möhring, *Computationally tractable classes of ordered sets*, Algorithms and Order (I. Rival, ed.), Kluwer Academic Publishers, 1989, 105–193.

[91] R.H. Möhring, A.S. Schulz, F. Stork, and M. Uetz, *On project scheduling with irregular starting time costs*, Operations Research Letters **28** (2001), 149–154.

[92] G.L. Nemhauser, and L.A. Wolsey, *Integer and Combinatorial Optimization*, Wiley-Interscience, Chichester, 1988.

[93] C.H. Papadimitriou, and K. Steiglitz, *Combinatorial optimization: Algorithms and complexity*, Prentice Hall, Englewood Cliffs, N. J., 1982.

[94] M. Pinedo, *Scheduling: Theory, algorithms and systems*, Prentice Hall, Englewood Cliffs, N. J., 1995.

[95] M.E. Posner, *The complexity of earliness and tardiness scheduling problems under id-encoding*, Working Paper 85-70, New York University, New York, USA, 1985.

[96] H.N. Psaraftis, *A dynamic programming approach for sequencing groups of identical jobs*, Operations Research **28** (1980), 1347–1359.

[97] D.R. Robinson, *A dynamic programming solution to cost-time tradeoff for CPM*, Management Science **22** (1975), 158–166.

[98] H. Röck, *The three-machine no-wait flow shop is NP-complete*, Journal of the Association for Computing Machinery **31** (1981), 336–345.

[99] M. Rothkopf, *The travelling salesman problem: On the reduction of certain large problems to smaller ones*, Operations Research **14** (1966), 532–533.

[100] N. Schabanel, *The databroadcast problem with preemption*, In Proceedings of the 17th International Symposium on Theoritical Aspects of Computer Science (STACS 2000) (Portland, Oregon), 2000, pp. 181–192.

[101] A. Schrijver, *Theory of linear and integer programming*, Wiley-Interscience, Chichester, 1986.

[102] M. Shahidehpour, and M. Marwali, *Maintenance scheduling in structured power systems*, Kluwer Academic Publishers, Norwell, Massachusets, 2000.

[103] M. Skutella, *Approximation algorithms for the discrete time-cost tradeoff problem*, Mathematics of Operations Research **23** (1998), 909–929.

[104] F. Spieksma, private communications.

[105] G. Steiner, and J.S. Yeomans, *Level schedules for mixed-model, just-in-time processes*, Management Science **39** (1993), 728–735.

[106] V.V. Vazirani, *Approximation algorithms*, Springer-Verlag, Berlin-Heidelberg, 2001.

[107] J.A.A. Van der Veen, and S. Zhang, *Low-complexity algorithms for sequencing jobs with a fixed number of job-classes*, Computers And Operations Research **23** (1996), 1059–1067.

[108] A.C.W.M. Vrouwenvelder (edit.), *Inspection and maintenance strategies*, HERON, **39** No. 2 (1994).

[109] D.A. Wismer, *Solution of the flow shop scheduling problem with no intermediate queues*, Operations Research **20** (1972), 689–697.

[110] G.J. Woeginger, *When does a dynamic programming formulation guarantee the existence of an FPTAS?*, Electronic Colloquium on Computational Complexity (ECCC) **8** (2001).

Samenvatting (Summary in Dutch)

Dit proefschrift heeft als onderwerp high multiplicity scheduling problemen, oftewel scheduling problemen waarin veelvuldigheid een rol speelt. In een traditioneel scheduling probleem, bestaat de probleembeschrijving uit een verzameling van taken en een verzameling van middelen. Het probleem bestaat er vervolgens uit om toewijzing in de tijd te vinden van de middelen aan de taken, op zodanige wijze dat een bepaalde doelstellingsfunctie wordt geoptimaliseerd. In een high multiplicity scheduling probleem, bestaat de probleembeschrijving niet uit individuele taken, maar uit types van taken, en wordt van ieder type omschreven hoeveel taken van het betreffende type er zijn. Hierdoor is de probleembeschrijving significant compacter dan wanneer ieder van de taken apart wordt omschreven. Dit proefschrift bestudeert efficientie beschrijvingen van verscheidene aan de praktijk gerelateerde high multiplicity scheduling problemen en hun oplossingen, alsmede de mate waarin het mogelijk is ze op efficiënte wijze op te lossen.

Vanuit theoretisch oogpunt zijn er verscheidene onderzoeksvragen die gesteld kunnen worden, waaronder de volgende drie:

1. Hoe verandert de complexiteit van een probleem wanneer we van enkelvoudigheid naar meervoudigheid gaan?

2. Kunnen algoritmes die voor enkelvoudige versies zijn ontwikkeld ook worden toegepast in high multiplicity problemen?

3. Hoe dient de oplossing van een high multiplicity probleem te worden gerepresenteerd?

Dit proefschrift beantwoordt deze vragen voor een aantal bekende combinatorische problemen.

Hoofdstuk 1 geeft een formele introductie in high multiplicity scheduling. Bovendien biedt het een gedetailleerde beschrijving van de onderzoeksvragen die in het kader van high multiplicity relevant zijn. Hoofdstuk 2 bevat een complexiteitsraamwerk voor de analyse van high multiplicity scheduling problemen. Doel van dit raamwerk is om output gerichte complexiteitsmaatstaven te definiëren voor algoritmes die een bepaalde natuurlijke beschrijving van de oplossing geven. Daarbij wordt onderscheid gemaakt naar algoritmes die de oplossing weergeven als een verzameling van intervallen en algoritmes die de oplossing weergeven als een projectie.

In Hoofdstuk 3 beschouwen we het bekende handelsreizigersprobleem, meer specifiek het high multiplicity traveling salesman problem. Dit probleem is op verscheidene manieren gerelateerd aan scheduling problemen. Een taak correspondeert in dit geval met het bezoeken van een stad. Hoofdstuk 3 onderzoekt hoe de structuur en de waarde van de optimale oplossing samenhangten met de veelvuldigheid, bijvoorbeeld door deze te parametriseren. Bovendien laten we zien hoe optimale oplossingen voor problemen met hoger veelvuldigheden kunnen worden verkregen op basis van oplossingen voor lagere veelvuldigheden.

Hoofdstuk 4 beschouwt een onderhoudsplanningsprobleem, het "periodic maintenance problem". In dergelijke problemen bestaat een aantal machines, en een taak correspondeert met het verlenen van onderhoud aan een machine. Doelstelling is om de onderhoudsbeurten zo in te plannen dat de som van de onderhouds- en productiekosten zo laag mogelijk is. In dit hoofdstuk richten we ons vooral op het berekenen van optimale oplossingen voor dit probleem. Daartoe onderzoeken we diverse mathematische formuleringen waaronder geheeltallige lineaire programmeringsformuleringen met een goed LP relaxatie. Op basis van deze formuleringen lossen we grote probleeminstanties op met behulp van branch-and-price technieken. Dit resulteert in de eerste exacte oplossingsmethoden voor periodic maintenance problemen, en we geven dan ook uitgebreide reken resultaten.

Hoofdstuk 5 beschouwt een variëteit aan supply chain gerelateerde scheduling problemen en hun high multiplicity aspecten. We gebruiken het in hoofdstuk 2 ontwikkelde instrumentarium om hun complexiteit te onderzoeken en om geschikte beschrijvingen voor de oplossingen van de problemen te definiëren. We laten bovendien zien hoe een aantal klassieke algoritmes voor traditionele scheduling problemen kan worden toegepast op high multiplicity problemen.

Hoofdstuk 6 beschouwt een speciale klasse van project scheduling proble-

men, namelijk project scheduling problems with irregular costs. We richten ons daarbij op een generalisatie van het klassieke discrete time-cost trade off problem, waarin de kosten onregelmatig verlopen, en afhankelijk zijn van de begin- en eindtijden van de taken. We geven een compleet overzicht over de complexiteit van het probleem en van haar approximeerbaarheid voor verscheidene natuurlijke klassen van precedentierelaties, en voor varianten waarin veelvuldigheid een rol speelt.

About the Author

Alexander Grigoriev was born on 26 May, 1971 in Ufa, Russia. In 1988 he graduated from the Specialized Physical and Mathematical School at Novosibirsk State University, Russia. From 1988 till 1994 he has studied mathematics/applied mathematics/operations research at the Novosibirsk State University. From 1991 till 1999 Alexander Grigoriev has worked at Sobolev Institute of Mathematics, Novosibirsk, Russia.

From October 1999 till October 2003, Alexander Grigoriev was a PhD-Candidate in the Operations Research Group at the Department of Quantitative Economics, Faculty of Economics and Business Administration, University of Maastricht. His research in combinatorial optimization was supervised by prof. dr. ir. Antoon W.J. Kolen, prof. dr. Yves Crama, and dr. Joris van de Klundert.