Approximation Schemes for Scheduling Problems

Dissertation

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr.-Ing.)
der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel

Dipl. Ing. Palmo Monaldo Mastrolilli

Kiel

2002

1. Gutachter: Prof. Dr. Klaus Jansen

2. Gutachter: Prof. Dr. Roberto Solis-Oba, London, Canada

3. Gutachter: Prof. Dr. Anand Srivastav

Tag der mündlichen Prüfung: 12.06.2002

Abstract

Most combinatorial optimization problems of great practical relevance are computationally intractable. In formal terms, they are classified as NP-hard optimization problems. The need to cope with these problems is a reason that forces us to relax the notion of optimality and resort to suboptimal approaches. An approximation scheme is a suboptimal algorithm that computes in polynomial time a solution that, though not being an optimal one, nevertheless can be made arbitrarily close to the optimum, with better approximations yielding longer running times.

This thesis describes new and improved approximation schemes for several NP-hard machine scheduling and related problems. A typical problem in machine scheduling consists of n jobs that are to be executed on a set of available machines subject to a variety of constraints. We consider a broad class of machine models, including single, identical and unrelated parallel machines. An interesting objective is minimizing *makespan*, that is the time to complete all jobs. Constraints that we study include the following. Jobs are released over time and must not be started before their release dates. Precedence constraints impose restrictions on the sequencing of jobs. In preemptive schedules a job may repeatedly be interrupted and continued later on another (or the same) machine; in nonpreemptive schedules a job must be processed in an uninterrupted fashion. Classical scheduling models assume that jobs have *fixed* processing times. This thesis includes problems where the processing times of the jobs can be reduced at some cost. The interdependence of cost and processing time accounts for the trade-off of an early completion time versus low cost.

We introduce novel ideas that improve earlier approaches and result in simple and efficient algorithms. For many scheduling problems our algorithms return approximate solutions of any fixed error $\varepsilon > 0$ in $O(n) + C$ time, where the constant hidden in $O(n)$ is reasonably small and independent of the error ε, whereas the additive constant C depends on ε and some other fixed parameters. We can prove that algorithms with time complexity

of this form are the best possible with respect to the number of jobs. Moreover and surprisingly, we show that for every fixed error $\varepsilon > 0$ it is possible to get rid of constant C, although this results in a much longer algorithm.

Concerning the methods we used, a basic approach adopted throughout this thesis is to transform any given instance x into a smaller instance x_ε of the same problem. This transformation depends on the desired precision ε of approximation; the closer ε is to zero, the closer x_ε should resemble x. Instance x_ε is computed by first structuring the input and then "merging" jobs that have similar characteristics. We show that an approximate solution for instance x can be obtained by computing an optimal (or approximate) solution for x_ε. By using this basic idea, we show that several previous approaches can be improved and enormously simplified. We remark that these transformations are themselves of interest. Indeed, they can be used as preprocessing steps in other algorithms (like branch and bound and cutting plane algorithms) to obtain faster approximate solutions. Other basic tools used in this thesis include linear programming and rounding. Rounding the input is a widely used technique to obtain polynomial time approximation schemes. One of our contributions is to show that the appropriate combinations of arithmetic and geometric rounding techniques yield novel and powerful rounding methods.

Acknowledgments

I wish to acknowledge my great debt with IDSIA director, Luca Maria Gambardella, not only for introducing me to the field and community of combinatorial optimization but also for his on-going encouragement and support, for his valuable comments and advice, and for all the freedom.

I am particularly indebted to my supervisor, Klaus Jansen, for drawing my interest on approximation algorithms, for teaching me about doing research and for supervising this thesis.

I am especially grateful to Roberto Solis-Oba for being coauthor, reading this thesis and suggesting several corrections.

IDSIA is truly a first class institute and I must thank all the people who have helped me directly or indirectly. My sincere thanks to Zaffa for the enjoyable hours spent running.

I dedicate this thesis to my wife Elena for the many happy times we have shared.

Monaldo Mastrolilli
Manno, Switzerland
December, 2001

I gratefully acknowledge funding support from Swiss National Science Foundation grants 21-55778.98 and 20-63733.00/1 to Luca Maria Gambardella.

Contents

Chapter 1

Introduction

The advent of computers initiated the formal study of efficiency of computation that led to the notion of NP–completeness. Karp's seminal work [47] established the nature of NP–completeness by showing that decision versions of several naturally occurring problems in combinatorial optimization are NP–complete, and thus are unlikely to have efficient (polynomial time) exact algorithms. Following Karp's work, many problems, including scheduling problems, were shown to be NP–complete. It is widely believed that P (the set of languages that can be recognized by Turing machines in deterministic polynomial time) is a proper subset of NP (the set of languages that can be recognized by Turing machines in non–deterministic polynomial time). Proving that P$\neq$NP is the most outstanding problem in theoretical computer science today.

The practical importance of NP–complete optimization problems necessitates tractable relaxations. By tractable we mean efficient solvability, and polynomial time is a robust theoretical notion of efficiency. A very fruitful approach has been to relax the notion of optimality and settle for *near–optimal* solutions. A near–optimal solution is one whose objective function value is within some small multiplicative "1" factor of the optimal value. Approximation algorithms are heuristics that provide provably good guarantees on the quality of the solutions they return. This approach was pioneered by the influential paper of Johnson [46] in which he showed the existence of good approximation algorithms for several NP–complete optimization problems. He also remarked that the optimization problems that are all indistinguishable in the theory of NP–completeness behave very differently when it comes to approximability. Remarkable work in the last couple of decades in both the design of approximation algorithms and prov-

ing inapproximability results has validated Johnson's remarks. The book on approximation algorithms edited by Hochbaum [32] gives a good glimpse of the current knowledge on the subject. The methodology of evaluating algorithms by the quality of their solutions is useful in comparing commonly used heuristics, and often the analysis suggests new and improved heuristics.

Sequencing and scheduling problems are motivated by applications in which it is necessary to allocate limited resources over time. The goal is to find an optimal allocation where optimality is defined by some problem dependent objective. Given the broad definition above it is no surprise that scheduling is a vast sub–area of optimization. The work in this thesis falls under the category of deterministic machine scheduling. Resources are modeled by machines and activities are modeled by jobs that can be executed by the machines. Deterministic refers to the scenario in which all parameters of the problem instance are known in advance. Problems in deterministic machine scheduling are combinatorial optimization problems. Consequently, techniques from the rich field of combinatorial optimization find motivation in, and are extensively applied to scheduling problems.

Most of the early work on scheduling, starting from early 1950's, was motivated by production planning and manufacturing, and was primarily done in the Operations Research and Management Science community. The advent of computers and their widespread use had a considerable impact both on scheduling problems and solution strategies. Computers and related devices are, in a very concrete sense, resources that need to be allocated to various processes. Many problems in scheduling found new applications in Computer Science. In addition, a number of new problems have been motivated by application areas in Computer Science such as parallel computing, databases, compilers, and time sharing.

Approximation algorithms for several problems in scheduling have been developed in the last three decades. In fact it is widely believed that the first NP optimization problem for which an approximation algorithm was formally designed and analyzed is the multiprocessor scheduling problem, this algorithm was designed by Graham in 1966 [26]. This precedes the development of the theory of NP–completeness.

An approximation scheme for an NP-complete optimization problem in minimization (maximization) form is a polynomial time algorithm which, given any instance of the problem and any given value $\varepsilon > 0$ ($0 < \varepsilon < 1$), it returns a solution whose value is within factor $(1 + \varepsilon)$ $((1 - \varepsilon))$ of the optimal solution value for that instance. The main themes of this thesis revolve around the design of new and improved approximation schemes for several scheduling problems. Our coverage of scheduling results proceeds

in somewhat "standard" fashion in the sense of topical progression: we start with single machine results, then we present material on identical and unrelated parallel machines, and finally we consider job shop-like models.

1.1 Optimization and Approximation Algorithms

In this section we give some standard definitions in the field of optimization and approximation theory (see e.g. [5]). We assume the reader to be familiar with the basic concepts of computational complexity theory (see e.g. [19]). For a string x, we denote by $|x|$ the length of its binary encoding.

Definition 1.1 *(NPO problems) An NP optimization problem $\mathcal{P}$ is a four-tuple $(I, sol, m, goal)$ such that*

1. *I is the set of instances of $\mathcal{P}$, and I is recognizable in polynomial time.*
2. *Given an instance x of I, $sol(x)$ denotes the set of feasible solutions of x. These solutions are short, that is, a polynomial p exists such that, for any $y \in sol(x)$, $|y| \leq p(|x|)$. Moreover, for any x and for any y with $|y| \leq p(|x|)$, it is decidable in polynomial time whether $y \in sol(x)$.*
3. *Given an instance x and a feasible solution y of x, $m(x, y)$ denotes the positive rational measure of y (often also called the value of y). The function m is computable in polynomial time and is also called the objective function.*
4. *$goal \in \{max, min\}$.*

Informally, feasible solutions correspond to those solutions that satisfy the constraints of the given problem. The goal of an NP optimization problem with respect to an instance x is to find an optimum solution, that is, a feasible solution y such that $m(x, y) = goal\{m(x, y') : y' \in sol(x)\}$. In the following $OPT(x)$ will denote the optimal solution value for $x \in I$. For an instance $x \in I$, we also denote by $sol^*(x)$ the set of optimum solutions for x. The *class NPO* is the set of all NP optimization problems.

Given an NP optimization problem we can always derive from it an *underlying language* (or *underlying decision problem*) in the following way. Instead of looking for the best solution of instance x, we ask whether x admits a solution having measure, at least, k (or, at most, k) and, in the affirmative case we include the pair (x, k) in the underlying language. Thus, that language consists of all pairs (x, k), with $x \in I$ and $k > 0$, such that a

feasible solution y exists with $m(x, y) \geq k$ if $goal = \max$ and $m(x, y) \leq k$ otherwise.

Even if not explicitly introduced, underlying the definition of NPO problems there is a nondeterministic computation model. This is formally stated by the following result which basically shows that the class NPO is the natural optimization counterpart of the class NP.

Theorem 1.1 *[5] For any optimization problem in NPO, the corresponding decision problem belongs to NP.*

The relationship between classes NP and NPO, which holds in the case of nondeterministic computations, can be translated, in the case of deterministic algorithms, by the following definition, that introduces the class of NPO problems which are efficiently solvable.

Definition 1.2 *(Class PO) An optimization problem $\mathcal{P}$ belongs to the* class PO *if it is in NPO and there exists a polynomial time computable algorithm A that, for any instance $x \in I$ of $\mathcal{P}$, returns an optimal solution $y \in sol^*(x)$, together with its value $OPT(x)$.*

For all the optimization problems in NPO $\setminus$ PO the intrinsic complexity is not precisely known. Just as with the decision problems in NP $\setminus$ P, no polynomial-time algorithms for them have been found but no proof of intractability is known either. In fact, the question "PO = NPO?" is strictly related to the question "P = NP?" since it can be proved that the two questions are equivalent in the sense that a positive answer to the first would imply a positive answer to the second and vice versa. In the following we say that a problem $\mathcal{P} \in NPO$ is NP-complete if the underlying language of $\mathcal{P}$ is NP-complete.

From the preceding result we may easily derive a first consequence concerning the complexity of NPO problems. In fact it turns out that if we could solve an NP-complete optimization problem in polynomial time we could also solve its underlying decision problem in polynomial time. Hence, unless P = NP, no problem in NPO whose underlying language is NP-complete can belong to PO.

1.1.1 Approximate Solutions

In the previous subsection we have seen that, due to their inherent complexity, NP-complete optimization problems cannot be efficiently solved in an exact way, unless P=NP. Therefore, if we want to solve an NP-complete

optimization problem by means of an efficient (polynomial-time) algorithm, we have to accept the fact that the algorithm does not always return an optimal solution but rather an approximate one. In order to express the quality of an approximate solution the next definition introduces the notion of performance ratio.

Definition 1.3 *(Performance ratio) Let $\mathcal{P}$ be an NPO problem. For any instance x and for any solution $y \in sol(x)$, the* performance ratio *of y with respect to x is defined as*

$$R(x,y) = \max\left\{\frac{OPT(x)}{m(x,y)}, \frac{m(x,y)}{OPT(x)}\right\}.$$

Observe that the performance ratio is always a number greater than or equal to 1 and is as close to 1 as the solution is close to an optimum solution. It is interesting to consider situations in which such a quality measure is bounded by a constant for all input instances.

Definition 1.4 *(r-approximate algorithm) Let $\mathcal{P}$ be an NPO problem and let A be an algorithm that, for any instance x of $\mathcal{P}$ such that $sol(x) \neq \emptyset$, returns a feasible solution $A(x)$ in polynomial time. Given a fixed constant $r \geq 1$, we say that A is an* r-approximate *algorithm for $\mathcal{P}$ if, for any instance x of $\mathcal{P}$, the performance ratio of the feasible solution $A(x)$ with respect to x verifies the following inequality*

$$R(x, A(x)) \leq r.$$

Within the class NPO, the class of problems that allow polynomial time r-approximate algorithms plays a very important role. In fact, the existence of a polynomial time r-approximate algorithm for an NP-complete optimization problem shows that, despite the inherent complexity of finding the optimal solution of the problem, such a solution can somehow be approached. The complexity class APX consists of all NPO problems $\mathcal{P}$ such that, for some $r \geq 1$, there exists a polynomial-time r-approximate algorithm for $\mathcal{P}$.

Let us now define the class of those NPO problems for which we can obtain an arbitrarily good approximate solution in polynomial time with respect to the size of the problem instance.

Definition 1.5 *(PTAS) Let $\mathcal{P}$ be an NPO problem. An algorithm A is said to be a polynomial-time approximation scheme (PTAS) for $\mathcal{P}$ if, for any instance x of $\mathcal{P}$ and any rational number $r > 1$, A when applied to input (x, r) returns an r-approximate solution of x in time polynomial in $|x|$.*

While always being polynomial in $|x|$, the running time of a PTAS may also depend arbitrarily on $1/(r-1)$ with better approximations yielding longer running times. In some cases we may construct approximation schemes with running times that are polynomial both in the size of the instance and in $1/(r-1)$. In such cases the possibility of approaching the optimal solution with arbitrarily small error is more concrete. Fully polynomial time approximation schemes (FPTAS) are polynomial time approximation schemes having running time growing polynomially with the reciprocal of the error bound.

Definition 1.6 *(FPTAS) A PTAS is said to be a fully polynomial-time approximation scheme (FPTAS) if its running time is also polynomial in* $1/(r-1)$.

Let us use PTAS (FPTAS) to denote also the class of NPO problems that admit a (fully) polynomial time approximation scheme. With respect to worst case approximation, an FPTAS is the strongest possible result that we can derive for an NP-complete problem. Some problems are known not to have fully polynomial time approximation schemes unless P = NP. This lower bound applies to strongly NP-complete problems that fulfill some weak and natural supplementary conditions (Garey and Johnson [22]):

Theorem 1.2 *Let* $\mathcal{P} = (I, sol, m, goal)$ *is a strongly NP-complete problem that admits a polynomial* p *such that* $OPT(x) \leq p(|x|, \max(x))$, *for every input* $x \in I$, *whereas* $\max(x)$ *denotes the value of the largest number occurring in* x. *If* $P \neq NP$, *then* $\mathcal{P}$ *does not belong to the class FPTAS.*

By the previous arguments, the following inclusions hold:

$$PO \subseteq FPTAS \subseteq PTAS \subseteq APX \subseteq NPO$$

These inclusions are strict if and only if P $\neq$ NP (see e.g. [5]).

Moreover, following Cesati and Trevisan [11], we introduce the class of NPO problems that admit an efficient PTAS.

Definition 1.7 *(EPTAS) A PTAS is said to be an efficient polynomial-time approximation scheme (EPTAS) if its running time is bounded by* $f(r)|x|^c$, *where* f *is an arbitrary function and* c *is a fixed constant. We call EPTAS the class of NPO problems admitting an EPTAS.*

Under likely assumptions, Cesati and Trevisan [11] prove that for some problems (including natural ones) there cannot be efficient approximation schemes. The results reported in [11] are related to Parameterized Complexity Theory, and state that the existence of such an algorithm would imply that the classes $W[P]$ and FPT are equal (see [16]).

Remark 1.1 *Sometimes, it is more convenient to work with a measure of relative error. For any input, the* relative error *of the approximation algorithm is defined to be* $|A(x) - OPT(x)|/OPT(x)$. *The relative error is always nonnegative. An approximation algorithm has a* relative error bound *of* ε *if for every instance* x

$$\frac{|A(x) - OPT(x)|}{OPT(x)} \leq \varepsilon. \tag{1.1}$$

It follows that for a minimization problem we have

$$r = 1 + \varepsilon,$$

whereas for a maximization problem

$$r = 1 - \varepsilon.$$

Note that for minimization problems the inequality (1.1) becomes $A(x) \leq (1+\varepsilon)OPT(x)$, *whereas for maximization problems it becomes* $A(x) \geq (1-\varepsilon)OPT(x)$. *Moreover, for minimization problems the worst case ratio* $r = 1 + \varepsilon$ *is greater or equal to* 1, *whereas for maximization problems the worst case ratio* $r = 1 - \varepsilon$ *is a real number from the interval* $[0, 1]$.

1.1.2 A Bit of History

The first paper with a polynomial time approximation algorithm for an NP-complete problem is probably the paper [26] by Graham from 1966. It studies simple heuristics for scheduling on identical parallel machines. In 1969, Graham [27] extended his approach to a PTAS. However, at that time these were isolated results. The concept of an approximation algorithm was formalized in the beginning of the 1970s by Garey, Graham and Ullman [20]. The paper [46] by Johnson may be regarded as the real starting point of the field.

In the mid-1970s, a number of polynomial time approximation schemes was developed in the work of Horowitz and Sahni [35, 36], Sahni [71], and Ibarra and Kim [37]. The terms 'approximation scheme', 'PTAS', 'FPTAS'

are due to a seminal paper by Garey and Johnson [22] from 1978. Also the first inapproximability results were derived around this time; inapproximability results are results that show that unless P=NP some optimization problem does not have a PTAS or that some optimization problem does not have a polynomial time r-approximation algorithm for some specific value of r. Sahni and Gonzalez [72] proved that the traveling salesman problem without the triangle-inequality cannot have a polynomial time approximation algorithm with bounded worst case ratio. Garey and Johnson [21] derived inapproximability results for the chromatic number of a graph. Lenstra and Rinnooy Kan [53] derived inapproximability results for scheduling of precedence constrained jobs.

In the 1980s theoretical computer scientists started a systematic theoretical study of these concepts. Major breakthroughs that give polynomial time approximation schemes for specific optimization problems were the papers by Fernandez de la Vega and Lueker [15] on bin packing, by Hochbaum and Shmoys [33, 34] on the scheduling problem of minimizing the makespan on an arbitrary number of parallel machines, by Baker [6] on many optimization problems on planar graphs (like maximum independent set, minimum vertex cover, minimum dominating set), and by Arora [3] on the Euclidean traveling salesman problem. In the late 1980s, Papadimitriou and Yannakakis [69] provided tools and ideas from computational complexity theory for getting inapproximability results. The complexity class APX was born; this class contains all optimization problems that possess a polynomial time approximation algorithm with a finite, positive worst case ratio. In 1992 Arora, Lund, Motwani, Sudan and Szegedy [4] showed that the hardest problems in APX cannot have a PTAS unless P=NP.

1.2 Scheduling Models

Scheduling theory encompasses a large and diverse set of models, algorithms, and results. Even a succinct overview of the field would take many pages. Hence we review only those concepts that are directly relevant to this thesis and refer the reader to the many excellent books and surveys available [8, 52, 70]. The problems we consider involve scheduling or allocating jobs to machines under various constraints. Unless otherwise mentioned n denotes the number of jobs and m denotes the number of machines. A schedule is non–preemptive if each job runs uninterrupted on one machine from start to finish. In a preemptive schedule, a job may be interrupted or may switch machines at any time. A schedule specifies for each time instant, the set of

jobs executing at that instant, and the machines on which they are executing. In this thesis we focus mostly on non–preemptive schedules. However preemptive schedules play an important role, as relaxations, that can be used in the design and analysis of algorithms for non–preemptive schedules. Jobs can have several constraints on them; in our work we consider *release dates*, *delivery times* and *precedence constraints*. If job J_j has a release date r_j, then it cannot be started before time r_j. Precedence constraints, specified in the form of a directed acyclic graph, model dependencies between jobs. After completing its processing on the machine, a job may require an additional delivery time $q_j \geq 0$; if C_j denotes the time J_j completes processing, then it is delivered at time $C_j + q_j$. Delivery is a *non-bottleneck* activity, in that all jobs may be simultaneously delivered. If job J_j precedes job J_i, denoted by $J_j \prec J_i$, then i cannot be started until J_j has been finished. The two objective functions we will be concerned with are *makespan* and *maximum delivery time*. Makespan, denoted by $C_{\max}$, is simply the maximum over all jobs of the completion time of the job ($C_{\max} = \max_j C_j$). Maximum delivery time as the name suggests is simply $L_{\max} = \max_j L_j$, where $L_j = C_j + q_j$. We consider both single machine scheduling and parallel machine scheduling.

Most classical scheduling models assume that in a scheduling problem the jobs to be scheduled have *fixed* processing times. However, in real-life applications the processing time of a job often depends on the amount of resources such as facilities, manpower, funds, etc. allocated to it, and so its processing time can be reduced when additional resources are assigned to the job. This accelerated processing of a job comes at a certain cost, though. A scheduling problem in which the processing times of the jobs can be reduced at some expense is called a scheduling problem with *controllable* processing times.

Scheduling problems with controllable processing times have gained importance in scheduling research since the early works of Vickson [79, 80]. For a survey of this area until 1990, the reader is referred to [67]. More recent results include [12, 13, 65, 76, 77].

In this thesis we consider both models, with fixed and controllable processing times. Solving a scheduling problem with controllable processing times amounts to specifying a schedule σ that indicates the starting times for the operations and a vector δ that gives their processing times and costs. We denote by $T(\sigma, \delta)$ the makespan (or the maximum delivery time) of schedule σ with processing times according to δ, and we denote by $C(\delta)$ the total cost of δ. The problem of scheduling jobs with controllable processing times is a bicriteria optimization problem for which we can define the

following three natural optimization problems.

P1. Minimize $T(\sigma, \delta)$, subject to $C(\delta) \leq \kappa$, for some given value $\kappa \geq 0$.

P2. Minimize $C(\delta)$, while ensuring that $T(\sigma, \delta) \leq \tau$, for some given value $\tau \geq 0$.

P3. Minimize $T(\sigma, \delta) + \alpha C(\delta)$, for some given value $\alpha > 0$.

Throughout this thesis, we will adhere to what has become the adopted convention for problem classification in the scheduling literature. Following Graham et al. [25], we employ a three-field classification $\alpha|\beta|\gamma$ where the first field corresponds to the machine or processor setting, the second to job characteristics, and the third to the optimality criterion of interest.

1.3 Organization and Overview of Contributions

We next describe the problems considered in this thesis and our results in more details.

CHAPTER 2: In this chapter we consider the following single machine scheduling problem. A set of n jobs is to be processed without interruption on a single machine. Each job J_j has a release date $r_j \geq 0$ when it first becomes available for processing and, after completing its processing on the machine, requires an additional delivery time $q_j \geq 0$. Feasible schedules are further restricted by job precedence constraints given by the partial order $\prec$, where $J_j \prec J_k$ means that job J_k must be processed after job J_j. We consider the problem with fixed processing times and problem P3 with controllable processing times.

When the processing times are fixed and there are no precedence constraints (problem denoted as $1|r_j|L_{\max}$ in [25]), Hall and Shmoys [29, 31] propose two polynomial time approximation schemes with running times $O((\frac{n}{\varepsilon})^{O(1/\varepsilon)})$ and $O(n \log n + n(1/\varepsilon^{O(1/\varepsilon^2)}))$. For the corresponding problem P3 with controllable processing times, Zdrzalka [82] gives a polynomial time approximation algorithm with a worst-case ratio of $3/2 + \varepsilon$, where $\varepsilon > 0$ can be made arbitrarily small. When the precedence constraints are imposed and the job processing times are fixed ($1|r_j, prec|L_{\max}$), Hall and Shmoys [30] give a PTAS. This consists of executing, for $\log_2 \Delta$ times, an extended version of their previous PTAS for $1|r_j|L_{\max}$, where Δ denotes an upper bound on the optimal value of any given instance whose data are assumed to be integral.

Our contribution is twofold. First, we show that for problem $1|r_j, prec|L_{\max}$ with fixed processing times there exists a PTAS running in $O(n + \ell + 1/\varepsilon^{O(1/\varepsilon)})$ time, where ℓ denotes the number of precedence constraints. Note that the time complexity of this PTAS is the best possible with respect to the input size and it is an improvement of previous results [29, 30, 31] with respect to both, the input size and $1/\varepsilon$. We remark that the constant that depends on $1/\varepsilon$ is now additive. Moreover the existence of a PTAS whose running time is also polynomial in $1/\varepsilon$ would imply that P=NP [22]. Second we provide the first known PTAS for problem $1|r_j, prec|L_{\max}$ with controllable processing that runs in linear time. This improves and generalizes all the previous results [29, 30, 31, 82].

CHAPTER 3: In this chapter, we discuss scheduling problems with identical parallel machines. The identical parallel machine scheduling model is defined as follows. We have a set of n jobs and m identical machines. Each job must be processed without preemption on one of the m machines, each of which can process at most one job at a time. When the processing times are fixed, Hall and Shmoys give a PTAS for the problem with release dates and delivery times [29] whose running time is $O(n^{\text{poly}(1/\varepsilon)})$.

We consider both, the problem with fixed processing times, and the problem with controllable processing times. We contribute in two ways. First we show that the problem with fixed processing times, release dates and delivery times belongs to the class EPTAS: we prove this by exhibiting a PTAS that runs in $O(n) + C$, where C is a constant that depends on the error ε and the multiplicative constant hidden in the $O(n)$ running time of our algorithm is reasonably small and does not depend on the accuracy ε. This considerably improves the time complexity of the previous result [29]. Second, we address several variants of the problem with controllable processing times and provide the first known polynomial time approximation schemes for them. Finally we study the problem when preemptions are allowed and describe efficient exact and approximate algorithms.

CHAPTER 4: In this chapter we deal with the problem of scheduling a set of n independent jobs on a set of m unrelated parallel machines. Each machine can process at most one job at a time, and each job has to be processed without interruption by exactly one machine. Processing job J_j on machine i requires $p_{ij} \geq 0$ time units when processed by machine i and incurs a cost $c_{ij} \geq 0$, $i = 1, \ldots, m$, $j = 1, ..., n$. We consider the problem of minimizing the objective function that is a weighted sum of the makespan

and total cost.

When m is a constant (the restricted case we are focusing on this chapter) and $c_{ij} = 0$, Horowitz and Sahni [36] proved that for any $\varepsilon > 0$, a $(1+\varepsilon)$-approximate solution can be computed in $O(nm(nm/\varepsilon)^{m-1})$ time, which is polynomial in both n and $1/\varepsilon$ if m is constant. Lenstra, Shmoys and Tardos [54] also gave an approximation scheme for the problem with running time bounded by the product of $(n+1)^{m/\epsilon}$ and a polynomial of the input size. Even though for fixed m their algorithm is not fully polynomial, it has a much smaller space complexity than the one in [36]. Recently, Jansen and Porkolab [42] presented a fully polynomial-time approximation scheme for the problem whose running time is $n(m/\varepsilon)^{O(m)}$. They combine the previous (dynamic [36] and linear [54] programming) approaches. Their algorithm has to solve at least $O((m^3/\varepsilon^2)^m)$ many linear programs. In order to obtain a linear running time for the case when m is fixed, they use the price-directive decomposition method proposed by Grigoriadis and Khachiyan [28] for computing approximate solutions of block structured convex programs. The final ingredient is an intricate rounding technique based on the solution of a linear program and a partition of the job set.

In contrast to the previous approach [42], the algorithm proposed in this chapter (that works also for the general variant with $c_{ij} \neq 0$) is extremely simple and even faster: first we preprocess the data to obtain a smaller instance with at most $\min\{n, (\log m/\varepsilon)^{O(m)}\}$ jobs. The preprocessing step requires linear time. Then by using dynamic programming we compute an approximate solution for the grouped jobs in $(\log m/\varepsilon)^{O(m^2)}$ time. Both steps together imply a fully polynomial-time approximation scheme that runs in $O(n) + C$ time where $C = (\log m/\varepsilon)^{O(m^2)}$. We remark that the multiplicative constant hidden in the $O(n)$ running time of our algorithm is reasonably small and does not depend on the accuracy ε.

CHAPTER 5: In this chapter, we discuss the job shop scheduling problem. We consider both, the problem with fixed processing times, and the problem with controllable processing times. In the job shop scheduling problem with fixed processing times, there is a set of n jobs that must be processed on a given set M of m machines. Each job J_j consists of a sequence of μ operations $O_{1j}, O_{2j}, \ldots, O_{\mu j}$ that need to be processed in this order. Operation O_{ij} must be processed without interruption on machine $m_{ij} \in M$, during p_{ij} time units. Each machine can process at most one operation at a time, and each job may be processed by a most one machine at any time. For any given schedule, let C_{ij} be the completion time of operation O_{ij}. The

objective is to find a schedule that minimizes the makespan.

For those instances where m and μ are fixed (the restricted case we are focusing on in this chapter), Shmoys et al. [75] gave approximation algorithms that compute $(2+\varepsilon)$-approximate solutions in polynomial time for any fixed $\varepsilon > 0$. This result has recently been improved by Jansen, Solis-Oba and Sviridenko [45] who have shown that $(1+\varepsilon)$-approximate solutions of the problem can be computed in polynomial time. In order to speed up the whole algorithm, they suggest [44] a number of improvements: they use the logarithmic potential price decomposition method of Grigoriadis and Khachiyan [28] to compute an approximate solution of the linear program in linear time, and a novel rounding procedure to bring down to a constant the number of fractional assignments in any solution of the linear program. The overall running time is $O(n)$, where the multiplicative constant hidden in the $O(n)$ running time is exponentially in m, μ and ε.

In contrast to the approximation schemes introduced in [45, 44], our algorithm, in addition to being more efficient, is simple and intuitive. We show that we can preprocess in linear time the input to obtain a new instance with a constant number of grouped jobs. This immediately gives a linear time approximation scheme with running time $O(n) + C$, where C is a constant that depends on m, ε and μ. Again, we remark that the multiplicative constant hidden in the $O(n)$ running time of our algorithm is reasonably small and does not depend on the accuracy ε.

In the second part of this chapter we consider two variants of the job shop problem with controllable processing times. The first variant allows *continuous* changes to the processing times of the operations. The second assumes only *discrete* changes. In the case of continuously controllable processing times, we assume that the cost of reducing the time needed to process an operation is a linear function of the processing time. In the case of discretely controllable processing times, there is a finite set of possible processing times and costs for every operation O_{ij}.

We present the first known polynomial time approximation schemes for problems P1, P2, and P3, when the number m of machines and the number μ of operations per job are fixed. Our algorithms can handle both, continuously and discretely controllable processing times, and they can be extended to the case of convex piecewise linear processing times and cost functions. These results improve the 4/3-approximation algorithm for problem P3 described in Nowicki [65].

CHAPTER 6: In this chapter we study a generalization of the job shop scheduling problem (see Chapter 5) called the *flexible job shop problem* [62], which models a wide variety of problems encountered in real manufacturing systems [18, 78]. In the flexible job shop problem an operation O_{ij} can be processed by any machine from a given group $M_{ij} \subseteq M$. The processing time of operation O_{ij} on machine $k \in M_{ij}$ is p_{ij}^k. The goal is to choose for each operation O_{ij} an eligible machine and a starting time so that the maximum completion time $C_{\max}$ over all jobs is minimized.

We study the preemptive and non-preemptive versions of the flexible job shop scheduling problem when the number of machines m and the number of operations per job μ are fixed. We generalize the techniques described in [44] for the job shop scheduling problem and design a linear time approximation scheme for the flexible job shop problem.

Moreover, we present a linear time approximation scheme for the preemptive version of the flexible job shop problem without migration. No migration means that each operation must be processed by a unique machine. Hence if an operation is preempted, its processing can only be resumed on the same machine on which it was being processed before the preemption. We also study the preemptive flexible job shop problem with migration and in which every operation has both a release time and a delivery time. We present a $(2+\varepsilon)$-approximation algorithm for it. Both algorithms produce solutions with only a constant number of preemptions.

CHAPTER 7: In the classical *Knapsack Problem* (KP) we have a set of n items and a knapsack of limited capacity. To each item we associate a positive profit p_j and a positive weight w_j. The problem calls for selecting the set of items with maximum overall profit among those whose overall weight does not exceed the knapsack capacity $c > 0$. The *k-item Knapsack Problem* (kKP), is a KP in which an upper bound of k is imposed on the number of items that can be selected in a solution.

We contribute by presenting a new idea. We show that appropriate combinations of arithmetic and geometric rounding techniques yields novel and powerful rounding methods. To the best of our knowledge, these techniques have never been combined together. By using the described rounding technique, we first show that any given instance can be reduced to a smaller one having at most $\min\{n, O(k/\varepsilon)\}$ items. Then we present the first known EPTAS for kKP requiring linear space and running time $O(n + k \cdot (1/\varepsilon)^{O(1/\varepsilon)})$. Our algorithm is superior to the one provided by Caprara et al. [10]: they yield a $(1-\varepsilon)$-approximate solution within $O(n^{\lceil 1/\varepsilon \rceil - 2})$ and $O(n^{\lceil 1/\varepsilon \rceil - 1})$ run-

ning time, for KP and kKP respectively. Since KP is a special case of kKP, we also improve the previous result for KP. Finally we present a faster FPTAS for kKP that runs in $O(n + k/\varepsilon^4 + 1/\varepsilon^5)$ time and has a bound of $O(n + 1/\varepsilon^4)$ on space requirements. The previous known FPTAS provided by Caprara et al. [10] requires $O(nk^2/\varepsilon)$ time and $O(n + k^3/\varepsilon)$ space.

CHAPTER 8: The running time of a PTAS may depend arbitrarily on the reciprocal of the error bound with better approximations yielding longer running times. In this chapter we show that if problem $\mathcal{P}$ belongs to class EPTAS then, for every fixed $r > 1$, it is possible to obtain in constant time an r-approximation algorithm that runs in polynomial time and where the hidden constant is reasonably small and independent of the accuracy r. By using this result and the polynomial time approximation schemes described in this thesis, we show that for every fixed $r > 1$ there exists a r-approximation algorithms for problems $1|r_j, prec|L_{\max}$, $P|r_j|L_{\max}$, $Rm||C_{\max}$ with costs, and $Jm|op \leq \mu|C_{\max}$, running in linear time, where the hidden constant is always reasonably small and independent of the accuracy r. These approximation algorithms are to some extent *optimal*, since their time complexity is the best possible.

Remark 1.2 *Throughout this thesis we discuss several transformations of the input problem in order to add structure to the input data. Some transformations may potentially increase the objective function value by a factor of* $1 + O(\varepsilon)$ $(1 - O(\varepsilon)$ *for maximization problems), so we can perform a constant number of them while still staying within* $1 + O(\varepsilon)$ $(1 - O(\varepsilon))$ *of the original optimum. Others are transformations which do not increase the objective function value. When we describe the first type of transformation, we shall say it produces* $1 + O(\varepsilon)$ $(1 - O(\varepsilon))$ *loss, while the second produces no loss. Moreover, we shall assume that* $1/\varepsilon$ *is integral for simplicity.*

Publications. The work described in this thesis is mostly contained in papers published earlier [17, 39, 40, 41, 58, 60, 61] and in preparation [59]. Chapter 2 appeared in [59, 60]. Parts of Chapter 3 are joint work with Klaus Jansen and appeared in [39, 61]. Chapter 4 and parts of Chapter 5 are joint work with Aleksei Fishkin and Klaus Jansen and appeared in [17]. Parts of Chapter 5 and Chapter 6 are joint work with Klaus Jansen and Roberto Solis-Oba. Chapter 7 appeared in [58].

Chapter 2

Scheduling on a Single Machine

2.1 Introduction

In this chapter we consider the following single machine scheduling problem. A set, $J = \{J_1, ..., J_n\}$, of n jobs is to be processed without interruption on a single machine. For each job J_j there is an interval $[\ell_j, u_j]$, $0 \leq \ell_j \leq u_j$, specifying its possible processing times. The cost for processing job J_j in time ℓ_j is $c_j^\ell \geq 0$ and for processing it in time u_j the cost is $c_j^u \geq 0$. For any value $\delta_j \in [0, 1]$ the cost for processing job J_j in time $p_j(\delta_j) = \delta_j \ell_j + (1 - \delta_j)u_j$ is $c_j(\delta_j) = \delta_j c_j^\ell + (1 - \delta_j)c_j^u$, where δ_j is the *compression parameter*. Additionally, each job J_j has a release date $r_j \geq 0$ when it first becomes available for processing and, after completing its processing on the machine, requires an additional delivery time $q_j \geq 0$; if s_j $(\geq r_j)$ denotes the time J_j starts processing, then it has been delivered at time $s_j + p_j(\delta_j) + q_j$, for compression parameter δ_j. Feasible schedules are further restricted by job precedence constraints given by the partial order $\prec$, where $J_j \prec J_k$ means that job J_k must be processed after job J_j. Let π be a permutation of the set J that is consistent with the precedence constraints, and $\pi(i)$ the job which is in position i in permutation π; π denotes a processing order of jobs. Denote by $Q(\delta, \pi)$ the (earliest) *maximum delivery time* of all the jobs for compression parameters $\delta = (\delta_1, ..., \delta_n)$ and processing order π, then

$$Q(\delta, \pi) = \max_{1 \leq i_1 \leq i_2 \leq n} (r_{\pi(i_1)} + \sum_{j=i_1}^{i_2} p_{\pi(j)}(\delta_{\pi(j)}) + q_{\pi(i_2)}).$$

The *total cost* of compression parameters δ is equal to $\sum_{j\in J} c_j(\delta_j)$, and the *total scheduling cost* for compression parameters δ and processing order π is defined as

$$K(\delta,\pi) = Q(\delta,\pi) + \sum_{j\in J} c_j(\delta_j).$$

The problem is to find δ^* and π^* minimizing $K(\delta,\pi)$.

When all processing times are fixed ($\ell_j = u_j$), the problem is equivalent to the well-known sequencing problem denoted as $1|r_j, prec|L_{\max}$ in Graham et al. [25]. Since the special case with fixed processing times and without precedence constraints (noted $1|r_j|L_{\max}$ in [25]) is strongly NP-complete [55], the stated problem is also strongly NP-complete.

Hall and Shmoys [29, 31] propose two polynomial time approximation schemes for problem $1|r_j|L_{\max}$, the running time of which are $O((\frac{n}{\varepsilon})^{O(1/\varepsilon)})$ and $O(n\log n + n(1/\varepsilon^{O(1/\varepsilon^2)}))$. For the corresponding problem with controllable processing times, Zdrzalka [82] gives a polynomial time approximation algorithm with a worst-case ratio of $3/2+\varepsilon$, where $\varepsilon > 0$ can be made arbitrarily small. When the precedence constraints are imposed and the job processing times are fixed ($1|r_j, prec|L_{\max}$), Hall and Shmoys [30] give a PTAS. This consists of executing, for $\log_2 \Delta$ times, an extended version of their previous PTAS for $1|r_j,|L_{\max}$, where Δ denotes an upper bound on the optimal value of any given instance whose data are assumed to be integral. This polynomial running time should be contrasted with the time complexity of their result for problem $1|r_j|L_{\max}$ [29, 31], where they were able to achieve a considerably better time.

We contribute in two ways. First, we show that for problem $1|r_j, prec|L_{\max}$ there exists a PTAS running in $O(n+\ell+1/\varepsilon^{O(1/\varepsilon)})$ time, where ℓ denotes the number of precedence constraints. Note that the time complexity of this PTAS is the best possible with respect to the input size and it is an improvement of previous results [29, 30, 31] with respect to both, the input size and $1/\varepsilon$. We remark that the constant that depends on $1/\varepsilon$ is now additive. Second we provide the first known PTAS for problem $1|r_j, prec|L_{\max}$ with controllable processing times that runs in linear time. This improves and generalizes all the previous results [29, 30, 31, 82].

2.2 Fixed Processing Times

As first step to the construction of a PTAS for problem $1|prec, r_j|L_{\max}$, we discuss several techniques which add structure to the input data. Here the main idea is to turn a difficult instance into an instance that is easier to

tackle. We show that the precedence graph can be simplified into a more structured graph. This simplification depends on the desired precision ε of approximation; the closer ε is to zero, the closer the modified graph should resemble the original one. With this simplified precedence structure it is possible to partition jobs into a constant number of subsets of jobs having similar precedence relations. Then, jobs belonging to the same subset can be grouped together into single jobs. The resulting instance is modified to obtain a constant number of instances with different release dates and delivery times; the final step consists of executing the extended Jackson's rule on this constant number of instances; the best schedule generated is output.

2.2.1 Simplifying the Input

We start providing some lower and upper bounds on the optimal value OPT. We define $P = \sum_{j=1}^{n} p_j$, $p_{\max} = \max_j p_j$, $r_{\max} = \max_j r_j$, $q_{\max} = \max_j q_j$ and $d = \max_{j=1,\dots,n}\{r_j + p_j + q_j\}$. Let $LB = \max\{P, d\}$, we claim that $LB \leq OPT \leq 3LB$. Indeed, since P and d are lower bounds on OPT, LB is also a lower bound on OPT. We show that $3LB$ is an upper bound on OPT by exhibiting a schedule with value at most $3LB$. Starting from time $r_{\max}$ all jobs have been released and they can be scheduled one after the other in any fixed ordering of the jobs that is consistent with the precedence relation; this can be obtained by topologically sorting the precedence graph. Then every job is completed by time $r_{\max} + P$ and the maximum delivery time is bounded by $r_{\max} + P + q_{\max} \leq P + 2d \leq 3LB$. By dividing every release, delivery and processing time by LB, we may (and will) assume, without loss of generality, that $LB = 1$ and

$$1 \leq OPT \leq 3. \tag{2.1}$$

Following Lageweg, Lenstra and Rinnoy Kan [49], if $J_j \prec J_k$ and $r_j > r_k$, then we can reset $r_k := r_j$ and each feasible schedule will remain feasible. Similarly, if $q_j < q_k$ then we can reset $q_j := q_k$ without changing the objective function value of any feasible schedule. Thus, by repeatedly applying these updates we can always obtain an equivalent instance that satisfies

$$J_j \prec J_k \Longrightarrow (r_j \leq r_k \text{ and } q_j \geq q_k) \tag{2.2}$$

Such a resetting requires $O(\ell)$ time. In the following we will assume, without loss of generality, that condition (2.2) holds.

A technique used by Hall and Shmoys [29] allows us to deal with only a constant number of release dates and delivery times. The idea is to round

each release and delivery time down to the nearest multiple of $i\varepsilon$, for $i \in \mathbb{N}$. Since $r_{\max} \leq d \leq 1$, the number of different release dates and delivery times is now bounded by $1/\varepsilon + 1 < 2/\varepsilon$ (for simplicity, we assume $0 < \varepsilon < 1$). Clearly, the optimal value of this transformed instance cannot be greater than OPT. Every feasible solution for the modified instance can be transformed into a feasible solution for the original instance just by adding ε to each job's starting time, and reintroducing the original delivery times. It is easy to see that the solution value may increase by at most 2ε. Thus in the remainder of this paper, without loss of generality, we shall restrict our attention to the case where there are only a constant number of release dates and delivery times.

Therefore, we will assume henceforth that the input instance has a constant number of release dates and delivery times, and that condition (2.2) holds. We shall refer to this instance as x. By the previous arguments, $OPT \geq OPT(x)$, where $OPT(x)$ denotes the optimal value for instance x.

Partitioning the Set of Jobs

Partition the set of jobs in two subsets:

$$\begin{aligned} L &= \{J_j : p_j > \varepsilon\}, \\ S &= \{J_j : p_j \leq \varepsilon\}. \end{aligned}$$

Let us say that L is the set of *large* jobs, while S the set of *small* jobs. Observe that the number of large jobs is bounded by $P/\varepsilon \leq 1/\varepsilon$. We further partition the set S of small jobs as follows. For each small job $J_j \in S$ consider the following three subsets of L:

$$\begin{aligned} Pre(j) &= \{J_i \in L : J_i \prec J_j\}, \\ Suc(j) &= \{J_i \in L : J_j \prec J_i\}, \\ Free(j) &= L - (Pre(j) \cup Suc(j)). \end{aligned}$$

Let us say that $T(j) = \{Pre(j), Suc(j), Free(j)\}$ represents a *3-partition of set* L with respect to job J_j. The number τ of distinct 3-partitions of L is clearly bounded by the number of small jobs and by $3^{|L|} \leq 3^{1/\varepsilon}$, therefore $\tau \leq \min\left\{n, 3^{1/\varepsilon}\right\}$. Let $\{T_1, ..., T_\tau\}$ denote the set of all distinct 3-partitions. Now, we define the *execution profile* of a small job J_j to be a 3-tuple $< i_1, i_2, i_3 >$ such that $r_j = \varepsilon \cdot i_1$, $q_j = \varepsilon \cdot i_2$ and $T(j) = T_{i_3}$, where $i_1, i_2 = 0, 1, ..., 2/\varepsilon$ and $i_3 = 1, ..., \tau$.

Corollary 2.1 *The number π of distinct execution profiles is bounded by $\pi \leq \tau \cdot 4/\varepsilon^2 = 2^{O(1/\varepsilon)}$.*

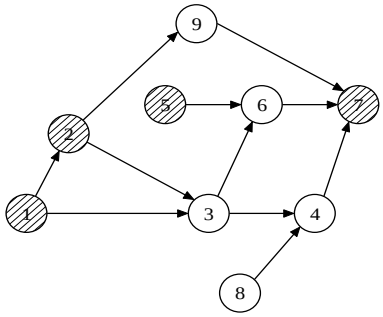

Figure 2.1: Graph of Example 2.1

Partition the set S of small jobs into π subsets, $S_1, S_2, ..., S_\pi$, such that jobs belonging to the same subset have the same execution profile. Clearly, $S = S_1 \cup S_2, ... \cup S_\pi$ and $S_h \cap S_i = \emptyset$, for $i \neq h$. We illustrate the above by the following.

Example 2.1 *Consider the precedence structure given by the graph in Figure 2.1. Shaded nodes represent large jobs, while the others denote small jobs. Assume that* $r_3 = r_4 = r_9$ *and* $q_3 = q_4 = q_9$. *Since* $Pre(3) = Pre(4) = Pre(9)$ *and* $Suc(3) = Suc(4) = Suc(9)$, *jobs* J_3, J_4 *and* J_9 *establish the same 3-partition of set* L *and therefore* $T(3) = T(4) = T(9)$. *Moreover, jobs* J_3, J_4 *and* J_9 *have the same execution profile since they have equal release dates and delivery times. Therefore, the set* $S = \{J_3, J_4, J_6, J_8, J_9\}$ *of small jobs is partitioned into 3 subsets* $S_1 = \{J_3, J_4, J_9\}$, $S_2 = \{J_6\}$ *and* $S_3 = \{J_8\}$.

Adding New Precedences

Let us say that job J_h is a *neighbor* of set S_i $(i = 1, ..., \pi)$ if:

- J_h is a small job;
- $J_h \notin S_i$;
- there exists a precedence relation between job J_h and some job in S_i.

Moreover, we say that J_h is a *front-neighbor* (*back-neighbor*) of S_i if J_h is a neighbor of S_i and there is a job $J_j \in S_i$ such that $J_j \prec J_h$ $(J_h \prec J_j)$.

Lemma 2.1 *A neighbor J_h of S_i cannot be at the same time a front-neighbor and a back-neighbor of S_i, for $i = 1, ..., \pi$.*

Proof. By contradiction assume that J_h is a front-neighbor and a back-neighbor of S_i. By this assumption, there exist two jobs $J_a, J_b \in S_i$ such that $J_a \prec J_h$ and $J_h \prec J_b$. Therefore, by condition (2.2) we have $r_a \leq r_h$, $q_a \geq q_h$, $r_h \leq r_b$ and $q_h \geq q_b$. Since $J_a, J_b \in S_i$, then $r_a = r_b$ and $q_a = q_b$, and hence $r_h = r_a = r_b$ and $q_h = q_a = q_b$.

It is easy to check that the following condition holds

$$J_x \prec J_y \Longrightarrow (Pre(x) \subseteq Pre(y) \text{ and } Suc(x) \supseteq Suc(y)).$$

Therefore, since $J_a \prec J_h$ and $J_h \prec J_b$, we have $Pre(a) \subseteq Pre(h)$, $Suc(a) \supseteq Suc(h)$, $Pre(h) \subseteq Pre(b)$ and $Suc(h) \supseteq Suc(b)$. Recall that $J_a, J_b \in S_i$ and therefore $Pre(a) = Pre(b)$ and $Suc(a) = Suc(b)$. It follows that $T(h) = T(a) = T(b)$, but also $r_h = r_a = r_b$ and $q_h = q_a = q_b$. Hence, job J_h has the same execution profile of J_a and J_b, and we have $J_h \in S_i$ by definition of S_i. The claim follows by observing that this is a contradiction, since by definition, a neighbor J_h of S_i cannot belong to S_i. ■

Let $n_i = |S_i|$ $(i = 1, ..., \pi)$, and let $(J_{1,i}, ..., J_{n_i,i})$ denote any fixed ordering of the jobs from S_i that is consistent with the precedence relation. In the rest of this section we consider the restricted problem in which the jobs from S_i are processed according to this fixed ordering. Furthermore, every back-neighbor (front-neighbor) J_h of S_i $(i = 1, ..., \pi)$ must be processed before (after) every job from S_i. This can be accomplished by adding a directed arc from $J_{j,i}$ to $J_{j+1,i}$, for $j = 1, ..., n_i - 1$, and by adding a directed arc from J_h to $J_{1,i}$, if J_h is a back-neighbor of S_i, or else an arc from $J_{n_i,i}$ to J_h, if J_h is a front-neighbor. Note that the number of added arcs can be bounded by $n + \ell$. The above is illustrated by the following.

Example 2.2 *Consider Example 2.1. Observe that (J_3, J_4, J_9) is an ordering of the jobs from S_1 that is consistent with the precedence relation. Job J_8 is a back-neighbor of S_1, while job J_6 is a front-neighbor of S_1. The new precedence structure is given by the graph in Figure 2.2, where the new added arcs are emphasized.*

We observe that condition (2.2) is valid also after these changes. Indeed, if J_h is a back-neighbor of S_i then there is a job $J_j \in S_i$ such that $J_h \prec J_j$, and therefore by condition (2.2) we have $r_h \leq r_j$ and $q_h \geq q_j$. But, the jobs from S_i have the same release dates and delivery times, therefore $r_h \leq r_j$ and $q_h \geq q_j$ for each $J_j \in S_i$. It follows that if we restrict J_h to be processed

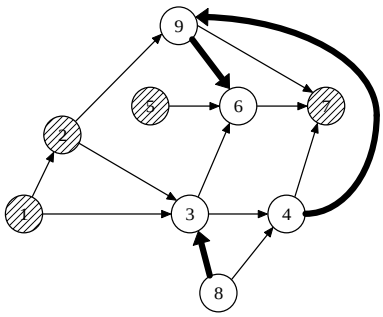

Figure 2.2: Graph of Example 2.2

before the jobs from S_i, condition (2.2) is still valid. Similar arguments hold if J_h is a front-neighbor. Moreover, all the jobs from S_i have the same release dates and delivery times, therefore condition (2.2) is still satisfied, if we restrict these jobs to be processed in any fixed ordering that is consistent with the precedence relation.

In Section 2.2.3 we show that these new precedence constraints do not increase the length of the optimal schedule considerably. Here we observe that the new added precedence arcs do not create cycles in the resulting precedence graph. (Note that a cycle means that there are at least two jobs, J_a and J_b, such that $J_a \prec J_b$ and $J_b \prec J_a$; clearly this is an unacceptable situation.) If J_h is a back-neighbor of S_i $(i = 1, ..., h)$ then a new directed arc has been added from J_h to $J_{1,i}$. This may generate a cycle if and only if there exists a job J_k from S_i such that $J_k \prec J_h$, and therefore if and only if J_h is also a front-neighbor of S_i. But this is prevented by Lemma 2.1. Similar arguments hold if J_h is a front-neighbor.

Grouping Small Jobs

Consider the jobs $(J_{1,i}, ..., J_{n_i,i})$ from subset S_i, for $i = 1, ..., \pi$, sorted according to the precedence relations. Let $J_{j,i}$ and $J_{j+1,i}$ be two consecutive jobs from S_i such that $p_{j,i} + p_{j+1,i} \leq \varepsilon$, for $j = 1, ..., n_i - 1$. We "group" together these two jobs to form a *grouped* job having the same release and delivery time as $J_{j,i}$ (and $J_{j+1,i}$), but processing time equal to the sum of the processing times of $J_{j,i}$ and $J_{j+1,i}$, i.e., $p_{j,i} + p_{j+1,i}$. (This is equivalent to say that we shall consider only those schedules where jobs $J_{j,i}$ and $J_{j+1,i}$ are processed together, i.e., $J_{j+1,i}$ just after $J_{j,i}$.) Furthermore, the new grouped job must be processed after the predecessors of $J_{j,i}$ and before the

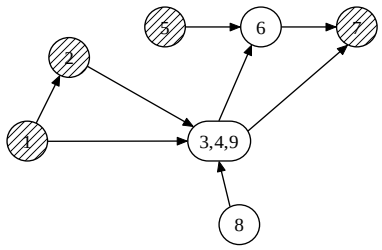

Figure 2.3: Graph of Example 2.3

successors of $J_{j+1,i}$. Observe that, with the exception of $J_{j,i}$ ($J_{j+1,i}$), the set of predecessors (successors) of $J_{j+1,i}$ ($J_{j,i}$) is the same as of $J_{j,i}$ ($J_{j+1,i}$). We repeat this process, by using the modified set of jobs, until there are no more couples of consecutive jobs with sum of processing times less or equal to ε. The same procedure is performed for all other subsets S_i.

Example 2.3 *Consider Example 2.2. By assuming that $p_3 + p_4 + p_9 \leq \varepsilon$, we can group jobs J_3, J_4 and J_9 together and get a new instance whose precedence structure is given by the graph in Figure 2.3.*

Let x' denote this new modified instance obtained from the given one x by performing the transformations as described so far. Let us denote the corresponding processing times, release dates and delivery times of jobs in x' by p'_j, r'_j and q'_j, respectively. Note that in instance x' the set of large jobs is the same as in x, and all the new grouped jobs are still "small", i.e., their processing times are not greater than ε. For simplicity of notation, let us use again L and S to denote, respectively, the set of large and small jobs with respect to instance x'.

Lemma 2.2 *The number ν of jobs in instance x' is bounded by $\nu \leq \frac{1}{\varepsilon} + \frac{2}{\varepsilon} + 2\pi = 2^{O(1/\varepsilon)}$.*

Proof. Let us use S'_i to denote set S_i after the described grouping procedure, for $i = 1, ..., \pi$. Let $n'_i = |S'_i|$. Observe that the sum $P'_i = \sum_{J_j \in S'_i} p'_j$ of processing times of the jobs from S'_i is equal to $P_i = \sum_{J_j \in S_i} p_j$.

First observe that every large job has processing time larger than ε and therefore there are at most $P/\varepsilon \leq 1/\varepsilon$ large jobs in x'. By the way in which small jobs are grouped together, at the end of the described grouping procedure, the sum of processing times of any couple of consecutive jobs

from S'_i cannot be smaller than ε. Therefore $\sum_{J_j \in S'_i} p'_j \geq \left\lfloor \frac{n'_i}{2} \right\rfloor \cdot \varepsilon$. Since $(\frac{n'_i}{2} - 1) \cdot \varepsilon < \left\lfloor \frac{n'_i}{2} \right\rfloor \cdot \varepsilon$, we have $\sum_{i=1}^{\pi} n'_i < 2(\pi + \sum P'_i/\varepsilon) \leq 2(\pi + 1/\varepsilon)$. ■

Note that instance x' can be computed in $O(n + \ell + 2^{O(1/\varepsilon)})$ time: the time required to partition the set of jobs into π subsets can be bounded by $O(n + \ell + \pi)$; $O(n + \ell)$ is the time to add new precedences and group jobs.

We have already observed that the new precedence constraints added in Section 2.2.1 do not invalidate condition (2.2). Furthermore, let G denote a set of jobs that have been grouped together as described in this section. Then, all jobs from G have the same release date r_G and delivery time q_G, and set G has been replaced with a single job J_G having r_G and q_G as release date and delivery time, respectively. If there exist two jobs J_j and J_k such that $J_j \notin G$, $J_k \in G$ and $J_j \prec J_k$ ($J_k \prec J_j$), by condition (2.2), we know that $r_j \leq r_k$ and $q_j \geq q_k$ ($r_k \leq r_j$ and $q_k \geq q_j$), but, by construction, we have also $r_j \leq r_G$, $q_j \geq q_G$ and $J_j \prec J_G$ ($r_G \leq r_j$, $q_G \geq q_j$ and $J_G \prec J_j$). Therefore, grouping jobs does not invalidate condition (2.2).

Lemma 2.3 *For each couple of jobs, J_j and J_k, of instance x' the following condition holds: $J_j \prec J_k \Longrightarrow (r'_j \leq r'_k$ and $q'_j \geq q'_k)$.*

2.2.2 The Main Algorithm

We make use of the following algorithm that is often called the *extended Jackson's rule* (since this generalizes a procedure to solve $1||L_{\max}$ due to Jackson [38]): schedule the jobs starting at the smallest r_j-value; at each decision point t given by a release date or a finishing time of some job, schedule a job j with the following properties: $r_j \leq t$, all its predecessors are scheduled, and it has the largest delivery time.

Our PTAS consists of executing the extended Jackson's rule on a constant number of instances obtained from x' by changing the release dates and delivery times of jobs; the best schedule generated is output. Without loss of generality, let us renumber jobs such that $p'_1 \geq p'_2 \geq ... \geq p'_\lambda \geq ... \geq p'_\nu$, where λ denotes the number of large jobs. For $j = 1, .., \lambda$, let $R_j = \{r'_j + i\varepsilon : i \in \mathbb{N}$ and $r'_j + i\varepsilon \leq 3 - \varepsilon\}$. The release dates of large jobs $J_1, ..., J_\lambda$ are reset to new values taken from $R_1, ..., R_\lambda$, respectively. Depending on these values, the other release dates and delivery times may also change. More precisely, our main algorithm performs the following steps.

(S-1) Initialize the solution *BestFound* to be the empty solution and set the corresponding value V to infinity.

(S-2) For each $(\rho_1, ..., \rho_\lambda) \in R_1 \times ... \times R_\lambda$ such that $\rho_j \geq \max_{J_h \in Pre(j)} \rho_h$ (for $j = 1, ..., \lambda$):

(S-2.1) Modify instance x' to get instance x'' with release dates and delivery times equal to the following values

$$\begin{array}{ll} r''_j := \rho_j & \text{for } J_j \in L, \\ r''_j := \max\{r'_j, \max_{h:J_h \in Pre(j)} r''_h\}, & \text{for } J_j \in S, \\ q''_j := \max\{q'_j, \max_{h:r''_j < r''_h} q''_h\} & \text{for } J_j \in L, \\ q''_j := \max\{q'_j, \max_{h:J_h \in Suc(j)} q''_h\}, & \text{for } J_j \in S. \end{array} \tag{2.3}$$

(S-2.2) Apply the extended Jackson's rule to instance x''. Let Σ and $m(x'', \Sigma)$ denote the solution and the solution value returned. If $m(x'', \Sigma) < V$, then let $BestFound := \Sigma$ and $V := m(x'', \Sigma)$.

(S-3) Return solution $BestFound$ of value V.

Step (S-2.1) can be implemented as follows. Release dates and delivery times are updated separately. Consider any fixed ordering of the jobs that is consistent with the precedence relation; this can be obtained by topologically sorting the precedence graph. To update release dates, the jobs are processed in this order; when job J_j is processed, if J_j is a small job then r''_j is set to $\max\{r'_j, \max_{h:J_h \in Pre(j)} r''_h\}$, otherwise to ρ_j. Let $J_1, ..., J_\nu$ denote any fixed ordering of the jobs that is consistent with the precedence relation and such that $r''_1 \leq ... \leq r''_\nu$. (Observe that in the following section we show that if $J_j \prec J_k$ then $r''_j \leq r''_k$ (see Lemma 2.4). Therefore it is always possible to find an ordering of the jobs that is consistent with the precedence relation and such that $r''_1 \leq ... \leq r''_\nu$.) To update the delivery times, the jobs are processed in this reverse order, i.e., from ν to 1; when job J_j is processed, if J_j is small then q''_j is set to $\max\{q'_j, \max_{h:J_h \in Suc(j)} q''_h\}$, otherwise to $\max\{q'_j, \max_{h:r''_j < r''_h} q''_h\}$.

2.2.3 Analysis

It is easy to see that $|R_j| \leq 3/\varepsilon$ for every $J_j \in L$. Therefore the number of different tuples $(\rho_1, ..., \rho_\lambda)$ can be bounded by $(3/\varepsilon)^{1/\varepsilon}$, since the number λ of large jobs is not greater than $1/\varepsilon$. Recall that ν is the number of jobs in instance x', and therefore the number of precedences in x' is at most ν^2. Then, Steps (S-2.1) and (S-2.2) can be implemented to run in $O(\nu^2)$ time. Hence, the total running time of Steps (S-1)-(S-3) is $1/\varepsilon^{O(1/\varepsilon)}$.

The next lemma shows that condition (2.2) is valid for each instance x'' considered in Step (S-2.1).

Lemma 2.4 *In Step (S-2.1), for each pair of jobs, J_j and J_k, of instance x'' the following condition holds: $J_j \prec J_k \Longrightarrow (r''_j \leq r''_k$ and $q''_j \geq q''_k)$.*

Proof. We first prove by induction that

$$J_j \prec J_k \Longrightarrow r''_j \leq r''_k. \tag{2.4}$$

Consider any fixed ordering $J_1, ..., J_\nu$ of the jobs that is consistent with the precedence relation. Trivially, condition (2.4) holds for set $\{J_1\}$. Assume that condition (2.4) holds for set $N_{k-1} = \{J_1, ..., J_{k-1}\}$, then we prove that condition (2.4) holds for set $N_k = \{J_1, ..., J_k\}$, for $2 \leq k \leq \nu$. If there is no job from set N_{k-1} that must be processed before J_k, then condition (2.4) holds for set N_k. Otherwise, we distinguish between the following cases:

1. If $J_k \in L$

   (a) and there is a large job J_j from set N_{k-1} such that $J_j \prec J_k$, then $r''_j \leq r''_k$ since $\rho_j \leq \rho_k$ (see Step (S-2));

   (b) and there is a small job J_j from set N_{k-1} such that $J_j \prec J_k$, then $r'_j \leq r'_k \leq \rho_k = r''_k$ by definition of R_k and by Lemma 2.3; furthermore $Pre(j) \subseteq Pre(k)$, since $J_j \prec J_k$, and $r''_k = \rho_k \geq \max_{J_h \in Pre(k)} \rho_h \geq \max_{J_h \in Pre(j)} \rho_h = \max_{J_h \in Pre(j)} r''_h$; therefore

   $$r''_k \geq \max\{r'_j, \max_{h:J_h \in Pre(j)} r''_h\} = r''_j.$$

2. If $J_k \in S$

   (a) and there is a large job J_j from set N_{k-1} such that $J_j \prec J_k$, then $J_j \in Pre(k)$ and therefore $r''_j \leq r''_k$;

   (b) and there is a small job J_j from set N_{k-1} such that $J_j \prec J_k$, then $r'_j \leq r'_k$ by Lemma 2.3; furthermore $Pre(j) \subseteq Pre(k)$ since $J_j \prec J_k$, and $\max_{J_h \in Pre(k)} r''_h \geq \max_{J_h \in Pre(j)} r''_h$; therefore

   $$r''_k = \max\{r'_k, \max_{h:J_h \in Pre(k)} r''_h\} \geq \max\{r'_j, \max_{h:J_h \in Pre(j)} r''_h\} = r''_j.$$

Hence, we have proved that if $J_j \prec J_k$ then $r''_j \leq r''_k$. This result guarantees that it is always possible to find an ordering of the jobs that is consistent with the precedence relation and such that $r''_1 \leq ... \leq r''_\nu$. Let $J_1, ..., J_\nu$ denote this ordering. In the following, we prove by induction that

$$J_j \prec J_k \Longrightarrow q''_j \geq q''_k. \tag{2.5}$$

Trivially, condition (2.5) holds for set $\{J_\nu\}$. Assume that condition (2.5) is true for set $N_{j+1} = \{J_{j+1}, ..., J_\nu\}$, then we prove that condition (2.5) holds for set $N_j = \{J_j, ..., J_\nu\}$, for $1 \leq j \leq \nu - 1$. If there is no job from set N_{j+1} that must be processed after J_j, then condition (2.4) holds for set N_j. Otherwise, we distinguish between the following cases:

1. If $J_j \in L$

   (a) and there is a large job J_k from set N_{j+1} such that $J_j \prec J_k$, then $r''_j \leq r''_k$, $q'_j \geq q'_k$ by Lemma 2.3, and $\max_{h:r''_j<r''_h} q''_h \geq \max_{h:r''_k<r''_h} q''_h$; it follows that

   $$q''_j = \max\{q'_j, \max_{h:r''_j<r''_h} q''_h\} \geq \max\{q'_k, \max_{h:r''_k<r''_h} q''_h\} = q''_k;$$

   (b) and there is a small job J_k from set N_{j+1} such that $J_j \prec J_k$, then $Suc(k) \subseteq Suc(j)$, and since $J_1, ..., J_\nu$ denote an ordering of the jobs that is consistent with the precedence relation, we have $Suc(k) \subseteq Suc(j) \subseteq N_{j+1}$. In the previous case (1.a) we have shown that $q''_j \geq \max_{h:J_h \in Suc(j)} q''_h$ and hence $q''_j \geq \max_{h:J_h \in Suc(k)} q''_h$. By observing that $q''_j \geq q'_j$ and $q'_j \geq q'_k$ by Lemma 2.3, we have

   $$q''_j \geq \max\{q'_k, \max_{h:J_h \in Suc(k)} q''_h\} = q''_k.$$

2. If $J_j \in S$

   (a) and there is a large job J_k from set N_{j+1} such that $J_j \prec J_k$, then $q''_j \geq q''_k$ since $J_k \in Suc(j)$;

   (b) and there is a small job J_k from set N_{j+1} such that $J_j \prec J_k$, then $q'_j \geq q'_k$ by Lemma 2.3; furthermore $Suc(j) \supseteq Suc(k)$ since $J_j \prec J_k$, and $\max_{J_h \in Suc(j)} q''_h \geq \max_{J_h \in Suc(k)} q''_h$; therefore

   $$q''_j = \max\{q'_j, \max_{J_h \in Suc(j)} q''_h\} \geq \max\{q'_k, \max_{J_h \in Suc(k)} q''_h\} = q''_k.$$

■

Now, we examine an artificial situation that we shall use as tool in analyzing our algorithm. Focus on instance x and a particular optimal solution Σ^* for x, consisting of job starting times $s^*_1, s^*_2, ..., s^*_n$. If $i\varepsilon \leq s^*_j < (i+1)\varepsilon$, for some $i \in \mathbb{N}$, then let $\rho^*_j := i\varepsilon$. Consider the modified instance x^* in which the processing times of all jobs J_j remain unchanged, while release r^*_j and delivery q^*_j times are set as follows

$$\begin{array}{ll} r_j^* := \rho_j^* & \text{for } J_j \in L, \\ r_j^* := \max\{r_j, \max_{h:J_h \in Pre(j)} r_h^*\}, & \text{for } J_j \in S, \\ q_j^* := \max\{q_j, \max_{h:r_j^* < r_h^*} q_h^*\} & \text{for } J_j \in L, \\ q_j^* := \max\{q_j, \max_{h:J_h \in Suc(j)} q_h^*\}, & \text{for } J_j \in S. \end{array} \tag{2.6}$$

By using the same arguments as in the proof of Lemma 2.4, we can easily show that condition (2.2) holds also for instance x^*. Furthermore, let $OPT(x^*)$ denote the optimum value for x^*. Then we have the following

Lemma 2.5 $OPT(x^*) = OPT(x)$.

Proof. Instance x^* is obtained from x by increasing (or leaving unchanged) release dates and delivery times. Therefore, any feasible solution for x^* is also a feasible solution for x and $OPT(x^*) \geq OPT(x)$. The claim follows by proving that there exists a solution for x^* of value $\leq OPT(x)$.

Consider the optimal solution Σ^* for x, consisting of job starting times $s_1^*, s_2^*, ..., s_n^*$. It is easy to check that $r_j^* \leq s_j^*$ $(j = 1, ..., n)$ and, therefore, we can schedule the jobs of instance x^* as in Σ^*: the starting time of job J_j is s_j^*, for $j = 1, ..., n$. Let J_c be the job that finishes last, i.e., its delivery is completed last, then the value of this solution is equal to $s_c^* + p_c + q_c^*$. If we prove that $s_c^* + p_c + q_c^* \leq OPT(x)$, then the claim follows.

We prove that $s_c^* + p_c + q_c^* \leq OPT(x)$ by induction. Let $J_1, ..., J_n$ denote any fixed ordering of the jobs that is consistent with the precedence relation and such that $r_1^* \leq ... \leq r_n^*$. (Note that this is possible since condition (2.2) holds for instance x^*.) If $c = n$, then $q_c^* = q_c$ and $s_c^* + p_c + q_c^* \leq OPT(x)$. Otherwise, assume that $s_j^* + p_j + q_j^* \leq OPT(x)$ for every $j = c+1, ..., n$ (induction hypothesis).

If J_c is a large job, let J_h denote the job with $r_c^* < r_h^*$ and $q_h^* = \max_{j:r_c^* < r_j^*} q_j^*$ (ties are broken arbitrarily). Since $r_c^* < r_h^*$ and $r_c^* = \rho_c^*$, it follows that $r_c^* \leq s_c^* < r_h^* \leq s_h^*$, and job c is processed before job h in Σ^*, i.e. $s_h^* \geq s_c^* + p_c$. From induction hypothesis we know that $s_h^* + p_h + q_h^* \leq OPT(x)$, and hence $s_c^* + p_c + p_h + q_h^* \leq OPT(x)$. Observe that $s_c^* + p_c + q_c \leq OPT(x)$. It follows that $s_c^* + p_c + \max\{q_h^*, q_c\} \leq s_c^* + p_c + \max\{(p_h + q_h^*), q_c\} \leq OPT(x)$. Note that $q_c^* = \max\{q_h^*, q_c\}$, and we have $s_c^* + p_c + q_c^* \leq OPT(x)$.

Otherwise, if J_c is a small job, let J_h denote the job such that $J_h \in Suc(c)$ and $q_h^* = \max_{j:J_j \in Suc(c)} q_j^*$ (ties are broken arbitrarily). Since $J_h \in Suc(c)$, it follows that job J_c must be processed before job J_h, i.e. $s_h^* \geq s_c^* + p_c$. From induction hypothesis we know that $s_h^* + p_h + q_h^* \leq OPT(x)$, and hence $s_c^* + p_c + p_h + q_h^* \leq OPT(x)$. It follows that $s_c^* + p_c + \max\{q_h^*, q_c\} \leq$

$s_c^* + p_c + \max\{(p_h + q_h^*), q_c\} \leq OPT(x)$, since $s_c^* + p_c + q_c \leq OPT(x)$. The claim follows by observing that $q_c^* = \max\{q_h^*, q_c\}$. ■

By inequality (2.1) and by definition of large jobs, in any optimal solution the starting time of each large job cannot be later than $3-\varepsilon$. Therefore, $\rho_j^* \in R_j$, for each $J_j \in L$, and in one of the iterations of step (S-2), we have $(\rho_1, ..., \rho_\lambda) = (\rho_1^*, ..., \rho_\lambda^*)$. Now, for simplicity of notation, let us use x'' to denote the modified instance of step (S-2.1) when $(\rho_1, ..., \rho_\lambda) = (\rho_1^*, ..., \rho_\lambda^*)$. Consider instance x'' and ungroup all the jobs that have been grouped together in Section 2.2.1: if job J_j is part of a grouped job J_g then, when we ungroup, we assume that the release date r_j'' and delivery time q_j'' of job J_j are equal to r_g'' and q_g'', respectively. Let x_u'' denote the resulting instance.

Lemma 2.6 *In instance x_u'', $r_j'' = r_j^*$ and $q_j'' = q_j^*$, for $j = 1, ..., n$.*

Proof. In Section 2.2.1 we modified the precedence structure by adding new precedences among jobs. We claim that the introduction of these precedences do not change sets $Pre(j)$ and $Suc(j)$ of jobs. Recall that we restricted the problem such that every back-neighbor (front-neighbor) J_h of S_i $(i = 1, ..., \pi)$ must be processed before (after) every job from S_i. If J_h is a back-neighbor of S_i, then there exists a job $J_j \in S_i$ such that $J_h \prec J_j$. It follows that $Pre(h) \subseteq Pre(j)$ and $Suc(j) \subseteq Suc(h)$. Recall that the jobs from S_i have, by definition, the same sets $Pre(j)$ and $Suc(j)$. Therefore, by assuming $J_h \prec J_j$ for each $J_j \in S_i$, we do not change the set $Pre(j)$ of any job $J_j \in S_i$, nor we change the set $Suc(h)$ of J_h. Similarly, if J_h is a front-neighbor of S_i, then by assuming that J_h is processed after all jobs from S_i, we do not change the set $Suc(j)$ of any job $J_j \in S_i$, neither set $Pre(h)$. Furthermore, jobs from S_i were further restricted to be processed in any fixed ordering that is consistent with the precedence relation. Clearly, adding precedences among jobs J_j having the same sets $Pre(j)$ and $Suc(j)$, does not change these sets for J_j. Therefore, the precedences added in Section 2.2.1 do not change the execution profile of any job.

According to (2.6), it is easy to check that small jobs having the same execution profile are set to the same r_j^* and q_j^* values. Consider any subset H of jobs having the same execution profile, and whose sum p_H of processing times is not greater than ε. Replace the jobs from H with a job J_H having the same execution profile as the jobs in H, and processing time p_H, i.e., J_H is a small job in the modified set of jobs. Update the release dates and delivery times of this modified set of jobs according to (2.6). Then it is easy to see that $r_H^* = r_j^*$ and $q_H^* = q_j^*$ for each $J_j \in H$. The claim follows by observing that instance x' is obtained from x by adding new precedences

which do not change the execution profile of any job, and by replacing subsets of jobs with new small jobs having the same execution profiles. ■

Consider step (S-2.2), and let $m(x'', \Sigma)$ denote the value of the solution Σ returned by the extended Jackson's rule when applied to instance x''. In order to show that the described algorithm is a PTAS, it is sufficient to prove the following

Lemma 2.7 $m(x'', \Sigma) \leq (1 + \varepsilon)OPT$.

Proof. We start by making two simple observations. First, consider any set C of jobs belonging to instance x''. Then ungroup the jobs from C and let C^u denote the resulting set of jobs. By Lemma 2.6, we have the following equation

$$\min_{J_j \in C} r''_j + \sum_{J_j \in C} p''_j + \min_{J_j \in C} q''_j = \min_{J_j \in C^u} r^*_j + \sum_{J_j \in C^u} p_j + \min_{J_j \in C^u} q^*_j. \tag{2.7}$$

Second, we observe that

$$OPT(x^*) \geq \min_{J_j \in C^u} r^*_j + \sum_{J_j \in C^u} p_j + \min_{J_j \in C^u} q^*_j \tag{2.8}$$

Let us define a critical job J_c as one that finishes last in Σ, i.e., its delivery is completed last. Associated with a critical job J_c there is a *critical sequence* B consisting of those jobs that are processed before J_c without idle time in between. Let us fix a critical job J_c and denote the last job J_b in the critical sequence with $q''_c > q''_b$ as an *interference job* for the critical sequence. Let C denote the set of jobs processed after J_b in the critical sequence. By the way that J_b was chosen, clearly $q''_k \geq q''_c$ for all jobs $J_k \in C$.

We claim that if there is no interference job then $m(x'', \Sigma) \leq OPT$, otherwise $m(x'', \Sigma) \leq OPT + p''_b$. Assume that an interference job J_b exists. Then, let s_b denote the starting time of job J_b, then $m(x'', \Sigma) = s_b + p''_b + \sum_{J_j \in C} p''_j + q''_c$. Let us say that a job is *available* at time t if it has been released and all its predecessors have been completed at time t. Notice that for the extended Jackson's rule to schedule job J_b at time s_b implies that no job $J_k \in C$ could have been available at time s_b, since otherwise such a job J_k would had priority over job J_b. Now, there are two alternative situations that could make a job $J_k \in C$ not available at time s_b: (1) $r''_k \leq s_b$ but there is a predecessor J_j of J_k that has not been yet scheduled at time s_b; (2) no job from C has been released at time s_b. Consider case (1) and, without loss of generality, let J_j denote some not yet scheduled predecessor

of J_k for which all its predecessors have completed their processing. By Lemma 2.4, we have $r''_j \leq r''_k$ and $q''_j \geq q''_k$. Recall that $q''_k \geq q''_c$ for all $J_k \in C$, and $q''_c > q''_b$ by definition of interference job. Therefore job J_j has $q''_j > q''_b$, $r''_j \leq s_b$, and it is available at time s_b since all its predecessors have completed their processing. By the definition of the extended Jackson's rule, no unscheduled job J_j could have been available at the time when J_b is processed with $q''_j > q''_b$, since otherwise such a job J_j would have had priority over job J_b. Therefore case (1) is not possible. According to case (2) we have $s_b < \min_{J_j \in C} r''_j$. Furthermore, by the way that J_b was chosen, clearly $q''_c \leq \min_{J_j \in C} q''_j$. Now we can bound the value of $m(x'', \Sigma)$, using (2.7) and (2.8), as

$$\begin{aligned} m(x'', \Sigma) &\leq p''_b + \min_{J_j \in C} r''_j + \sum_{J_j \in C} p_j + \min_{J_j \in C} q''_j \\ &\leq p''_b + OPT(x^*) \leq p''_b + OPT. \end{aligned}$$

If there is no interference job, using (2.7) and (2.8), it is easy to check that $m(x'', \Sigma) = \min_{J_j \in B} r''_j + \sum_{J_j \in B} p''_j + \min_{J_j \in B} q''_j \leq OPT$.

Then if there is no interference job or J_b is a small job we have $m(x'', \Sigma) \leq (1+\varepsilon) \cdot OPT$, by definition of small jobs (grouped or not). Now, consider the last case in which the interference job J_b is a large job. Recall that $q''_c > q''_b$ and $s_b < \min_{J_j \in C} r''_j$, thus $r''_c > r''_b$. But J_b cannot be a large job since according to (2.3) if $r''_c > r''_b$ then $q''_b \geq q''_c$. ■

Solution Σ can be easily transformed into a feasible solution for the original instance. This results in a $(1+3\varepsilon)$-approximate solution, and therefore we have given a polynomial time approximation scheme for $1|r_j, prec|L_{\max}$.

Theorem 2.1 *There exists a PTAS for problem $1|r_j, prec|L_{\max}$ that runs in $O(n + \ell + 1/\varepsilon^{O(1/\varepsilon)})$ time.*

2.3 Controllable Processing Times

In the following we generalize the techniques presented in Section 2.2 and present the first PTAS for $1|r_j, prec|L_{\max}$ with controllable processing times.

2.3.1 Simplifying the Input

We start by transforming any given instance x into a standard form. Let $d_j = \min\{\ell_j + c^{\ell}_j, u_j + c^{u}_j\}$, $D = \sum_{j=1}^{n} d_j$, $r_{\max} = \max_j r_j$ and $q_{\max} = \max_j q_j$.

Lemma 2.8 *Without loss of generality, we can assume that the following holds:*

- $1 \leq OPT(x) \leq 3$;
- $\max\{D, r_{\max}, q_{\max}\} \leq 1$;
- $0 \leq \ell_j \leq u_j \leq 3$ *and* $0 \leq c_j^u \leq c_j^\ell \leq 3$.

Proof. We begin by bounding the largest number occurring in any given instance x. First compute for every job J_j the *shortest job scheduling cost* $d_j = \min\{\ell_j + c_j^\ell, u_j + c_j^u\}$. Let $D = \sum_{j=1}^n d_j$, $r_{\max} = \max_j r_j$ and $q_{\max} = \max_j q_j$. Let $LB = \max\{D, r_{\max}, q_{\max}\}$, we claim that $LB \leq OPT(x) \leq 3LB$. Indeed, since D, $r_{\max}$ and $q_{\max}$ are lower bounds for $OPT(x)$, LB is also a lower bound for $OPT(x)$. We show that $3LB$ is an upper bound for $OPT(x)$ by exhibiting a schedule with value at most $3LB$. Starting from time $r_{\max}$ all jobs have been released and they can be scheduled one after the other in any fixed ordering of the jobs that is consistent with the precedence relation; this can be obtained by topologically sorting the precedence graph. Then every job can be completed by time $r_{\max} + D$ and the total scheduling cost is bounded by $r_{\max} + D + q_{\max} \leq 3LB$. By dividing every $\ell_j, u_j, c_j^\ell, c_j^u, r_j$ and q_j by LB, we may (and will) assume, without loss of generality, that $r_{\max}, q_{\max} \leq 1$, $LB = 1$ and $1 \leq OPT(x) \leq 3$.

Furthermore, we can assume, without loss of generality, that $0 \leq \ell_j \leq u_j \leq 3$ and $0 \leq c_j^u \leq c_j^\ell \leq 3$, for all jobs J_j: if $c_j^\ell < c_j^u$, then there exists an optimal solution with $\delta_j = 1$ (i.e., the processing time of job J_j is equal to ℓ_j). Then, we can reset $c_j^u := c_j^\ell$ without affecting the value of the objective function of any feasible schedule. Moreover, in any optimal solution the processing time of any job cannot be larger than 3; therefore, if $u_j > 3$ we can reduce, without loss of generality, the interval of possible processing times and get an equivalent instance by setting $c_j^u := \frac{u_j - 3}{u_j - \ell_j}(c_j^\ell - c_j^u) + c_j^u$ and $u_j = 3$. Similar arguments hold if $c_j^\ell > 3$. ■

By using the same arguments as in Section 2.2 we assume, without loss of generality, that condition (2.2) holds and that there are only a constant number of release dates and delivery times. Then, we perform the same transformations described in Subsection 2.2.1 for problem $1|r_j, prec|L_{\max}$ with two main differences. The first difference is that now we partition the set of large L and small S jobs differently:

$$\begin{aligned} L &= \left\{J_j : d_j > \varepsilon^2\right\}, \\ S &= \left\{J_j : d_j \leq \varepsilon^2\right\}. \end{aligned}$$

Observe that the number of large jobs is now bounded by $1/\varepsilon^2$. Therefore, the number τ of distinct 3-partitions of L is bounded by $\min\left\{n, 3^{1/\varepsilon^2}\right\}$. Let $\{T_1, ..., T_\tau\}$ denote the set of all distinct 3-partitions of L. Again, we define the *execution profile* of a small job J_j to be a 3-tuple $< i_1, i_2, i_3 >$ such that $r_j = \varepsilon \cdot i_1$, $q_j = \varepsilon \cdot i_2$ and $T(j) = T_{i_3}$, where $i_1, i_2 = 0, 1, ..., 1/\varepsilon$ and $i_3 = 1, ..., \tau$.

Corollary 2.2 *The number π of distinct execution profiles is bounded by* $\pi \leq \min\left\{n, 3^{1/\varepsilon^2}(1+1/\varepsilon)^2\right\}$.

The second difference is that jobs belonging to the same subset S_i ($i = 1, ..., \pi$) are merged together in a different way, as described below.

Compact Representation of Job Subsets

Consider set S_i ($i = 1, ..., \pi$). Note that the number of jobs in S_i may be $O(n)$. We replace the jobs from S_i with one *compact* job $J_i^{\#}$. Job $J_i^{\#}$ has the same release $r_i^{\#}$ and delivery time $q_i^{\#}$ as the jobs from S_i. Furthermore, if $J_j \prec J_k$ ($J_k \prec J_j$), $J_j \in S_i$ and $J_k \notin S_i$, then in the new modified instance we have $J_i^{\#} \prec J_k$ ($J_k \prec J_i^{\#}$). Finally, the processing requirement of $J_i^{\#}$ is specified by a finite set of alternative pairs of processing times and costs determined as follows. Consider the following set $V_S = \left\{\frac{\varepsilon}{\pi}, \frac{\varepsilon}{\pi}(1+\varepsilon), \frac{\varepsilon}{\pi}(1+\varepsilon)^2, ..., 3\right\}$ of $O(1/\varepsilon^3)$ many values. Recall that π is the number of distinct execution profiles. Let A_i (B_i) be the value obtained by rounding $\sum_{j\in S_i} \ell_j$ ($\sum_{j\in S_i} u_j$) up to the nearest value from set V_S. The possible processing times for $J_i^{\#}$ are specified by set P_i of values from V_S that fall in interval $[A_i, B_i]$, i.e., $P_i := V_S \cap [A_i, B_i]$. For each value $p \in P_i$, we compute the corresponding cost value $C_i(p)$ as follows. Consider the problems (S_i, p) of computing the minimum sum of costs for jobs belonging to S_i, when the total sum of processing times is at most p, for every $p \in P_i$. We can formulate problem (S_i, p) by using the following linear program $LP(S_i, p)$:

$$\begin{array}{lll} \min & \sum_{j\in S_i}(\delta_j c_j^{\ell} + (1-\delta_j)c_j^{u}) & \\ \text{s.t.} & \sum_{j\in S_i}(\delta_j \ell_j + (1-\delta_j)u_j) \leq p & \\ & 0 \leq \delta_j \leq 1 & J_j \in S_i \end{array}$$

By setting $\delta_j = 1 - x_j$, it is easy to see that an optimal solution for $LP(S_i, p)$ can be obtained by solving the following linear program:

$$\begin{array}{lll} \max & \sum_{j\in S_i}(c_j^\ell - c_j^u)x_j & \\ \text{s.t.} & \sum_{j\in S_i}(u_j - \ell_j)x_j \leq p - \sum_{j\in S_i}\ell_j & \\ & 0 \leq x_j \leq 1 & J_j \in S_i \end{array}$$

Note that $p - \sum_{j=1}^{n}\ell_j$ is non-negative, since $p \in P_i$ and the smallest value of P_i cannot be smaller than $\sum_{j=1}^{n}\ell_j$. The previous linear program corresponds to the classical knapsack problem with relaxed integrality constraints. By partially sorting jobs in nonincreasing ratio $(c_j^\ell - c_j^u)/(u_j - \ell_j)$ ratio order, the set $\{LP(S_i, p) : p \in P_i\}$ of $O(1/\varepsilon^3)$ many problems can be solved in $O(|S_i|\log\frac{1}{\varepsilon} + (1/\varepsilon^3)\log\frac{1}{\varepsilon})$ time by employing a median-finding routine (we refer to Lawler [50] for details). For each value $p \in P_i$, the corresponding cost value $C_i(p)$ is equal to the optimal solution value of $LP(S_i, p)$ rounded up to the nearest value of set V_S. It follows that the number of alternative pairs of processing times and costs for each compact job $J_i^\#$ is bounded by the cardinality of set P_i. Furthermore, since $\sum_{i=1}^{\pi}|S_i| \leq n$, it is easy to check that the amortized total time to compute the processing requirements of all compact jobs is $O(n(1/\varepsilon^3)\log\frac{1}{\varepsilon})$. Therefore, every set S_i is transformed into one compact job $J_i^\#$ with $O(1/\varepsilon^3)$ alternative pairs of costs and processing times. We use $S^\#$ to denote the set of compact jobs.

Now, let us consider the modified instance as described so far and turn our attention to the set L of large jobs. We map each large job $J_j \in L$ to a new job $J_j^\#$ which has the same release date, delivery time and set of predecessors and successors as job J_j, but a more restricted set of possible processing times and costs. More precisely, let A_j (B_j) be the value obtained by rounding ℓ_j (u_j) up to the nearest value from set $V_L = \{\varepsilon^3, \varepsilon^3(1+\varepsilon), \varepsilon^3(1+\varepsilon)^2, ..., 3\}$. The possible processing times for $J_j^\#$ are specified by set $P_j := V_L \cap [\ell_j, u_j]$. For each value $p \in P_j$, the corresponding cost value $C_j(p)$ is obtained by rounding up to the nearest value of set V_L the cost of job J_j when its processing time is p. We use $L^\#$ to denote the set of jobs obtained by transforming jobs from L as described so far.

Let $f(x, \varepsilon)$ denote this modified instance. We observe that $f(x, \varepsilon)$ can be computed in $O(n(1/\varepsilon^3)\log\frac{1}{\varepsilon} + 2^{O(1/\varepsilon^2)})$ time: the time required to partition the set of jobs into π subsets can be bounded by $O(n+\ell+2^{O(1/\varepsilon^2)})$; $O(n+\ell)$ is the time to add new precedences; $O(n(1/\varepsilon^3)\log\frac{1}{\varepsilon})$ is the time to compute the alternative pairs of costs and processing times. Moreover, this new instance has at most $\nu = 3^{1/\varepsilon^2}\cdot(1+1/\varepsilon)^2 + 1/\varepsilon^2$ jobs; each job has a constant number of alternative pairs of costs and processing times. Now let us focus on $f(x, \varepsilon)$ and consider the problem of finding the schedule for

$f(x, \varepsilon)$ with the minimum scheduling cost such that compact jobs can be preempted, while interruption is not allowed for jobs from $L^{\#}$.

The following lemma shows that the optimal solution value of $f(x, \varepsilon)$ has value close to $OPT(x)$. Moreover, it gives a bound on the number of preempted jobs. (A proof of the following lemma can be found in subsection 2.3.3.)

Lemma 2.9 *For any fixed $r > 1$ and any x, it is possible to compute in constant time an optimal solution for $f(x, \varepsilon)$ with at most $1/\varepsilon$ preempted compact jobs. Moreover, $OPT(f(x, \varepsilon)) \leq (1 + 4\varepsilon)OPT(x)$.*

2.3.2 Generating a Feasible Solution

In this subsection we show how to transform the optimal solution y^* for instance $f(x, \varepsilon)$ into a $(1 + \varepsilon)$-approximate solution for instance x. This is accomplished as follows.

First, replace the jobs from $L^{\#}$ with the corresponding large jobs of instance x. Let $p_j^{\#}$ and $c_j^{\#}$ denote the processing time and cost, respectively, of job $J_j^{\#} \in L^{\#}$ according to solution y^*, then it is easy to check that the corresponding job $J_j \in L$ can be processed in time and cost at most $p_j^{\#}$ and $c_j^{\#}$, respectively.

Second, we replace each compact job $J_i^{\#}$ with the corresponding small jobs from set S_i as follows. Remove job $J_i^{\#}$, this clearly creates gaps into the schedule. Then, fill in the gaps by inserting the small jobs from set S_i according to any fixed ordering that is consistent with the precedence relation, and by allowing preemption; the processing time and cost of these small jobs are chosen according to the optimal solution of $LP(S_i, p_i^{\#})$ (see Subsection 2.3.1), where $p_i^{\#}$ denotes the processing time of job $J_i^{\#}$ according to solution y^*. (Recall that the optimal solution of $LP(S_i, p_i^{\#})$ chooses the processing requirements of jobs from S_i such that the sum of processing times is at most $p_i^{\#}$ and the sum of costs is minimum.) However, these replacements do not yield a feasible solution for x, since there may be a set M of preempted small jobs. By Lemma 2.9, we have that the number of preempted small jobs is at most $1/\varepsilon$. For each $J_j \in M$ let s_j be the time at which job J_j starts in the preemptive schedule. Remove each $J_j \in M$ and schedule J_j without interruption at time s_j with processing time p_j and cost c_j, where $p_j + c_j = d_j$. It is easy to see that the maximum delivery time may increase by at most $\sum_{J_j \in M} p_j$ and the cost by at most $\sum_{J_j \in M} c_j$.

Therefore, the solution value is increased by at most $\sum_{J_j \in M} d_j \leq |M|\varepsilon^2 \leq \varepsilon \leq \varepsilon \cdot OPT(x)$, since $M \subseteq S$ and $|M| \leq 1/\varepsilon$.

Finally, we have already observed that every feasible solution for the modified instance with only a constant number of release dates and delivery times can be transformed into a feasible solution for the original instance by simply adding ε to each job's starting time and reintroducing the original delivery times. This may increase the value of the solution by at most 2ε. Therefore, by Lemma 2.9, the value of the returned solution is at most $(1 + 7\varepsilon) \cdot OPT(x)$, that confirms that this construction does in fact yield an $(1 + O(\varepsilon))$-approximate solution of x. To conclude, we have shown that problem $1|r_j, prec|L_{\max}$ with controllable processing times admits a PTAS.

Theorem 2.2 *There exists a linear time PTAS for problem $1|r_j, prec|L_{\max}$ with controllable processing times.*

2.3.3 Analysis

In this section we prove Lemma 2.9. We first examine an artificial situation that we shall use as tool in the remaining part of the proof. Let us focus on instance x and let us relax our problem by allowing preemption only for small jobs. Let Σ^* denote an optimal solution of x when small jobs can be preempted and let $OPT_{relax}(x)$ denote its value. Clearly, $OPT_{relax}(x) \leq OPT(x)$. According to Σ^*, let δ^* be the compression parameters of jobs, and let $s_1^*, s_2^*, ..., s_n^*$ and $t_1^*, t_2^*, ..., t_n^*$ be the job starting and completion times, respectively. If $i\varepsilon \leq s_j^* < (i+1)\varepsilon$, for some $i \in \mathbb{N}$, then let $\rho_j^* := i\varepsilon$. Consider the modified instance x^* in which the processing time p_j^* and cost c_j^* of job J_j $(j = 1, ..., n)$ are fixed to $p_j(\delta_j^*)$ and $c_j(\delta_j^*)$, respectively, while release r_j^* and delivery q_j^* times are reset to the following values:

$$\begin{array}{ll} r_j^* := \rho_j^*, & \text{for } J_j \in L, \\ r_j^* := \max\{r_j, \max_{h:J_h \in Pre(j)} r_h^*\}, & \text{for } J_j \in S, \\ q_j^* := \max\{q_j, \max_{h:r_j^* < r_h^*} q_h^*\}, & \text{for } J_j \in L, \\ q_j^* := \max\{q_j, \max_{h:J_h \in Suc(j)} q_h^*\}, & \text{for } J_j \in S. \end{array} \tag{2.9}$$

By using the same arguments as in the proof of Lemma 2.4, we have that condition (2.2) is valid for instance x^*.

Lemma 2.10 *For each pair of jobs, J_j and J_k, of instance x^* the following condition holds: $J_j \prec J_k \Rightarrow (r_j^* \leq r_k^*$ and $q_j^* \geq q_k^*)$.*

Let $OPT_{relax}(x^*)$ denote the optimum value for x^* when small jobs can be preempted. Then we have the following

Lemma 2.11 $OPT_{relax}(x^*) = OPT_{relax}(x)$.

Proof. (The following proof is obtained by modifying the proof of Lemma 2.5.) Instance x^* is obtained from x by fixing job processing times and costs as in Σ^*, and by increasing (or leaving unchanged) release dates and delivery times. Therefore, any feasible solution for x^* is also a feasible solution for x and $OPT_{relax}(x^*) \geq OPT_{relax}(x)$. The claim follows by proving that there exists a solution for x^* of value at most $OPT_{relax}(x)$.

Consider the optimal solution Σ^*. It is easy to check that $r_j^* \leq s_j^*$ $(j = 1, ..., n)$ and, therefore, we can schedule the jobs of instance x^* exactly as in Σ^*: the starting and the completion time of job J_j are s_j^* and t_j^*, respectively, for $j = 1, ..., n$. Let J_c be the job whose delivery is completed last, then the value of this solution is equal to $t_c^* + q_c^*$. If we prove that $t_c^* + q_c^* \leq OPT_{relax}(x)$, then the claim follows.

We prove that $t_c^* + q_c^* \leq OPT_{relax}(x)$ by induction. Let $J_1, ..., J_n$ denote any fixed ordering of the jobs that is consistent with the precedence relation and such that $r_1^* \leq ... \leq r_n^*$. (Note that this is possible by Lemma 2.10.) If $c = n$, then $q_c^* = q_c$ and $t_c^* + q_c^* \leq OPT_{relax}(x)$. Otherwise, assume that $t_j^* + q_j^* \leq OPT_{relax}(x)$ for every $j = c + 1, ..., n$ (induction hypothesis).

If J_c is a large job, let J_h denote the job with $r_c^* < r_h^*$ and $q_h^* = \max_{j:r_c^* < r_j^*} q_j^*$ (ties are broken arbitrarily). Since $r_c^* < r_h^*$ and $r_c^* = \rho_c^*$, it follows that $r_c^* \leq s_c^* < r_h^* \leq s_h^*$, and job c is completed before job h starts in Σ^*, since J_c starts before job h and a large job cannot be preempted, i.e. $s_h^* \geq t_c^*$. From induction hypothesis we know that $t_h^* + q_h^* \leq OPT_{relax}(x)$, and hence $t_c^* + p_h^* + q_h^* \leq OPT(x)$. Observe that $t_c^* + q_c \leq OPT(x)$. It follows that $t_c^* + \max\{q_h^*, q_c\} \leq t_c^* + \max\{(p_h^* + q_h^*), q_c\} \leq OPT_{relax}(x)$. Note that $q_c^* = \max\{q_h^*, q_c\}$, and we have $t_c^* + q_c^* \leq OPT_{relax}(x)$.

Otherwise, if J_c is a small job, let J_h denote the job such that $J_h \in Suc(c)$ and $q_h^* = \max_{j:J_j \in Suc(c)} q_j^*$ (ties are broken arbitrarily). Since $J_h \in Suc(c)$, it follows that job J_c must be completed before job J_h starts, i.e. $s_h^* \geq t_c^*$. From the induction hypothesis we know that $t_h^* + q_h^* \leq OPT_{relax}(x)$, and hence $t_c^* + p_h^* + q_h^* \leq OPT_{relax}(x)$. It follows that $t_c^* + \max\{q_h^*, q_c\} \leq t_c^* + \max\{(p_h^* + q_h^*), q_c\} \leq OPT_{relax}(x)$, since $t_c^* + q_c \leq OPT_{relax}(x)$. The claim follows by observing that $q_c^* = \max\{q_h^*, q_c\}$. ■

Now consider the preemptive version of the extended Jackson's rule: schedule the jobs starting at the smallest r_j-value; at each decision point t given by a release date or a finishing time of some job, schedule a job j with the following properties: $r_j \leq t$, all its predecessors are scheduled, and it has the largest delivery time. A preemption occurs at a release date if the newly released job has a bigger delivery time than the that of the

job currently being processed. We shall denote this algorithm as PJR. PJR is known [49] to solve optimally problem $1|r_j, prec|L_{\max}$ when preemption is allowed (in the notation of Graham et al. [25] this problem is denoted $1|r_j, pmtm, prec|L_{\max}$). Therefore, if we apply PJR to instance x^* we obtain a solution whose value cannot be greater than $OPT_{relax}(x^*)$ (since this algorithm solves the more relaxed problem where every job can be preempted). Interestingly, the following lemma shows that no large job is preempted by the rule, and this because of release dates and delivery values.

Lemma 2.12 *The preemptive extended Jackson's rule when applied to instance x^* returns a solution where no large job is preempted.*

Proof. By contradiction, assume that there exists a large job J_j that is preempted by using PJR. Then, there exists a job J_k with $q_k^* > q_j^*$ that is released when J_j is processed, thus $r_j^* < r_k^*$. But J_j cannot be a large job since according to (2.9) if $r_j^* < r_k^*$ then $q_j^* \geq q_k^*$. ■

Recall that we have partitioned the set S of small jobs into π subsets, $S_1, S_2, ..., S_\pi$, such that jobs belonging to the same subset have the same execution profile. Now, according to (2.9), it is easy to check that jobs belonging to the same subset S_i $(i = 1, ..., \pi)$ are reset to the same r_j^*, q_j^*-values. Moreover, if J_h is a (front) back-neighbor of S_i $(i = 1, ..., \pi)$ then by Lemma 2.10 we have $r_h^* \leq r_j^*$ and $q_h^* \geq q_j^*$ ($r_j^* \leq r_h^*$ and $q_j^* \geq q_h^*$), for every $J_j \in S_i$. Therefore, we do not change the value of the solution returned by PJR if we assume that job J_h is processed before (after) the jobs from S_i. Observe that after these changes, if $J_j \prec J_k$ ($J_k \prec J_j$), $J_j \in S_i$ and $J_k \notin S_i$, then $J_j \prec J_k$ ($J_k \prec J_j$) for each $J_j \in S_i$. Again, we do not change the value of the solution returned by PJR if we assume that all the jobs from S_i are processed one after the other, according to any fixed ordering that is consistent with the precedence relation. For simplicity, let us use again x^* to denote this modified instance. Moreover, consider the instance x° obtained from instance x^* by replacing all the jobs from S_i $(i = 1, ..., \pi)$ with a single job having the same release date and delivery time as the jobs from S_i, and processing time and cost equal to $\sum_{J_j \in S_i} p_j^*$ and $\sum_{J_j \in S_i} c_j^*$, respectively. Then, it is straightforward to check that the solution value returned by PJR is equal to $OPT_{relax}(x^*)$.

Now we show that from $f(x, \varepsilon)$ it is possible to obtain an instance $x^\blacktriangle$ that is nearly the same as x°. Instance $f(x, \varepsilon)$ is obtained from x by replacing each large job J_j with a job $J_j^\#$ having the same release date and delivery time of J_j, but processing times belonging to P_j (see subsection 2.3.1). By definition of P_j, there exists a value $p_j \in P_j$ such that $0 \leq p_j - p_j^* \leq$

$\max\{\varepsilon^3, \varepsilon p_j^*\}$. The corresponding cost value $C_j(p_j)$ is obtained by rounding up to the nearest value of set V_L the cost of job J_j when its processing time is p_j. Recall that we are assuming, without loss of generality, that $\ell_j \leq u_j$ and $c_j^u \leq c_j^\ell$, for all jobs J_j. Therefore, since $p_j \geq p_j^*$, we have that the cost of job J_j when its processing time is p_j is not greater than the cost of job J_j when its processing time is p_j^*. It follows that $C(p_j) - c_j^* \leq \max\{\varepsilon^3, \varepsilon c_j^*\}$. Moreover, the jobs from set S_i $(i = 1, ..., \pi)$ have been replaced with a single job $J_i^\#$. Job $J_i^\#$ has the same release $r_i^\#$ and delivery time $q_i^\#$ as the jobs from S_i, but processing times belonging to P_i (see subsection 2.3.1). By definition of P_i, there exists a value $p_i \in P_i$ such that $0 \leq p_j - \sum_{J_j \in S_i} p_j^* \leq \max\{\varepsilon/\pi, \varepsilon \sum_{J_j \in S_i} p_j^*\}$. The corresponding cost $C_i(p_i)$ is obtained by first computing the minimum sum of costs $LP(S_i, p_i)$ for jobs belonging to S_i, when the total sum of processing times is at most p_i, and then by rounding $LP(S_i, p)$ up to the nearest value of set V_S. Note that $LP(S_i, p_i)$ is not greater than $\sum_{J_j \in S_i} c_j^*$ since $p_i \geq \sum_{J_j \in S_i} p_j^*$. Therefore, it is easy to check that $C(p_i) - \sum_{J_j \in S_i} c_j^* \leq \max\{\varepsilon/\pi, \varepsilon \sum_{J_j \in S_i} c_j^*\}$. Now, let us use $x^\blacktriangle$ to denote the instance obtained from $f(x, \varepsilon)$ by fixing the processing time and cost of each job $J_j^\#$ to p_j and $C(p_j)$, respectively; moreover, assume that the release dates and delivery times of jobs in $x^\blacktriangle$ are set as in x°. Then, instance $x^\blacktriangle$ is "nearly the same" as x° and the optimal solution value of $x^\blacktriangle$ may be greater than $OPT_{relax}(x^*)$ by at most $\sum_{J_j^\# \in L^\#} (\max\{\varepsilon^3, \varepsilon p_j^*\} + \max\{\varepsilon^3, \varepsilon c_j^*\}) + \sum_{J_i^\# \in S^\#} (\max\{\varepsilon/\pi, \varepsilon \sum_{J_j \in S_i} p_j^*\} + \max\{\varepsilon/\pi, \varepsilon \sum_{J_j \in S_i} c_j^*\}) \leq 4\varepsilon OPT(x)$.

By the previous arguments the optimal solution of instance $f(x, \varepsilon)$ can be easily obtained by executing the preemptive extended Jackson's rule on a constant number of instances obtained from $f(x, \varepsilon)$ by changing the release dates, delivery times, and fixing processing times and costs of jobs; the best schedule generated is output. Without loss of generality, let us renumber jobs such that $L^\# = \{J_1^\#, ..., J_\lambda^\#\}$, where $\lambda = |L^\#|$. For $j = 1, .., \lambda$, let $R_j = \{r_j^\# + i\varepsilon : i \in \mathbb{N} \text{ and } r_j^\# + i\varepsilon \leq 3 - \varepsilon\}$. The release dates of jobs $J_1^\#, ..., J_\lambda^\#$ are reset to new values taken from $R_1, ..., R_\lambda$, respectively. Depending on these values, the other release dates and delivery times may also change. Moreover, the processing requirements of each job $J_j^\#$ are fixed according to the set of alternative pairs $(P_j, C_j(P_j))$, as computed in subsection 2.3.1. More precisely, our main algorithm performs the following steps.

(S-1) Initialize the solution $BestFound$ to be the empty solution and set the corresponding value V to infinity.

(S-2) For each $(\tau_1, ..., \tau_\nu) \in P_1 \times ... \times P_\nu$ and for each $(\rho_1, ..., \rho_\lambda) \in R_1 \times ... \times R_\lambda$ such that $\rho_j \geq \max_{J_h^\# \in Pre(j)} \rho_h$ (for $j = 1, ..., \lambda$):

(S-2.1) Modify instance $f(x, \varepsilon)$ to get instance $x^\blacktriangle$ with release dates $r_j^\blacktriangle$, delivery times $q_j^\blacktriangle$, processing times $p_j^\blacktriangle$ and costs $c_j^\blacktriangle$ fixed to the following values

$$
\begin{aligned}
&p_j^\blacktriangle := \tau_j, \\
&c_j^\blacktriangle := C_j(\tau_j), \\
&r_j^\blacktriangle := \rho_j, && \text{for } J_j^\# \in L^\#, \\
&r_j^\blacktriangle := \max\{r_j^\#, \max_{h: J_h^\# \in Pre(j)} r_h^\blacktriangle\}, && \text{for } J_j^\# \in S^\#, \\
&q_j^\blacktriangle := \max\{q_j^\#, \max_{h: r_j^\diamond < r_h^\diamond} q_h^\blacktriangle\}, && \text{for } J_j^\# \in L^\#, \\
&q_j^\blacktriangle := \max\{q_j^\#, \max_{h: J_h^\# \in Suc(j)} q_h^\blacktriangle\}, && \text{for } J_j^\# \in S^\#.
\end{aligned}
$$

(S-2.2) Apply the preemptive extended Jackson's rule to instance $x^\blacktriangle$. Let Σ and $m(x^\blacktriangle, \Sigma)$ denote the solution and the solution value returned. If $m(x^\blacktriangle, \Sigma) < V$, then let $BestFound := \Sigma$ and $V := m(x^\blacktriangle, \Sigma)$.

(S-3) Return solution $BestFound$ of value V.

Since $OPT_{relax}(x^*) \leq OPT(x) \leq 3$, and by definition of large jobs, in any optimal solution the starting time of each large job cannot be later than $3 - \varepsilon$. Therefore, $\rho_j^* \in R_j$, for each $J_j^\# \in L^\#$, and in one of the iterations of step (S-2), we have job processing times, costs, release dates and delivery times such that the application of PJR on that instance returns the optimal solution.

Chapter 3

Scheduling on Identical Machines

3.1 Introduction

A bank of identical parallel machines is a setting that is important from both the theoretical and practical points of view. From the theoretical viewpoint, it is a generalization of the single machine, and a special case of the flexible job shop (see Chapter 6). From the practical point of view, it is important because the occurrence of resources that can be used in parallel is common in the real-word.

In this chapter, we discuss scheduling problems with identical parallel machines. The identical parallel machine scheduling model is defined as follows. We have a set of n jobs and m identical machines. Each job must be processed without preemption on one of the m machines, each of whom can process at most one job at a time. We consider both, the problem with fixed processing times, and the problem with controllable processing times. We contribute in two ways. First we show that the problem with fixed processing times, release dates and delivery times belongs to the class EPTAS: we prove this by exhibiting a PTAS that runs in $O(n + f(1/\varepsilon))$ time, where $f(1/\varepsilon)$ is a constant that depends on the error ε. Second, we address several variants of the problem with controllable processing times and provide the first known polynomial time approximation schemes for them. Moreover, we study the problem when preemption is allowed and describe efficient exact and approximate algorithms.

3.2 Fixed Processing Times

In the scheduling problem with identical parallel machines, release dates and delivery times, the input consists of m identical machines M_i, $(i = 1, ..., m)$, and n independent jobs J_j $(j = 1, ..., n)$. Each job J_j must be processed without interruption during p_j time units on a single machine M_i, $(i = 1, ..., m)$, which can process at most one job at a time. Each job has a release date $r_j \geq 0$, when it first becomes available for processing, and, after completing its processing on the machine, requires an additional delivery time $q_j \geq 0$; if s_j $(\geq r_j)$ denotes the time when J_j starts processing, then it is delivered at time $L_j = s_j + p_j + q_j$. Our objective is to minimize, over all possible schedules, the *maximum delivery time*, i.e., $L_{\max} = \max_j L_j$. The problem as stated is strongly NP-complete even if $m = 1$ [55]. In the notation of Graham et al. [25], this problem is denoted $P|r_j|L_{\max}$.

Hall and Shmoys [29, 31] give a PTAS for problem $P|r_j|L_{\max}$ [29], whose running time is $O(n^{\text{poly}(1/\varepsilon)})$. Note that the time complexity of this PTAS is polynomial for any fixed ε, but the exponent in the polynomial depends on $1/\varepsilon$. This should be contrasted with the algorithm of Shmoys (see [32] p. 370) and with that of Alon et al. [1] that achieve linear time (i.e., the exponential dependence on ε is only hidden in the constant) using integer programming in fixed dimension, for the special case of release dates and delivery times equal to zero (denoted $P||C_{\max}$ in [25]). To some extent, this is not surprising, since release dates and delivery times add a substantial degree of difficulty to the problem: to see this, observe that problem $P||C_{\max}$ is trivial for $m = 1$, weakly NP-complete for $m = 2$, and strongly NP-complete if and only if the number of machines is arbitrary [19]; on the other hand $P|r_j|L_{\max}$ is strongly NP-complete starting from $m = 1$ [55].

Moreover, the polynomial-time approximation scheme provided in [29] does not settle entirely the question of approximability of problem $P|r_j|L_{\max}$. The running time of a PTAS for a strongly NP-complete problem (that fulfills some natural conditions, see Theorem 1.2) cannot be a polynomial function of $1/\varepsilon$ as otherwise P=NP. Typical running times are $n^{O(1/\varepsilon)}$ or $2^{O(1/\varepsilon)}n$. While algorithms of the former kind are useless even for moderate values of $1/\varepsilon$ and n, the latter ones can return in a reasonable amount of time a good approximation for an enormous instance. In several cases, the development of algorithms of the second type requires new (and sometimes very complex) techniques. For some interesting problems (including the Euclidean TSP) only an $n^{\text{poly}(1/\varepsilon)}$ approximation scheme is known. Cesati and Trevisan [11] prove that for some problems (including natural ones) there cannot be approximation schemes running in time $f(1/\varepsilon)n^{O(1)}$, for any function f. (The

results reported in [11] are related to Parameterized Complexity Theory [16], and state that the existence of such algorithms would imply that the classes $W[P]$ and FPT are equal.) This motivates the definition of *efficient* PTAS as an approximation scheme running in time $f(1/\varepsilon)n^c$, where f is an arbitrary function and c is a constant independent of ε. Therefore, within the class PTAS, the class EPTAS of problems admitting an efficient PTAS plays an important role [11].

In the following we address the issue of whether problem $P|r_j|L_{\max}$ admits an efficient PTAS. We answer affirmatively to this problem by showing an EPTAS that runs in $O(n+f(1/\varepsilon))$ time, where $f(1/\varepsilon)$ is a constant that depends exponentially on $1/\varepsilon$. This linear complexity bound is a substantial improvement in terms of n compared to the above mentioned results for the problem. Note that the time complexity of this EPTAS is best possible with respect to the number of jobs. Moreover, the existence of a PTAS whose running time is also polynomial in $1/\varepsilon$ for $P|r_j|L_{\max}$ would imply P=NP.

The basic idea is to cluster the set of jobs into blocks and consider a restricted problem where jobs belonging to the same block are processed together. We show that such a restriction does not increase the length of the optimal schedule considerably. Furthermore this allows us to simplify the instance such that all jobs, except for a constant number, have large processing times. Then, by re-structuring the input, it is possible to show that only a constant number of different assignments of jobs to machines exists. We show that this also bounds to a constant the number of different schedules for each machine. Then we compute the optimal lengths of these schedules and use these values as input of an integer linear program (ILP) with a constant number of variables. The final ingredient is the use of Lenstra's famous algorithm [56] to solve (ILP).

We begin by providing some lower and upper bounds of the optimal value OPT. We define $P = \sum_{j=1}^{n} p_j$, $p_{\max} = \max_j p_j$, $r_{\max} = \max_j r_j$, $q_{\max} = \max_j q_j$ and $d = \max_{j=1,\ldots,n}\{r_j + p_j + q_j\}$. Let $LB = \max\{P/m, d\}$, we claim that $LB \leq OPT \leq 4LB$. Indeed, since P/m and d are lower bounds for OPT, LB is also a lower bound for OPT. We show that $4LB$ is an upper bound for OPT by exhibiting a schedule with value $4LB$. Starting from time $r_{\max}$ all jobs have been released and they can be scheduled by using the classical list scheduling algorithm: we assign the jobs one by one to the machines; every time a job is assigned, it is put on a machine with the current minimal workload. As the minimimal workload is at most P/m and as the newly assigned job adds at most $p_{\max}$ to the workload, the maximum delivery time always remains bounded by $r_{\max} + P/m + p_{\max} + q_{\max} \leq P/m + 3d \leq 4LB$. By dividing every release, delivery and processing time

by LB, we may (and will) assume, without loss of generality, that $LB = 1$ and

$$1 \leq OPT \leq 4. \tag{3.1}$$

3.2.1 Restricted Problem

In the following we cluster the set of jobs into blocks and consider only those schedules in which jobs belonging to the same block are processed one after the other on the same machine. We show that such a restriction does not increase the length of the optimal schedule considerably.

Following [31], we first perform a preprocessing step that will simplify the algorithm that follows. The idea is to round each release and delivery time down to the nearest multiple of $i\varepsilon$, for $i \in \mathbb{N}$. Since $r_{\max} \leq \max_{j=1,\ldots,n}\{r_j + p_j + q_j\} \leq 1$, the number of different release dates and delivery times is now bounded by $1/\varepsilon + 1 < 2/\varepsilon$ (we assume $0 < \varepsilon < 1$ for simplicity). Clearly, the optimal value of this transformed instance cannot be greater than OPT. Then, by adding ε to each job's starting time, and reintroducing the original delivery times, any feasible solution for the modified instance can be transformed into a feasible solution for the original instance. It is easy to see that the solution value may increase by at most 2ε. Thus in the remainder of this paper, with $1 + 2\varepsilon$ loss we shall restrict our attention to the case where there are only a constant number of release dates and delivery times.

Next, we partition the set of jobs into subsets $S_1, S_2, ..., S_\ell$ of jobs having the same release and delivery times, i.e., two jobs, J_a and J_b, belong to the same subset if and only if $r_a = r_b$ and $q_a = q_b$. Since we are assuming that there are at most $2/\varepsilon$ distinct delivery times, as well as $2/\varepsilon$ release dates, the number ℓ of subsets $S_1, S_2, ..., S_\ell$ is bounded by $4/\varepsilon^2$. Consider subset S_h, for $h = 1, ..., \ell$, and let J_a and J_b be two jobs from S_h such that $p_a, p_b < \varepsilon^3$. We "merge" together these two jobs to form a composed job J_c having the same release and delivery time as J_a (and J_b), but processing time equal to the sum of the processing times of J_a and J_b, i.e., $p_c = p_a + p_b$. (This is equivalent to say that we shall consider only those schedules where jobs J_a and J_b are processed together, i.e., one after the other on the same machine.) We repeat this process, by using the modified set of jobs, until at most one job J_j from S_h has $p_j < \varepsilon^3$. The same procedure is performed for all other subsets S_i. At the end of this process, there are at most $4/\varepsilon^2$ jobs, one for each subset S_i, having $p_j < \varepsilon^3$, all the other jobs J_j in S_i have $\varepsilon^3 \leq p_j \leq \max\{1, 2\varepsilon^3\}$. We motivate the described preprocessing step with the following

Lemma 3.1 *With $1+10\varepsilon$ loss, we can assume that there are at most $4/\varepsilon^2$ jobs with processing times less than ε^3, while the remaining jobs have processing times not greater than* $\max\{1, 2\varepsilon^3\}$.

Proof. Let us focus on instance I with rounded release dates and delivery times, but where jobs are not yet merged together. Consider instance $I^{\#}$ obtained from I by merging jobs as described so far. The claim follows by showing that there exists a schedule for $I^{\#}$ with value at most $(1+10\varepsilon)OPT$.

Let us consider an optimal schedule Σ for I of value $OPT(I)$. Denote the starting time of job J_j in Σ as s_j^*, for $j = 1, ..., n$. Modify I to obtain instance $\tilde{I}$ in which the processing times of all jobs remain unchanged, while release dates $\tilde{r}_j$ and delivery times $\tilde{q}_j$ are set as follows: for all jobs J_j with $p_j < \varepsilon^3$, $\tilde{r}_j := r_j$ and $\tilde{q}_j := q_j$, while for those jobs J_j with $p_j \geq \varepsilon^3$, $\tilde{r}_j := s_j^*$ and $\tilde{q}_j := OPT(I) - p_j - s_j^*$. It is easy to see that Σ is an optimal schedule for the modified instance $\tilde{I}$ and that $OPT(\tilde{I}) = OPT(I)$. Moreover, any schedule that is optimal for $\tilde{I}$ is also optimal for the original instance I.

Now, consider instance $\tilde{I}^{\#}$ obtained from $\tilde{I}$ by merging jobs as described in Section 3.2.1. Note that only jobs with processing times less than ε^3 can be merged together. Furthermore, instance $\tilde{I}$ is obtained from I by changing the release dates and delivery times of only those jobs having processing times greater or equal to ε^3. By these arguments, it is easy to see that any schedule that is feasible for $\tilde{I}^{\#}$ is feasible also for $I^{\#}$ (as well as for I). Therefore, by exhibiting a schedule for $\tilde{I}^{\#}$ with value at most $(1+10\varepsilon)OPT(\tilde{I})$, we show that there exists a schedule for $I^{\#}$ with value at most $(1+10\varepsilon)OPT(I)$. The claim follows by observing that $OPT(I) \leq OPT$.

Consider instance $\tilde{I}$, and let A_i denote the set of jobs from $\tilde{I}$ with processing times greater or equal to ε^3 that are processed on machine M_i according to Σ, for $i = 1, ..., m$. Since $\tilde{I}^{\#}$ is obtained from $\tilde{I}$ by merging only those jobs with processing times less than ε^3, we find the jobs from A_i again also in $\tilde{I}^{\#}$. Likewise, let B_i denote the set of jobs from $\tilde{I}$ with processing times less than ε^3, and that are processed on machine M_i according to Σ, for $i = 1, ..., m$. Let us use $B_{i,x,y}$ to denote the set of jobs from B_i having release dates and delivery times equal to x and y, respectively, for $x, y = 0, \varepsilon, 2\varepsilon, ...$

Now, we exhibit a schedule for $\tilde{I}^{\#}$ with value at most $(1+10\varepsilon)OPT(\tilde{I})$. In the first step, assign the jobs from A_i to machine M_i, for $i = 1, ..., m$. Let us use $B^{\#}$ to denote the set of jobs which remain unassigned after the first step. Let $B^{\#}_{x,y}$ denote the set of jobs from $B^{\#}$ having release dates and delivery times equal to x and y, respectively, for $x, y = 0, \varepsilon, 2\varepsilon, ...$ In the second step, for $x, y = 0, \varepsilon, 2\varepsilon, ...$ and for $i = 1, ..., m$, greedily assign to machine M_i a number of jobs from $B^{\#}_{x,y}$ until the total size of assigned jobs exceeds the size

of jobs from $B_{i,x,y}$ or there remain no jobs to be assigned. Let $B^{\#}_{i,x,y}$ denote the set of jobs from $B^{\#}_{x,y}$ assigned to machine M_i. It can be easily shown that all jobs are assigned in the second step. Furthermore, since the jobs that are merged together cannot have a total processing time larger than $2\varepsilon^3$, we have $\sum_{J_j \in B^{\#}_{i,x,y}} p_j \leq \sum_{J_j \in B_{i,x,y}} p_j + 2\varepsilon^3$, and $\sum_{x,y} \sum_{J_j \in B^{\#}_{i,x,y}} p_j \leq \sum_{J_j \in B_i} p_j + 8\varepsilon$. The claim follows by showing that by these assignments of jobs to machines it is possible to schedule jobs such that, for $i = 1, ..., m$, the maximum delivery time $L^{\#}_i$ on machine i is at most $L_i + 10\varepsilon$, where L_i is the maximum delivery time of machine i according to solution Σ.

Consider the following algorithm for problem $1|r_j|L_{\max}$, known as *Extended Jackson's Rule:* whenever the machine is free and one of more jobs is available for processing, schedule an available job with largest delivery time. Let us focus on instance $\tilde{I}^{\#}$ and assume that jobs are assigned to machines as described before. Then apply the Extended Jackson's Rule to schedule the jobs within each machine. Let use $\Sigma^{\#}_i$ to denote the resulting schedule of machine i. Let us define a critical job J_c as one that finishes last in $\Sigma^{\#}_i$, i.e., its delivery is completed last. Associated with a critical job J_c there is a critical sequence consisting of those jobs tracing backward from J_c to the first idle time in the schedule. Let us fix a critical job J_c and denote, if any, the last job J_b in the critical sequence with $\tilde{q}_c > \tilde{q}_b$ as interference job for the critical sequence. Let $C^{\#}$ denote the set of jobs processed after J_b in the critical sequence. In the following we prove that if there is no interference job then $L^{\#}_i \leq L_i + 8\varepsilon$, otherwise $L^{\#}_i \leq L_i + 8\varepsilon + p_b$.

To begin, we make two simple observations. First, in $\tilde{I}^{\#}$ for any set $S^{\#}$ of jobs processed by machine i, (and in particular for the set $C^{\#}$ of jobs in the critical sequence), there exists a set S of jobs in $\tilde{I}$ processed by machine M_i such that

$$\min_{J_j \in S^{\#}} \tilde{r}_j + \sum_{J_j \in S^{\#}} p_j + \min_{J_j \in S^{\#}} \tilde{q}_j \leq \min_{J_j \in S} \tilde{r}_j + \sum_{J_j \in S} p_j + \min_{J_j \in S} \tilde{q}_j + 8\varepsilon. \tag{3.2}$$

Second,

$$L_i \geq \min_{J_j \in S} \tilde{r}_j + \sum_{J_j \in S} p_j + \min_{J_j \in S} \tilde{q}_j. \tag{3.3}$$

Assume that an interference job J_b exists. Then, let s_b denote the starting time of job J_b, then $L^{\#}_i = s_b + p_b + \sum_{J_j \in C^{\#}} p_j + q_c$. Notice that for Jackson's rule to schedule job J_b at time s_b implies that no job $J_j \in C^{\#}$ could have been available at time s_b, since otherwise such a job J_j would have taken priority over job J_b. Thus, $s_b < \min_{J_j \in C^{\#}} \tilde{r}_j$. Furthermore, by

the way that J_b was chosen, clearly $q_c \leq \min_{J_j \in C^\#} \tilde{q}_j$. Now we can bound the value of $L_i^\#$, using (3.2) and (3.3), as

$$\begin{aligned} L_i^\# &\leq p_b + \min_{J_j \in C^\#} \tilde{r}_j + \sum_{J_j \in C^\#} p_j + \min_{J_j \in C^\#} \tilde{q}_j \\ &\leq p_b + L_i + 8\varepsilon. \end{aligned}$$

If there is no interference job, let $S^\#$ denote the critical sequence, and it is easy to check that $L_i^\# = \min_{J_j \in S^\#} \tilde{r}_j + \sum_{J_j \in S^\#} p_j + \min_{J_j \in S^\#} \tilde{q}_j \leq L_i + 8\varepsilon$. Then if there is no interference job or $p_b < 2\varepsilon^3$ we have $L_i^\# \leq L_i + 8\varepsilon + 2\varepsilon^3 \leq L_i + 10\varepsilon$ (we are assuming that $0 < \varepsilon < 1$).

Otherwise, assume that $p_b \geq 2\varepsilon^3$, then $J_b \in A_i$ and hence $\tilde{r}_b = s_b^*$ and $\tilde{q}_b = OPT(I) - p_b - s_b^*$. By definition of the extended Jackson's rule, no job $J_j \in C^\#$ could have been available at the time when J_b is processed, since otherwise such a job J_j would have taken priority over job J_b. Thus, $\tilde{r}_c > \tilde{r}_b$ and $\tilde{q}_c > \tilde{q}_b$, but p_b cannot be $\geq 2\varepsilon^3$ since by construction if $\tilde{r}_c > \tilde{r}_b$ then $\tilde{q}_b \geq \tilde{q}_c$. ■

In the remainder of this paper, without loss of generality we shall restrict our attention to those schedules where jobs are clustered as described so far. Moreover, it is easy to check that the number of jobs is now at most $\min\{n, O(m/\varepsilon^3)\}$. Indeed, by Lemma 3.1 and by inequalities (3.1) the length of the optimal solution cannot be larger than $4(1+10\varepsilon)$. It follows that the number of jobs with processing times larger or equal to ε^3 is bounded by $4(1+10\varepsilon)m/\varepsilon^3$, whereas by Lemma 3.1 there are at most $4/\varepsilon^2$ jobs with processing times less than ε^3.

3.2.2 The Structure of Parallel Schedules

In this section we transform instance $I^\#$ with jobs merged as described in Section 3.2.1 into a more structured instance: the jobs in the new transformed instance may be slightly smaller and of prespecified values, than their counterparts in instance $I^\#$.

Let us use $\mathcal{N} = \{J_j : p_j < \varepsilon^3\}$ to denote the set of *negligible* jobs. Then we have the following

Lemma 3.2 *With $1+5\varepsilon$ loss, we can assume that the following conditions hold:*

- $p_j = 0$, *for* $J_j \in \mathcal{N}$;
- $p_j = \varepsilon^3(1+\varepsilon)^{\pi_j}$, *where* $\pi_j \in \mathbb{N}$ *and for* $J_j \notin \mathcal{N}$.

Proof. Set $p_j := 0$, for every $J_j \in \mathcal{N}$. Replacing jobs by smaller ones cannot increase the objective function value. Clearly, we can translate the optimal solution of the transformed instance back by replacing the zero processing times of negligible jobs with the original processing times. This may potentially increase the optimal value of the transformed instance by at most $\sum_{J_j \in \mathcal{N}} p_j$ (here p_j denotes the original processing time value of job J_j). By Lemma 3.1, there are at most $4/\varepsilon^2$ negligible jobs and, therefore, $\sum_{J_j \in \mathcal{N}} p_j \leq \sum_{J_j \in \mathcal{N}} \varepsilon^3 \leq 4\varepsilon$.

By definition, every non negligible job J_j has processing times $p_j \geq \varepsilon^3$. Round each p_j down to the nearest lower value of $\varepsilon^3(1+\varepsilon)^h$, where $h \in \mathbb{N}$ and $J_j \notin \mathcal{N}$. When the rounded values are replaced with the original ones, the objective function value may potentially increase by at most a factor of $1+\varepsilon$. ■

By Lemma 3.1, the optimal value of the restricted problem may potentially increase up to $OPT(1+10\varepsilon)$. According to Lemma 3.2, we can transform the original instance by replacing jobs with smaller ones. Clearly, the latter can only decrease the objective function value (or leave it unchanged), but can never increase it. Therefore, with (3.1), we can assume that the optimal solution of the transformed instance always remains bounded by $4(1+10\varepsilon)$.

Now we define the *job profile* of a job J_j to be a 3-tuple (ρ, π, δ) such that $r_j = \rho\varepsilon$, $p_j = \varepsilon^3(1+\varepsilon)^\pi$ and $q_j = \delta\varepsilon$. We adopt the convention that $\pi = -\infty$ if $p_j = 0$.

Corollary 3.1 *The number of distinct job profiles is bounded by*

$$\tau := \left\lceil 12\varepsilon^{-2} \log_{1+\varepsilon}(2/\varepsilon) \right\rceil .$$

Proof. Since we are assuming that there are at most $2/\varepsilon$ distinct delivery times, as well as $2/\varepsilon$ release dates, the number of distinct job profiles can be bounded by $4/\varepsilon^2 \cdot x$, where x is the number of different π values in any job profile. By Lemma 3.2, any processing time p_j can be either 0 or $\varepsilon^3(1+\varepsilon)^h$, for some $h \in \mathbb{N}$. Since by Lemma 3.1 $p_j \leq \max\left\{1, 2\varepsilon^3\right\} < 2$, we have $\varepsilon^3(1+\varepsilon)^h < 2$ and therefore $h < \log_{1+\varepsilon}(2/\varepsilon^3)$. Thus, the number x of different π values in any job profile can be bounded by $x < 1 + 3\log_{1+\varepsilon}(2/\varepsilon)$. Since x must be integral, we have $x \leq \left\lceil 3\log_{1+\varepsilon}(2/\varepsilon) \right\rceil$ and the claim follows. ■

According to the previous corollary we can partition the set of jobs into $N_1, ..., N_\tau$ subsets of jobs having the same job profile. We use the following notation to represent the input: the input jobs are represented as a vector

$\mathbf{n} = (n_1, ..., n_\tau)$, where n_i denotes the number of jobs in N_i. Notice that $n \geq \sum_{k=1}^{\tau} n_k$.

The *assignment profile* of a machine is a vector $\mathbf{a} = (a_1, a_2, ..., a_\tau)$, with $\mathbf{a} \leq \mathbf{n}$, where a_i is the number of jobs from N_i assigned to that machine. Every vector $\mathbf{a} = (a_1, a_2, ..., a_\tau)$ can be interpreted as an instance of problem $1|r_j|L_{\max}$ having a_i jobs with release, delivery and processing times as the jobs from N_i, for $i = 1, ..., \tau$. The *length* of assignment $\mathbf{a}$, denoted $L(\mathbf{a})$, is the optimal value of the corresponding instance of problem $1|r_j|L_{\max}$. An assignment profile $\mathbf{a}$ is said to be *interesting* if $L(\mathbf{a}) \leq 4(1 + 10\varepsilon)$. We denote by A the set of all interesting assignment profiles. Observe that we only need to consider assignment profiles in A if we want to solve the current transformed instance to optimality. Furthermore, for every $\mathbf{a} \in A$, the number of jobs in the corresponding instance of $1|r_j|L_{\max}$ is always bounded by $\nu = \sum_{i=1}^{\tau} a_i \leq 4/\varepsilon^2 + 4(1 + 10\varepsilon)/\varepsilon^3 = O(1/\varepsilon^3)$, since there are at most $4/\varepsilon^2$ negligible jobs, and the remaining jobs have processing times greater or equal to ε^3. A simple calculation shows that $|A|$, the number of all interesting assignment profiles, is at most $\tau^\nu = 1/\varepsilon^{O(1/\varepsilon^3)}$, a constant in ε that does not depend on the input. This property is important, as our integer linear program will have $2|A| + 1$ integer variables.

Consider the set of instances of problem $1|r_j|L_{\max}$ corresponding to all interesting assignment profiles. We compute and memorize the optimal solutions of these instances, together with their values $L(\mathbf{a})$, in a look-up table S. Thus, table S stores the set of all *interesting machine schedules*, and it can be determined in constant time. Indeed, table S can be computed by solving a constant number of instances of problem $1|r_j|L_{\max}$. Furthermore, each instance has $O(1/\varepsilon^3)$ jobs, thus, it can be solved in $O(1/\varepsilon^3!) = 1/\varepsilon^{O(1/\varepsilon^3)}$ time. Therefore, the set of all interesting machine schedules can be computed in $1/\varepsilon^{O(1/\varepsilon^3)}$ time.

Once we have computed a schedule for each possible assignment, we can now pursue an approach similar to that used for problem $P||C_{\max}$ by Shmoys (see [32] p. 370) and by Alon, Azar, Woeginger and Yadid [1]. An optimal solution for the transformed instance of problem $P|r_j|L_{\max}$ is obtained by determining, for each vector $\mathbf{a} \in A$, the number $x_{\mathbf{a}}$ of machines with assignment profile $\mathbf{a}$. In these terms, we have a feasible solution $X = (x_1, ..., x_{|A|})$ to the problem if m machines are used and all jobs were assigned. We formulate this problem by using an integer linear program (ILP) formulation,

$$
\begin{array}{lll}
\min & z & \\
\text{s.t.} & \sum_{\mathbf{a}\in A} x_{\mathbf{a}} = m & \\
& \sum_{\mathbf{a}\in A} x_{\mathbf{a}} \cdot \mathbf{a} = \mathbf{n} & \\
& y_{\mathbf{a}} \leq x_{\mathbf{a}} \leq m \cdot y_{\mathbf{a}} & \forall \mathbf{a} \in A \\
& z \geq y_{\mathbf{a}} \cdot L(\mathbf{a}) & \forall \mathbf{a} \in A \\
& x_{\mathbf{a}} \in \{0, 1, ..., m\} & \forall \mathbf{a} \in A \\
& y_{\mathbf{a}} \in \{0, 1\} & \forall \mathbf{a} \in A
\end{array}
$$

The first constraint states that exactly m machines must be used. The second constraint ensures that all jobs can be scheduled. In the third set of constraints, the 0-1 variables $y_{\mathbf{a}}$ indicate whether the assignment $\mathbf{a}$ is used ($y_{\mathbf{a}} = 1$) or not ($y_{\mathbf{a}} = 0$). In the fourth set of constraints, in case the assignment $\mathbf{a}$ is used, then the term $y_{\mathbf{a}} \cdot L(\mathbf{a})$ contributes to the objective function.

The number of integer variables in (ILP) is $2|A| + 1$, and we have already observed that this is a constant that does not depend at all on the input size. We now apply Lenstra's algorithm [56] to solve (ILP). The time complexity of Lenstra's algorithm is exponential in the number of variables, but polynomial in the logarithms of the coefficients. The coefficients in (ILP) are at most $\max\{n, m, 4(1 + 10\varepsilon)\}$, and so (ILP) can be solved within an overall time complexity of $O(\log^{O(1)} n)$ [56]. Note that here the additive hidden constant depends exponentially on $1/\varepsilon$.

An optimal solution of the transformed instance is obtained by first assigning jobs according to the optimal solution of (ILP), and then for each machine assignment we schedule the corresponding jobs to optimality (these optimal schedules have been already computed and memorized in table S). By lemmas 3.1 and 3.2, the optimal solution of the transformed instance can be easily translated back to an approximate solution for the original instance. This can be accomplished in linear time. Furthermore, it is easy to check that the preprocessing steps require linear time, so the total running time of the algorithm is $O(n + f(1/\varepsilon))$, where $f(1/\varepsilon)$ is a constant for fixed ε. To summarize,

Theorem 3.1 *Problem $P|r_j|L_{\max}$ admits a polynomial time approximation scheme with linear running time.*

Corollary 3.2 *Problem $P|r_j|L_{\max}$ belongs to EPTAS.*

3.3 Controllable Processing Times

In this section we consider the identical parallel machine scheduling problem with controllable processing times. The problem can be stated as follows. We have a set $\mathcal{J}$ of n jobs, $J_1, ..., J_n$, and m identical machines, $M = \{1, ..., m\}$. Each job J_j must be processed without preemption on one of the m machines, each of which can process at most one job at a time. The processing time of job J_j lies in an interval $[\ell_j, u_j]$ (with $0 \leq \ell_j \leq u_j$). For each job J_j we have to choose a machine $m_j \in \{1, \ldots, m\}$, and $\delta_j \in [0, 1]$ and get then processing time and cost that depends linear on δ_j:

$$\begin{aligned} p_j(\delta_j) &= \delta_j \ell_j + (1 - \delta_j) u_j \\ c_j(\delta_j) &= \delta_j c_j^{\ell} + (1 - \delta_j) c_j^{u} \end{aligned}$$

The makespan of the solution is given by $max_{1 \leq i \leq m} \sum_{J_j \in \mathcal{J}, m_j = i} p_j(\delta_j)$ and the total cost by $\sum_{j \in J} c_j(\delta_j)$.

Nowicki and Zdrzalka [68] consider the preemptive scheduling of m identical parallel machines for jobs having processing costs which are linear functions of controllable processing times. They provide a $O(n^2)$ greedy algorithm which generates the set of Pareto-optimal points. A pair (σ, δ) is called Pareto-optimal if there does not exist another pair (σ', δ') that improves on (σ, δ) with respect to one of $C(\delta)$ and $T(\sigma, \delta)$, and stays equal or improves with respect to the other one. When preemption is not allowed and the machines are not identical, Trick [77] gave a polynomial-time 2.618-approximation algorithm to minimize a weighted sum of the cost and the makespan (P3). The latter result was improved by Shmoys and Tardos [76] by providing a polynomial-time 2-approximation algorithm. When processing times are fixed, Hochbaum and Shmoys [33] designed a PTAS for the identical parallel machine scheduling problem subject to makespan minimization.

In this section we settle the approximability of the described problems by obtaining the first polynomial time approximation schemes for P1, P2 and P3. We describe two new technical ideas.

Our first idea concerns the processing times of jobs. We show how to guess in polynomial time the processing times and costs by using a rounding technique for large and compact representation for small processing times. Using our guessing step and the algorithm described in [33] for fixed processing times, we obtain a PTAS for P2 which provides a solution with minimum cost and makespan at most $(1+\epsilon)\tau$, for any fixed $\varepsilon > 0$. Note that for problem P2, deciding if there is a solution with $T \leq \tau$ is already NP-complete,

therefore the best that we can expect, unless P=NP, is to find a solution with cost at most the optimal cost and makespan not greater than $\tau(1+\varepsilon)$. In Section 3.3.2, by using a PTAS for P2, we present a PTAS for P1 which delivers a solution with cost $C \leq \kappa$ and makespan at most $(1+\epsilon)$ times greater then the minimum makespan subject to $C \leq \kappa$. In Section 3.3.3 a PTAS for P3 is proposed.

Our second idea shows that we can exploit continuous knapsack problems [50] to get "good" lower and upper bounds for our optimization problems and to get fast optimal and approximative algorithms. Based on a modified version of list scheduling algorithm and a continuous relaxation of the knapsack problem we obtain a fast 2-approximation algorithm for problems P2 and P1. In Section 3.3.4 we propose a linear time algorithm to solve optimally problem P2 when preemption is allowed. This algorithm is again based on a continuous knapsack problem and McNaughton's rule [63]. By using a similar approach we obtain exact and approximate solutions for the other two preemptive variants P1 and P3.

Remark 3.1 *If $c_j^\ell \leq c_j^u$, then the case with $\delta_j = 1$ (with processing time equal to ℓ_j) dominates the other choices. Furthermore, in each solution we can replace the processing time and cost of such a job by $p_j(\delta_j = 1)$ and $c_j(\delta_j = 1)$, respectively, without having a larger makespan or higher total cost. In that case, we revise the c_j^u values, when $c_j^\ell \leq c_j^u$, by setting $c_j^u := c_j^\ell$. Therefore, in the following, we will assume, without loss of generality, that it is always $c_j^\ell \geq c_j^u$. Furthermore, if $c_j^u = c_j^\ell$ then, without loss of generality, we set $u_j := \ell_j$, for every job J_j.*

3.3.1 A PTAS for P2

Our approximation scheme conceptually has the following two steps.

1. Guessing processing times: Identify processing times and costs for all jobs such that there is a feasible schedule with minimum cost and makespan at most $\tau(1+\varepsilon)$.

2. Scheduling jobs: when processing times and costs for all jobs are fixed, the problem turns out to be the classical identical parallel machine scheduling problem subject to makespan minimization. The latter can be solved approximately with any given accuracy in polynomial time [33].

Therefore, the first step of guessing the processing times immediately gives a PTAS for P2.

Without loss of generality, we may assume that the makespan must be at most 1 by dividing all processing times by τ. Furthermore, we may assume that $u_j \leq 1$ for all jobs $j \in J$; otherwise we can define $u_j = 1$ and adjust the value of c_j^u to get an equivalent instance.

Guessing Step Our approach is based on dynamic programming. One way to make the computation more efficient is to reduce the number of distinct processing times that may occur in any feasible solution. We begin by transforming the given instance into a more structured one in which every job can assume at most $O(\frac{1}{\varepsilon}\log\frac{n}{\varepsilon})$ distinct processing times which are multiple of ε/n. This is accomplished as follows.

1. Divide interval $[0,1]$ into subintervals of length ε/n.
2. Let S be the set of values obtained by rounding the following values $\frac{\varepsilon}{n}, \frac{\varepsilon}{n}(1+\varepsilon), \frac{\varepsilon}{n}(1+\varepsilon)^2, ..., 1$ up to the nearest value among $\{\varepsilon/n, 2\varepsilon/n, ..., 1\}$.
3. For each job J_j consider as possible processing times the set S_j of values from S that fall in interval $[\ell_j, u_j]$.

Lemma 3.3 *Given an instance I, we can obtain in polynomial time another instance I' such that*

- *Every possible processing time is equal to $k(\varepsilon/n)$ for some $k \in \{1, ..., n/\varepsilon\}$, and there are $O(\frac{1}{\varepsilon}\log\frac{n}{\varepsilon})$ different processing times.*
- *If C^* is the minimum cost for I when the makespan is at most 1 then for I' there is a solution with cost and makespan at most C^* and $1+2\varepsilon$, respectively.*

Proof. Let b the cardinality of S, then $\frac{\varepsilon}{n}(1+\varepsilon)^{b-2} < 1$ (and $\frac{\varepsilon}{n}(1+\varepsilon)^{b-1} \geq 1$) and therefore $b-2 < \frac{\log_2 \frac{n}{\varepsilon}}{\log_2(1+\varepsilon)} \leq \frac{1}{\varepsilon}\log_2\frac{n}{\varepsilon}$, for every $\varepsilon \leq 1$. We prove that our transformation may potentially increase the makespan value by a factor of $1+2\varepsilon$ while the minimum cost is lower or the same. Indeed, let $p_j(\delta_j)$ be any feasible processing time for any given job J_j in instance I. It is easy to see that in instance I' there exists a corresponding value $\tilde{p}_j(\tilde{\delta}_j)$ belonging to the restricted set S_j such that $\tilde{p}_j(\tilde{\delta}_j) \leq p_j(\delta_j)(1+\varepsilon) + \frac{\varepsilon}{n}$. Now, let H be a set of jobs on one machine with $\sum_{J_j \in H} p_j(\delta_j^*) \leq 1$ in some optimal solution and let δ_j^* be the corresponding δ_j-values, for $J_j \in H$. Then, if we replace each $p_j(\delta_j^*)$ with the corresponding value $\tilde{p}_j(\tilde{\delta}_j^*)$ in instance I'

we obtain a solution with cost at most the optimal cost and makespan $\sum_{J_j \in H} \tilde{p}_j(\tilde{\delta}_j^*) \leq \sum_{J_j \in H} (p_j(\delta_j^*)(1+\varepsilon) + \frac{\varepsilon}{n}) \leq 1 + 2\varepsilon$. ■

The above lemma allows us to work with instances with $O(\frac{1}{\varepsilon} \log \frac{n}{\varepsilon})$ distinct processing times and costs. A naive way of guessing the job processing times requires $O(n^{O(\frac{1}{\varepsilon} \log \frac{n}{\varepsilon})})$ time. In the following we show that we can exploit the restricted set of processing times to perform the guessing step in polynomial time.

We call J_j a *big* job if the processing time of J_j is greater than ε, and a *small* job otherwise. Clearly, we have the following different cases:

(1) $\ell_j \leq \varepsilon < u_j$. In this case J_j can be small or big.

(2) $\ell_j \leq u_j \leq \varepsilon$. In this case J_j is small.

(3) $\varepsilon < \ell_j \leq u_j$. In this case J_j is big.

By our transformation, big jobs have $\beta \leq \lceil \frac{1}{\varepsilon} \log \frac{1}{\varepsilon} \rceil + 1$ different processing times. Let us denote these β values by $p_1, p_2, ..., p_\beta$. Let x_i denote the total number of big jobs with processing times equal to p_i, for $i = 1, ..., \beta$. A configuration of the processing times for big jobs can be given by a vector $(x_1, x_2, ..., x_\beta)$. Let us call $(x_1, x_2, ..., x_\beta)$ a *β-configuration*. Since there are m machines for which the maximum completion time must be at most $(1+2\varepsilon)$ by Lemma 3.3, a β-configuration is said to be feasible if $\sum_{i=1}^{\beta} x_i p_i \leq m(1+2\varepsilon)$. Since $p_i > \varepsilon$, for $i = 1, ..., \beta$, a necessary condition of feasibility for any β-configuration is that $\sum_{i=1}^{\beta} x_i \leq m(1+2\varepsilon)/\varepsilon$. Therefore, the number h of feasible β-configurations is at most the number of β-tuples of non-negative integers with the sum at most $m(1+2\varepsilon)/\varepsilon$. The number $h(d)$ of β-tuples with the sum equals to d is $h(d) = \binom{d+\beta-1}{\beta-1}$. Therefore, $h \leq \sum_{d=0}^{\frac{m(1+2\varepsilon)}{\varepsilon}} \binom{d+\beta-1}{\beta-1} = \binom{\frac{m(1+2\varepsilon)}{\varepsilon}+\beta}{\beta}$. Since $\binom{x}{y} \leq (\frac{ex}{y})^y$ and $\varepsilon \leq 1$, we see that $h \leq \binom{\frac{3m}{\varepsilon}+\beta}{\beta} \leq \binom{\frac{3m}{\varepsilon}+\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}+2}{\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}+2} \leq \binom{(3m+1)(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}+2)}{\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}+2} \leq (4em)^{\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}+2}$.

A *feasible configuration* $\mathbf{c} = (x_1, x_2, ..., x_\beta, x_{\beta+1})$ is a $(\beta+1)$-dimensional vector where $(x_1, x_2, ..., x_\beta)$ is a feasible β-configuration and

$$x_{\beta+1} \in \{0, \varepsilon/n, 2\varepsilon/n, \ldots, \varepsilon n\}$$

represents the sum of the chosen processing times for small jobs. The number of different configurations is bounded by $\gamma \leq h(n^2+1)$.

To simplify the formulation of the dynamic program we use a vector representation of the reduced set S_j of possible processing times for job J_j. V_j is the set of $(\beta+1)$-dimensional vectors defined as follows: for each $p \in S_j$

with $p = p_i$, for some $i = 1, ..., \beta$, there is a vector $\mathbf{v} \in V_j$ such that the i-th entry of $\mathbf{v}$ is set to 1, and the other entries are zeros; otherwise for each small processing time p then the $(\beta + 1)$-st entry of $\mathbf{v}$ is p and the other entries of $\mathbf{v}$ are set to zero. Let $c(\mathbf{v})$ denote the cost of $\mathbf{v}$.

For every feasible configuration $\mathbf{c}$ and for every j, $1 \leq j \leq n$, we denote by $COST(j, \mathbf{c})$ the minimum total cost when jobs $J_1, ..., J_j$ have been assigned processing times according to configuration $\mathbf{c}$; in case no such assignment exists, $COST(j, \mathbf{c}) = +\infty$. It is easy to see that the following recurrence holds:

$$\begin{aligned} COST(1, \mathbf{v}) &= \begin{cases} c(\mathbf{v}) & \text{if } \mathbf{v} \in V_1 \\ +\infty & \text{if } \mathbf{v} \notin V_1 \end{cases} \\ COST(j, \mathbf{c}) &= \min_{\mathbf{v} \in V_j:\ \mathbf{v} \leq \mathbf{c}} \{c(\mathbf{v}) + COST(j-1, \mathbf{c} - \mathbf{v})\} \quad \text{for } j = 2, ..., n \end{aligned}$$

For each job we have to consider at most $O(\gamma \cdot \frac{1}{\varepsilon} \log \frac{n}{\varepsilon})$ combinations. Therefore, the running time of the dynamic program is $O(n^3 \log n \cdot m^{O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})})$.

Now we show that among all the generated assignments of processing times to jobs there exists an assignment $\tilde{A}$ with minimum cost and having a schedule with makespan at most $1 + O(\varepsilon)$.

Consider the transformed instance I' (see Lemma 3.3) and let A^* denote the assignment of processing times to jobs according to some optimal solution S^* for I'. Since we consider all the different feasible configurations, one of them must be the configuration corresponding to A^*. Let us call $\mathbf{c}^*$ this configuration. Let $\tilde{A}$ denote the assignment of processing times to jobs that respects $\mathbf{c}^*$ and that has been generated by our dynamic program. Let L^* and $\tilde{L}$ be the sum of all processing times according to A^* and $\tilde{A}$, respectively. The total cost of $\tilde{A}$ is not greater than the cost of A^*. Furthermore, for each big job of A^* there is a big job of $\tilde{A}$ having the same processing time, and vice versa. Finally, the sum of small job processing times in A^* has the same value as in $\tilde{A}$. It follows that $L^* = \tilde{L}$. Now, we show that the optimal solution value that respects $\tilde{A}$ is at most $1 + O(\varepsilon)$. Indeed, consider the solution obtained by scheduling first the big jobs as in S^* and then by adding greedily the small jobs as in list-schedule algorithm [26]. If the machine with the largest completion time finishes with a big job, then we get an optimal solution for I' with makespan at most $1 + 2\varepsilon$ (see Lemma 3.3). Otherwise, let p_k be the processing time of a small job J_k that ends last. Then the makespan is equal to $s_k + p_k$, where s_k is the starting time of job J_k. We see that $s_k + p_k < \frac{\tilde{L}}{m} + p_k = \frac{L^*}{m} + p_k \leq 1 + 3\varepsilon$.

Theorem 3.2 *There is a PTAS for P2 that provides a solution with minimum cost and makespan at most* $(1+\epsilon)\tau$*, if a solution with makespan not greater than* τ *exists.*

3.3.2 A PTAS for P1

Assume that a solution with cost at most κ exists. A PTAS for P1 is obtained by the following observations. If we know a makespan value $\tau \in [T^*, T^*(1+\varepsilon)]$, where T^* is the optimal makespan, by solving approximately the problem of minimizing the cost function subject to makespan at most τ (see Section 3.3.1), then we get a solution with makespan at most $\tau(1+\varepsilon) \leq T^*(1+O(\varepsilon))$ and cost $\leq \kappa$. The algorithm of Section 3.3.1 can be used to determine whether a schedule with makespan at most τ does not exist, or else produce a solution with makespan at most $\tau(1+\varepsilon)$. Since $\tau \in [0, nu_{\max}]$, where $u_{\max} = \max_j u_j$, in order to find a solution with makespan at most $T^*(1+O(\varepsilon))$ the algorithm of Section 3.3.1 must be applied $O(\log nu_{\max})$ times. Therefore, the resulting algorithm is polynomial in the binary encoding of the input size. Clearly, an upper bound on T^* better than $nu_{\max}$ helps us to improve the efficiency of the computation.

In the next sections we provide an approximation algorithm that delivers a solution with makespan $t \leq 2T^*$ and cost at most κ in $O(n \ln m + m \ln^2 m)$ time. A value $\tau \in [T^*, T^*(1+\varepsilon)]$ can be found by applying a binary search only on the following $\lceil \frac{1}{\varepsilon} \log \frac{2}{\varepsilon} \rceil$ values, $\frac{t}{2}(1+\varepsilon), \frac{t}{2}(1+\varepsilon)^2, ..., t$. Therefore, by executing the algorithm of Section 3.3.1 $O(\log(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}))$ times we get a PTAS for P1.

Theorem 3.3 *There is a PTAS for P1 that provides a solution with makespan at most* $(1+\epsilon)$ *times the minimum makespan and cost at most* κ*, if a solution with cost not greater than* κ *exists.*

Bounds

We begin by computing lower (LB) and upper (UB) bounds for the optimum makespan value τ. Consider some optimal solution S^* and let $\delta_1^*, ..., \delta_n^*$ denote the corresponding δ_j-values. Let us write the corresponding cost value as $C(\delta_j^*) = \sum_{j=1}^n c_j \delta_j^* + \sum_{j=1}^n c_j^u$, where $c_j = c_j^\ell - c_j^u$. Let $P^* = \sum_{j=1}^n p_j(\delta_j^*)$ be the sum of processing times in S^*. Consider any n values

$\tilde{\delta}_1, ..., \tilde{\delta}_n$, with $\tilde{\delta}_j \in [0,1]$ for $j = 1, ..., n$, such that

$$\tilde{P} = \sum_{j=1}^{n} p_j(\tilde{\delta}_j) \leq P^*, \tag{3.4}$$

$$\sum_{j=1}^{n} c_j \tilde{\delta}_j \leq \kappa - \sum_{j=1}^{n} c_j^u. \tag{3.5}$$

Note that $\sum_{j=1}^{n} c_j^u$ is a constant and $\kappa - \sum_{j=1}^{n} c_j^u$ is non-negative, otherwise no solution with cost at most κ exists. Furthermore $\delta_1^*, ..., \delta_n^*$ satisfy constraints (3.4) and (3.5). If $\kappa - \sum_{j=1}^{n} c_j^u = 0$ then the processing time of each job J_j must be u_j in any feasible solution with cost at most κ, and of course we obtain valid bounds by setting $LB = P^*/m$ and $UB = P^*$. Otherwise, if $\kappa - \sum_{j=1}^{n} c_j^u > 0$, we simplify the problem instance by dividing all processing costs by $\kappa - \sum_{j=1}^{n} c_j^u$.

Let x_j $(1 \leq j \leq n)$ be a non-negative variable. We define vector $\tilde{\delta}$ by setting $\tilde{\delta}_j = x_j^*$, where $x^* = (x_1^*, ..., x_n^*)$ is the optimal solution of the following linear program.

$$\text{minimize} \quad P = \sum_{j=1}^{n} [x_j \ell_j + (1 - x_j) u_j] \tag{3.6}$$

$$\text{subject to} \quad \sum_{j=1}^{n} \tilde{c}_j x_j \leq 1 \tag{3.7}$$

$$0 \leq x_j \leq 1 \qquad j = 1, ..., n, \tag{3.8}$$

where $\tilde{c}_j = c_j/(\kappa - \sum_{j=1}^{n} c_j^u)$. Note that the optimal solution of the linear program (3.6), (3.7) and (3.8), defines a vector $\tilde{\delta}$ that satisfies constraints (3.4) and (3.5). The objective function can be written as $P = \sum_{j=1}^{n} [(\ell_j - u_j)x_j] + \sum_{j=1}^{n} u_j$, where the second summation is a constant. Let $t_j = u_j - \ell_j$. An optimal solution of the previous linear program is optimal also for the following linear program,

$$\text{maximize} \quad \sum_{j=1}^{n} t_j x_j \tag{3.9}$$

$$\text{subject to} \quad \sum_{j=1}^{n} \tilde{c}_j x_j \leq 1 \tag{3.10}$$

$$0 \leq x_j \leq 1, \quad j = 1, ..., n \tag{3.11}$$

The previous LP is a continuous relaxation of the classical knapsack problem that can be solved as follows [50]: first, sort the jobs in non-increasing $t_j/\tilde{c}_j$ ratio order, so that, without loss of generality,

$$\frac{t_1}{\tilde{c}_1} \geq \frac{t_2}{\tilde{c}_2} \geq ... \geq \frac{t_n}{\tilde{c}_n}.$$

Then assign $x_j^* = 1$ until either (a) the jobs are exhausted or (b) the cost capacity 1 is exactly used up or (c) it is necessary to set x_j^* to a value that is less than 1 to use up the cost capacity 1 exactly. Let s the last job that uses up the cost capacity. Set $x_j^* = 0$, when $j > s$. In all three cases an optimal solution is obtained. The optimal solution of (3.6), (3.7) and (3.8) can be carried out in $O(n)$ time by partial sorting [50].

If we place all jobs on one machine with processing times according to the optimal solution of (3.6), (3.7) and (3.8), then the makespan of the schedule is equal to $\tilde{P}$. Furthermore, we obtain a valid lower bound by setting $LB = \tilde{P}/m$, since $\tilde{P} \leq P^*$. Therefore, $\tau \in [\tilde{P}/m, \tilde{P}]$. Dividing all processing times by $\tilde{P}$, we have $\tau \in [1/m, 1]$.

A 2-approximation Algorithm for P1

In the following we describe an algorithm that given $\tau \in [1/m, 1]$ and κ, either decides that there is no schedule with makespan at most τ and cost $\leq \kappa$, or produces a schedule with makespan at most $(2 - \frac{2}{m+1})\tau$ and cost $\leq \kappa$. Therefore a value $\tau \in [T^*, 2T^*]$ can be found by binary search on the following set of $\lceil m \log m \rceil$ values, $V = \{\frac{1}{m}(1 + \frac{1}{m}), \frac{1}{m}(1 + \frac{1}{m})^2, ..., 1\}$.

The algorithm works as follows. Since we are interested in some solution with makespan at most τ, transform the given instance into an equivalent one so that, for all jobs $J_j \in \mathcal{J}$, we have $u_j \leq \tau$. Then, as described in the previous section, solve the continuous relaxation of the classical knapsack problem and determine the vector $\tilde{\delta} = (\tilde{\delta}_1, ..., \tilde{\delta}_n)$ that minimize the sum $\tilde{P}$ of processing times. If $\tilde{P} > m\tau$ then there is no solution with makespan at most τ and cost κ. Otherwise, according to $\tilde{\delta}$ our algorithm works as follows.

Algorithm A

1. Assign the m jobs with the largest processing times to different machines. Let $k = 1$.

2. Schedule the remaining jobs as follows. Assign each job (one after another in any arbitrary order) to machine k until either (a) the jobs

are exhausted or (b) the completion time of the last added job $J_{f(k)}$ is greater than τ.

3. If the jobs are not exhausted, then let $k := k + 1$. If $k \leq m$ go to step 2, else go to step 4.

4. Remove jobs $J_{f(1)}, ..., J_{f(m-1)}$ from the schedule and assign them one after another to a machine with the currently smallest load.

Since $\tilde{P} \leq m\tau$ all jobs are scheduled by the algorithm above. The algorithm runs in $O(n + m \log m)$ time.

Theorem 3.4 *If $\tilde{P} \leq m\tau$ then algorithm A provides a solution with makespan $T^A \leq (2 - \frac{2}{m+1})\tau$, where processing times are chosen according $\tilde{\delta}$.*

Proof. We assume that $T^A > \tau$ otherwise we are done. Let J_f be a job with the largest completion time and let s_f be the start time of job J_f. Then no machine can be idle at any time prior to s_f, since otherwise job J_f would have been processed there. Two situations are possible:

1. J_f is one of the m biggest jobs. In that case the schedule is optimal since $p_f(\tilde{\delta}_f) \leq \tau$.

2. Otherwise, assume, without loss of generality, that $p_1(\tilde{\delta}_1) \geq p_2(\tilde{\delta}_2) \geq ... \geq p_n(\tilde{\delta}_n)$. The following inequalities hold,

$$\begin{aligned} p_f(\tilde{\delta}_f) &\leq \min_{j=1,...,m} p_i(\tilde{\delta}_i), \\ p_f(\tilde{\delta}_f) &\leq m\tau - \sum_{i=1}^{m} p_i(\tilde{\delta}_i), \\ p_f(\tilde{\delta}_f) &\leq \frac{1}{m}\sum_{i=1}^{m} p_i(\tilde{\delta}_i). \end{aligned}$$

It follows that $p_f(\tilde{\delta}_f) \leq \frac{1}{m}\sum_{i=1}^{m} p_i(\tilde{\delta}_i) \leq \frac{1}{m}(m\tau - p_f(\tilde{\delta}_f))$, hence $p_f(\tilde{\delta}_f) \leq \frac{m\tau}{m+1}$. Since $\tilde{P} = \sum_i p_i(\tilde{\delta}_i) \leq m\tau$, then

$$\begin{aligned} T^A &= s_f + p_f(\tilde{\delta}_f) \leq \frac{1}{m}\sum_{i \neq f} p_i(\tilde{\delta}_i) + p_f(\tilde{\delta}_f) \\ &\leq \tau + (1 - 1/m)p_f(\tilde{\delta}_f) \\ &\leq \tau + (1 - 1/m)\frac{m\tau}{m+1} = \tau + \frac{m-1}{m+1}\tau = (2 - \frac{2}{m+1})\tau. \end{aligned}$$

■

3.3.3 A PTAS for P3

Minimization of $T + \alpha C$, $(\alpha > 0)$ where T is the makespan and C are the total cost of the schedule. Using modified cost values $c'_j = \alpha c_j$, we can restrict us to the case $\alpha = 1$.

The first step is to obtain some lower bounds for the minimum objective value OPT. We define $d_j = min[c_j^\ell + \ell_j, c_j^u + u_j]$, $D = \sum_{j \in J} d_j$. Consider an optimum schedule with makespan T and total cost C (where $OPT = T + C$). In this schedule we have δ_j^* values with $c_j(\delta_j^*) + p_j(\delta_j^*) \geq d_j$. Then,

$$D = \sum_{j \in J} d_j \leq \sum_{j \in J} c_j(\delta_j^*) + \sum_{j \in J} p_j(\delta_j^*) \leq C + mT \leq mOPT.$$

On the other hand, we can generate a feasible schedule accordingly to the d_j values. Indeed, accordingly to d_j, we choose as processing time and cost either (ℓ_j, c_j^ℓ) or (u_j, c_j^u) (where $\delta_j = 1$ or $\delta_j = 0$, respectively). Since $c_j(\delta_j) \leq d_j$, the total cost is $\sum_{j \in J} c_j(\delta_j) \leq \sum_{j \in J} d_j = D$. If we place all jobs on one machine, then the makespan of the schedule is equal to $\sum_{j \in J} p_j(\delta_j) \leq D$. Since $c_j(\delta_j) + p_j(\delta_j) = d_j$, the objective value of this schedule is exactly D. Therefore, $OPT \leq D \leq mOPT$. This implies that $OPT \in [D/m, D]$. Dividing all processing times and cost values by D, we have that $OPT \in [1/m, 1]$.

Again, a makespan that is less or equal to $T + OPT \cdot \varepsilon$ can be found among the following $O(\frac{1}{\varepsilon} \log \frac{m}{\varepsilon})$ values: $\frac{\varepsilon}{m}, \frac{\varepsilon}{m}(1+\varepsilon), \frac{\varepsilon}{m}(1+\varepsilon)^2, ..., 1$. To see this we use a case analysis with $T \leq \varepsilon/m$ and $T > \varepsilon/m$. Once we have a value that is at most $T + OPT \cdot \varepsilon$ by applying the algorithm provided in Section 3.3.1 we get a solution with cost at most C and makespan at most $(1+\varepsilon)(T + OPT \cdot \varepsilon)$. The objective value of this solution is bounded by $OPT(1+3\varepsilon)$. The number of calls of the algorithm provided in Section 3.3.1 is $O(\log m)$.

3.3.4 Preemptive Problems

The preemptive problems are easy to solve. For instance, an optimal solution for problem P2 can be delivered as follows. First, solve the following linear program which defines the cost and the processing times of jobs.

$$\begin{aligned}
\text{Minimize } & \sum_{j=1}^{n}\sum_{i=1}^{m}(c_j^u x_{ij}^u + c_j^l x_{ij}^l) \\
\text{s. t. } & \sum_{i=1}^{m}(x_{ij}^u + x_{ij}^l) = 1 \quad j = 1, ..., n \\
& \sum_{j=1}^{n}(u_j x_{ij}^u + l_j x_{ij}^l) \leq \tau \quad i = 1, ..., m \\
& \sum_{i=1}^{m}(u_j x_{ij}^u + l_j x_{ij}^l) \leq \tau \quad j = 1, ..., n \\
& x_{ij}^u, x_{ij}^l \geq 0 \quad j = 1, ..., n \quad i = 1, ..., m
\end{aligned}$$

Let $\delta_j = \sum_{i=1}^{m} x_{ij}^u$, $j = 1, ..., n$. It is easy to see that a lower bound of the length of a preemptive schedule cannot be smaller than

$$LB = \max\{\max_j p_j(\delta_j), \frac{1}{m}\sum_{j=1}^{n} p_j(\delta_j)\}.$$

A schedule meeting this bound can be constructed in $O(n)$ time [63]: fill the machines successively, scheduling the jobs in any order and splitting jobs into two parts whenever the above time bound is met. Schedule the second part of a preempted job on the next machine at zero time. Due to the fact that $p_j(\delta_j) \leq LB$ for all j, the two parts of preempted job do not overlap. Similar arguments hold for P1 and P3.

The most time-consuming part of the previous algorithms is the solution of the linear program. As pointed out in [76] these linear programs fall into the class of fractional packing problems, and therefore a solution with cost and makespan at most $(1 + \varepsilon)$ times the required values can be found by a randomized algorithm in $O(n^2 \log n)$ expected time, or deterministically, in $O(mn^2 \log n)$ time. In the following we present an alternative approach that finds the optimal solution for P2 in $O(n)$ time. Regarding P1 and P3, we describe an algorithm that finds solutions with only makespan increased by a factor $(1 + \varepsilon)$ and in $O(n \log \log m)$ time.

Preemptive P2

In Section 3.3.2, we described a linear program which find processing times of jobs such that the sum of processing times is minimized and the cost

is at most κ. Similarly, we can formulate a linear program which define jobs processing times by determining the vector $(\tilde{\delta}_j)$ such that the total cost is minimized, the sum of processing times is at most τm and each processing time is at most τ. Therefore, by construction, we have $\max\{\max_j p_j(\tilde{\delta}_j), \frac{1}{m}\sum_{j=1}^{n} p_j(\tilde{\delta}_j)\} \leq \tau$, and again, a schedule meeting this bound can be constructed in $O(n)$ time [63]. Before defining the linear program, since it does not make sense to consider for any operation a processing time larger than τ, it is possible to build a new instance by setting the upper bound of the processing times to $\min\{\tau, u_j\}$ for each job J_j. Of course we have to adjust the value of c_j^u to reflect this change. The linear program is the following.

$$\begin{aligned} \text{Minimize} \quad & \sum_{j=1}^{n}(c_j^\ell - c_j^u)\tilde{\delta}_j + \sum_{j=1}^{n} c_j^u \\ \text{s. t.} \quad & \sum_{j=1}^{n}(u_j - \ell_j)\tilde{\delta}_j \geq \sum_{j=1}^{n} u_j - \tau m \\ & 0 \leq \tilde{\delta}_j \leq 1 \qquad j = 1, ..., n. \end{aligned}$$

The previous LP is a continuous relaxation of the classical knapsack problem in minimization form. The minimization form of the problem can easily be transformed into an equivalent maximization form and solved as described in Section 3.3.2. Therefore, for P2 we can find the optimal solution in $O(n)$ time.

Preemptive P1

Assume that there is a schedule with makespan and cost equal to T and κ, respectively. Since it does not make sense to consider for any operation a processing time larger than T, it is possible to build a new instance by setting the upper bound of the processing times to $\min\{1, u_j\}$ for each job J_j. Of course we have to adjust the value of c_j^u to reflect this change. Let $KP(T, \kappa)$ denote the linear program (3.6), (3.7) and (3.8), defined in Section 3.3.2 when makespan and cost are assumed to be at most T and κ, respectively, therefore after the preprocessing step described before. If $KP(T, \kappa)$ is infeasible then there is no solution with cost at most κ. Otherwise, let $(\tilde{\delta}_j)$ denote the optimal solution of $KP(T, \kappa)$. If $\frac{1}{m}\sum_{j=1}^{n} p_j(\tilde{\delta}_j) > T$ then there is no solution with makespan at most T subject to cost at most κ, respectively. Therefore, if such a solution exists, then $\max\{\max_j p_j(\tilde{\delta}_j), \frac{1}{m}\sum_{j=1}^{n} p_j(\tilde{\delta}_j)\} \leq T$, and

again, a schedule meeting this bound can be constructed in $O(n)$ time [63]. From Section 3.3.2, we can guess the minimum value τ of the makespan subject to $C \leq \kappa$, by solving $KP(T, \kappa)$ with at most $O(\log \tilde{P})$ different values of T, since $\tau \in [\tilde{P}/m, \tilde{P}]$. As in Section 3.3.2, we can find a makespan that is at most $\tau(1+\varepsilon)$ by calling $KP(T, \kappa)$ with at most $O(\log \log m)$ different values of T.

Preemptive P3

To solve P3 first guess the value of the makespan (as described in section 3.3.3) and then solve a LP similar to the one provided for the preemptive P2 that finds a processing time assignment with minimum cost and makespan at most equal to the guessed value. Therefore, we can obtain a schedule with objective function value bounded by $C + T(1+\varepsilon) \leq (C+T)(1+\varepsilon)$.

Chapter 4

Scheduling on Unrelated Machines

4.1 Introduction

In this chapter we deal with the problem of scheduling a set $\mathcal{J} = \{J_1, ..., J_n\}$ of n independent jobs on a set $M = \{1, ..., m\}$ of m unrelated parallel machines. Each machine can process at most one job at a time, and each job has to be processed without interruption by exactly one machine. Processing job J_j on machine i requires $p_{ij} \geq 0$ time units when processed by machine i and incurs a cost $c_{ij} \geq 0$, $i = 1, \ldots, m$, $j = 1, ..., n$. The *makespan* is the maximum job completion time among all jobs. We consider the problem of minimizing the objective function that is a weighted sum of the makespan and total cost.

When $c_{ij} = 0$ the problem turns into the classical makespan minimization form. Lenstra, Shmoys and Tardos [54] gave a polynomial-time 2-approximation algorithm for this problem; and this is the currently known best approximation ratio achieved in polynomial time. They also proved that for any positive $\epsilon < 1/2$, no polynomial-time $(1 + \epsilon)$-approximation algorithm exists, unless P=NP. Furthermore, Shmoys and Tardos [76] gave a polynomial-time 2-approximation algorithm for the general variant with cost. Since the problem is NP-complete even for $m = 2$, it is natural to ask how well the optimum can be approximated when there is only a constant number of machines.

In contrast to the previously mentioned inapproximability result for the general case, there exists a fully polynomial-time approximation scheme for the problem when m is fixed. Horowitz & Sahni [36] proved that for any

$\varepsilon > 0$, an ε-approximate solution can be computed in $O(nm(nm/\varepsilon)^{m-1})$ time, which is polynomial in both n and $1/\varepsilon$ if m is constant. Lenstra, Shmoys and Tardos [54] also gave an approximation scheme for the problem with running time bounded by the product of $(n+1)^{m/\epsilon}$ and a polynomial of the input size. Even though for fixed m their algorithm is not fully polynomial, it has a much smaller space complexity than the one in [36].

Recently, Jansen & Porkolab [42] presented a fully polynomial-time approximation scheme for the problem whose running time is $n(m/\varepsilon)^{O(m)}$. They combine the previous (dynamic [36] and linear [54] programming) approaches. Their algorithm has to solve at least $O((m^3/\varepsilon^2)^m)$ many linear programs. In order to obtain a linear running time for the case when m is fixed, they use the price-directive decomposition method proposed by Grigoriadis and Khachiyan [28] for computing approximate solutions of block structured convex programs. The final ingredient is an intricate rounding technique based on the solution of a linear program and a partition of the job set.

In contrast to the previous approach [42], the algorithm proposed in this chapter (that works also for the general variant with cost) is extremely simple and even faster: first we preprocess the data to obtain a smaller instance with at most $(\log m/\varepsilon)^{O(m)}$ grouped jobs. The preprocessing step requires linear time. Then using dynamic programming we compute an approximate solution for the grouped jobs in $(\log m/\varepsilon)^{O(m^2)}$ time. Both steps together imply a fully polynomial-time approximation scheme that runs in $O(n) + C$ time where $C = (\log m/\varepsilon)^{O(m^2)}$. We remark that the multiplicative constant hidden in the $O(n)$ running time of our algorithm is reasonably small and does not depend on the accuracy ε.

4.2 An Improved FPTAS

For simplicity, let $0 < \varepsilon < 1$ be an arbitrary small rational number, and let $m \geq 2$ be an integral value. Throughout this section, the values ε and m are considered to be constants and not part of the input.

The problem can be stated by using the following integer linear program ILP that represents the problem of assigning jobs to machines ($x_{ij} = 1$ means that job J_j has been assigned to machine i, and μ is any given positive

weight):

$$\begin{array}{lll} \min & T + \mu \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} c_{ij} & \\ s.t. & \sum_{j=1}^{n} x_{ij} p_{ij} \leq T, & i = 1, \ldots, m; \\ & \sum_{i=1}^{m} x_{ij} = 1, & j = 1, ..., n; \\ & x_{ij} \in \{0, 1\}, \quad i = 1, \ldots, m, & j = 1, ..., n. \end{array}$$

The first set of constraints relates the makespan T to the processing time on each of the machines, while the second set ensures that every job gets assigned.

We begin by computing some lower and upper bounds for the minimum objective value OPT. By multiplying each cost value by μ we may assume, without loss of generality, that $\mu = 1$. Let $d_j = \min_{i=1,...,m}(p_{ij} + c_{ij})$, and $D = \sum_{j=1}^{n} d_j$. Consider an optimum assignment (x_{ij}^*) of jobs to machines with makespan T^* and total cost C^*. Then, $D = \sum_{j=1}^{n} d_j \leq \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij}^* c_{ij} + \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij}^* p_{ij} \leq C^* + m \cdot T^* \leq m \cdot OPT$, where $OPT = T^* + C^*$. On the other hand, we can generate a feasible schedule according to the d_j values. Indeed, let $m_j \in \{1, ..., m\}$ denote any machine such that $d_j = p_{m_j,j} + c_{m_j,j}$. Assign every job J_j to machine m_j. The objective value of this schedule can be bounded by $\sum_{j \in J} c_{m_j,j} + \sum_{j \in J} p_{m_j,j} = D$. Therefore, $OPT \in [D/m, D]$, and by dividing all processing times and cost values by D/m, we get directly the following bounds for the optimum value:

$$1 \leq OPT \leq m.$$

4.2.1 Fast, Slow, Cheap and Expensive Machines

For each job J_j, we define four sets of machines: the set of *fast* machines

$$\mathcal{F}_j = \left\{ i : p_{ij} \leq \frac{\varepsilon}{m} d_j \right\},$$

the set of *cheap* machines

$$\mathcal{C}_j = \left\{ i : c_{ij} \leq \frac{\varepsilon}{m} d_j \right\},$$

the set of *slow* machines

$$\mathcal{S}_j = \left\{ i : p_{ij} \geq \frac{m}{\varepsilon} d_j \right\},$$

and the set of *expensive* machines

$$\mathcal{E}_j = \{ i : c_{ij} \geq d_j / \varepsilon \}.$$

Then, we can prove the following lemma.

Lemma 4.1 *With $1+3\varepsilon$ loss, we can assume that for each job J_j the following holds:*

- *$p_{ij} = 0$, for $i \in \mathcal{F}_j$, and $c_{ij} = 0$, for $i \in \mathcal{C}_j$;*
- *$p_{ij} = +\infty$, for $i \in \mathcal{S}_j$, and $c_{ij} = +\infty$, for $i \in \mathcal{E}_j$;*
- *for any other machine i, $p_{ij} = \frac{\varepsilon}{m} d_j (1+\varepsilon)^{\pi_i}$ and $c_{ij} = \frac{\varepsilon}{m} d_j (1+\varepsilon)^{\gamma_i}$, where $\pi_i, \gamma_i \in \mathbb{N}$.*

Proof. Set $p_{ij} = 0$, for every $i \in \mathcal{F}_j$ and $j = 1, ..., n$; consider an optimal assignment $A : \mathcal{J} \to M$ of jobs to machines. Clearly, the optimal value corresponding to A cannot be larger than OPT. Let F denote the set of jobs which are processed on fast machines according to A. Now, if we replace the processing times of the transformed instance by the original processing times, we may potentially increase the makespan of A by at most $\sum_{J_j \in F} \frac{\varepsilon}{m} d_j \leq \sum_{j=1}^{n} \frac{\varepsilon}{m} d_j = \varepsilon \frac{D}{m} \leq \varepsilon$. The other statement for the cost values follows in a similar way.

Now we show that there exists an approximate schedule where jobs are scheduled neither on slow nor on expensive machines. This allows us to set $p_{ij} = +\infty$, for $i \in \mathcal{S}_j$, and $c_{ij} = +\infty$, for $i \in \mathcal{E}_j$. Consider an optimal assignment $A : \mathcal{J} \to M$ of jobs to machines with T^* and C^* denoting the resulting makespan and cost, respectively. Let S and E represent, respectively, the set of jobs which are processed on slow and on expensive machines according to A. Then, assign every job $J_j \in S \cup E$ to machine m_j (recall that $m_j \in \{1, ..., m\}$ denote any machine such that $d_j = p_{m_j,j} + c_{m_j,j}$). Moving jobs $J_j \in S \cup E$ onto machines m_j may potentially increase the objective value by at most $\sum_{J_j \in S \cup E} d_j \leq \frac{\varepsilon}{m} \sum_{J_j \in S} p_{A(j),j} + \varepsilon \sum_{J_j \in E} c_{A(i),j} \leq \varepsilon T^* + \varepsilon C^*$, since $p_{A(j),j} \geq \frac{m}{\varepsilon} d_j$ for $J_j \in S$, and $c_{A(j),j} \geq d_j/\varepsilon$ for $J_j \in E$.

By the above arguments, all the positive costs c_{ij} and positive processing times p_{ij} are greater than $\frac{\varepsilon}{m} d_j$. Round every positive processing time p_{ij} and every positive cost c_{ij} down to the nearest lower value of $\frac{\varepsilon}{m} d_j (1+\varepsilon)^h$, for $h \in \mathbb{N}$, $i = 1, ..., m$ and $j = 1, ..., n$. Consider the optimal value of the rounded instance. Clearly, this value cannot be greater than OPT. It follows that by replacing the rounded values with the original ones we may increase each value by a factor $1+\varepsilon$, and consequently, the solution value potentially increases by the same factor $1+\varepsilon$. ■

We define the *execution profile* of a job J_j to be an m-tuple $< \pi_{1j}, ..., \pi_{mj} >$ such that $p_{ij} = \frac{\varepsilon}{m} d_j (1+\varepsilon)^{\pi_{ij}}$. We adopt the convention that $\pi_{ij} = +\infty$ if $p_{ij} = +\infty$, and $\pi_{ij} = -\infty$ if $p_{ij} = 0$. Likewise, we define the *cost profile* of a job J_j to be an m-tuple $< \gamma_{1j}, ..., \gamma_{mj} >$ such that $c_{ij} = \frac{\varepsilon}{m} d_j (1+\varepsilon)^{\gamma_{ij}}$.

Again, we adopt the convention that $\gamma_{ij} = +\infty$ if $c_{ij} = +\infty$, and $\gamma_{ij} = -\infty$ if $c_{ij} = 0$. Let us say that two jobs have the same profile if and only if they have the same execution and cost profile.

Lemma 4.2 *The number ℓ of distinct profiles is bounded by*

$$\ell \leq \left\lceil 2 + 2\log_{1+\varepsilon}(m/\varepsilon) \right\rceil^{2m}$$

.

Proof. The number of distinct job profiles can be bounded by x^{2m}, where x is the number of distinct elements in any job profile. By Lemma 4.1, any processing time p_{ij} (and cost c_{ij}) can be either 0 or $+\infty$ or $\frac{\varepsilon}{m} d_j (1+\varepsilon)^h$, for some $h \in \mathbb{N}$. Since by Lemma 4.1 any finite processing time and cost is less than $\frac{m}{\varepsilon} d_j$, we have $h < 2\log_{1+\varepsilon}(\frac{m}{\varepsilon})$. Thus, the number x of different values in any job profile can be bounded by $x < 3 + 2\log_{1+\varepsilon}(\frac{m}{\varepsilon})$. Since x must be integral, we have $x \leq \left\lceil 2 + 2\log_{1+\varepsilon}(\frac{m}{\varepsilon}) \right\rceil$ and the claim follows. ■

4.2.2 Grouping Jobs

Let $\delta := \frac{\varepsilon}{m}$, and partition the set of jobs in two subsets $L = \{J_j : d_j > \delta\}$ and $S = \{J_j : d_j \leq \delta\}$. Let us say that L is the set of *large* jobs, while S the set of *small* jobs. We further partition the set of small jobs into subsets S_i of jobs having the same profile, for $i = 1, ..., \ell$. Let J_a and J_b be two jobs from S_i such that $d_a, d_b \leq \delta/2$. We "group" together these two jobs to form a composed job J_c in which the processing time (and cost) on machine i is equal to the sum of the processing times (and costs) of J_a and J_b on machine i, and let $d_c = d_a + d_b$. We repeat this process, by using the modified set of jobs, until at most one job J_j from S_i has $d_j \leq \delta/2$. At the end, all jobs J_j in group S_i have $d_j \leq \delta$. The same procedure is performed for all other subsets S_i. At the end of this process, there are at most ℓ jobs, one for each subset S_i, having $d_j \leq \delta/2$. All the other jobs, have processing times larger than $\delta/2$. Therefore, the number of jobs in the transformed instance is bounded by $\frac{2D}{\delta} + \ell \leq \frac{2m}{\delta} + \ell = (\log m/\varepsilon)^{O(m)}$.

Note that the procedure runs in linear time, and a feasible schedule for the original set of jobs can be easily obtained from a feasible schedule for the grouped jobs. We motivate the described technique with the following

Lemma 4.3 *With $1+\varepsilon$ loss, the number of jobs can be reduced in linear time to be at most* $\min\{n, (\log m/\varepsilon)^{O(m)}\}$.

Proof. Consider the transformed instance I' according to Lemma 4.1 (small jobs are not yet grouped). Assume, without loss of generality, that there exists a solution SOL' for I' of value OPT'. It is sufficient to show that, by using the small jobs grouped as described previously, there exists a schedule of value $(1+\varepsilon)OPT'$.

Accordingly to SOL' we may assume that each machine executes the large jobs at the beginning of the schedule. Let c denote the total cost of large jobs when processed according to SOL', and let t_i denote the time at which machine i finishes to process large jobs, for $i = 1, ..., m$. Now, consider the following linear program LP_1:

$$\begin{array}{lllll} \min & T + c + \sum_{i=1}^{m} \sum_{J_j \in S} x_{ij} \frac{\varepsilon}{m} d_j (1+\varepsilon)^{\gamma_{ij}} & & & \\ s.t. & t_i + \sum_{J_j \in S} x_{ij} \frac{\varepsilon}{m} d_j (1+\varepsilon)^{\pi_{ij}} \leq T, & & & i = 1, \ldots, m; \\ & \sum_{i=1}^{m} x_{ij} = 1, & & & J_j \in S; \\ & x_{ij} \geq 0, & & i = 1, \ldots, m, & J_j \in S. \end{array}$$

Note that LP_1 formulates the integer relaxation of the original problem ILP for the subset of small jobs: we are assuming that machine i can start processing small jobs only at time t_i, and when the processing times and costs are structured as in Lemma 4.1.

For each S_ϕ, $\phi = 1, ..., \ell$, consider a set of decision variables $y_{\phi i} \in [0, 1]$ for $i = 1, ..., m$. The meaning of these variables is that $y_{\phi i}$ represents the fraction of jobs from S_ϕ processed on machine i. Consider the following linear program LP_2:

$$\begin{array}{lllll} \min & T + c + \sum_{i=1}^{m} \sum_{\phi=1}^{\ell} y_{\phi i} \sum_{J_j \in S_\phi} \frac{\varepsilon}{m} d_j (1+\varepsilon)^{\gamma_{ij}} & & & \\ s.t. & t_i + \sum_{\phi=1}^{\ell} y_{\phi i} \sum_{J_j \in S_\phi} \frac{\varepsilon}{m} d_j (1+\varepsilon)^{\pi_{ij}} \leq T, & & & i = 1, \ldots, m; \\ & \sum_{i=1}^{m} y_{\phi i} = 1, & & & \phi = 1, \ldots, \ell; \\ & y_{\phi i} \geq 0, & & i = 1, \ldots, m, & \phi = 1, \ldots, \ell. \end{array}$$

By setting $y_{\phi i} = \frac{\sum_{J_j \in S_\phi} x_{ij} d_j}{\sum_{J_j \in S_\phi} d_j}$, it is easy to check that any feasible set of values (x_{ij}) for LP_1 gives a feasible set of values $(y_{\phi i})$ for LP_2; furthermore, by these settings, the objective function value of LP_2 is equal to that of LP_1. But LP_1 is a relaxation of the original problem. Therefore, if we were able, by using the grouped small jobs, to get a feasible schedule of length at most $1+\varepsilon$ times the optimal value of LP_2, we would be done. In the remainder we show how to generate such a schedule. This solution is obtained by using the solution of LP_2 and the small grouped jobs.

Let $y^*_{\phi i}$ denote the values of variables $y_{\phi i}$ according to the optimal solution of LP_2. For every positive value $y^*_{\phi i}$, schedule a subset of grouped

jobs from S_ϕ on machine i until either (a) the jobs from S_ϕ are exhausted or (b) the total fraction of jobs assigned to i is equal to $y^*_{\phi i}$ (if necessary fractionalize one job to use up $y^*_{\phi i}$ exactly). We repeat this for the not yet assigned grouped small jobs and for every positive value $y^*_{\phi i}$. Note that if $y^*_{\phi i}$ is not fractional, then the jobs from S_ϕ are not preempted by the previous algorithm. In general, the number of preempted jobs from S_ϕ is at most $f_\phi - 1$, where $f_\phi = |\left\{y^*_{\phi i} : y^*_{\phi i} > 0, i = 1, ..., m\right\}|$. Now remove all the preempted jobs J_j and schedule them at the end on machines m_j. This increases the makespan and the cost by at most $\Delta = \delta \cdot \sum_{\phi=1}^{\ell}(f_\phi - 1)$, since every grouped small job has cost plus processing time bounded by δ when processed on machine m_j. A basic feasible solution of LP_2 has the property that the number of positive variables is at most the number of rows in the constraint matrix, $m + \ell$, therefore $\sum_{\phi=1}^{\ell}(f_\phi - 1) \leq m$. In order to bound the total increase Δ by ε we have to choose δ such that $\delta \leq \frac{\varepsilon}{m}$, and the claim follows. ■

4.2.3 Dynamic Programming

The optimal solution for the transformed instance with $k = (\log m/\varepsilon)^{O(m)}$ jobs can be computed in $m^k = m^{(\log m/\varepsilon)^{O(m)}}$, by simply enumerating all possible assignments and taking the smallest one. However, to avoid the exponential dependence on $1/\varepsilon$ (recall, our aim is to obtain a fully polynomial approximation scheme), we will not treat separately all of these schedules. Furthermore, for our problem we do not need to find an optimal solution for the transformed instance, an approximate solution suffices.

In the following, we show that an approximate schedule for the transformed instance can be computed in $(\log m/\varepsilon)^{O(m^2)}$ time. The first step is to round down every processing time and cost to the nearest lower value of $(\varepsilon/k)i$, for $i = 0, 1, \ldots, k/\epsilon$; clearly this does not increase the objective function value. Then, find the optimal solution SOL of the resulting instance by using a dynamic programming approach. Finally, by replacing the rounded values with the originals, it is easy to check that we may potentially increase the cost and the makespan of SOL by at most ε, respectively. This results in a $(1 + 2\varepsilon)$-approximate solution for the transformed instance.

We now present the dynamic programming algorithm. Let us to denote $J_1, ..., J_k$ the k jobs of the transformed instance. Let a *schedule configuration* $\mathbf{s} = (t_1, ..., t_m, c)$ be defined as an $(m+1)$-dimensional vector, where t_i denotes the completion time of machine i and c the total cost. For each job J_j let V_j denote the set of $(m+1)$-dimensional vectors defined as follows: for each

$i = 1, ..., m$, there is a vector $\mathbf{v} \in V_j$ whose entries are 0 except for the ith component which is p_{ij}, and the $(m+1)$-st component which is c_{ij}. Let $T(j, \mathbf{s})$ denote the truth value of the statement: "there is a schedule for jobs $J_1, .., J_j$ for which $\mathbf{s}$ is the corresponding schedule configuration". The values of all the $T(j, \mathbf{s})$ can be viewed as being arranged in a table, and the crux of the approach lies in the following very simple procedure that can be used for filling in the table entries:

$$\begin{aligned} T(1, \mathbf{v}) &= \begin{cases} \text{true} & \text{if } \mathbf{v} \in V_1 \\ \text{false} & \text{if } \mathbf{v} \notin V_1 \end{cases} \\ T(j, \mathbf{s}) &= \bigvee_{\mathbf{v} \in V_j:\ \mathbf{v} \leq \mathbf{s}} T(j-1, \mathbf{s} - \mathbf{v}) \quad \text{for } j = 2, ..., k. \end{aligned}$$

The reader should have no difficulty to bound the time to fill in the table entries by $O(mN)$, where N is the number of table entries. By the way we have rounded the values, the number of different completion times for each machine and different costs is bounded by $mk/\varepsilon + 1$. It follows that the total number of different schedule configurations is at most $(mk/\varepsilon + 1)^{m+1}$. By observing that it is sufficient to store the minimum cost value for each different m-dimensional vector $(t_1, ..., t_m)$ of completion times, we have $N = k(mk/\varepsilon + 1)^m$. Therefore, the total running time of the dynamic program is $O(km(mk/\varepsilon + 1)^m) = (\log m/\varepsilon)^{O(m^2)}$.

By Lemma 4.3, and by using the described dynamic programming approach, we have the following

Theorem 4.1 *For the problem of minimizing the weighted sum of the cost and the makespan in scheduling n jobs on m unrelated machines (m fixed), there exists a fully polynomial time approximation scheme that runs in $O(n) + (\log m/\varepsilon)^{O(m^2)}$ time.*

Corollary 4.1 *For any fixed m, there is a fully polynomial-time approximation scheme for the non-preemptive minimum makespan scheduling problem with n jobs and m unrelated parallel machines that runs in $O(n) + (\log m/\varepsilon)^{O(m^2)}$ time.*

Chapter 5

Job Shop Scheduling Problems

5.1 Introduction

In this chapter, we discuss the job shop scheduling problem. We consider both, the problem with fixed processing times, and the problem with controllable processing times.

In the job shop scheduling problem with fixed processing times, there is a set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of n jobs that must be processed on a given set $M = \{1, \ldots, m\}$ of m machines. Each job J_j consists of a sequence of μ operations $O_{1j}, O_{2j}, \ldots, O_{\mu j}$ that need to be processed in this order. Operation O_{ij} must be processed without interruption on machine $m_{ij} \in M$, during p_{ij} time units. Each machine can process at most one operation at a time, and each job may be processed by a most one machine at any time. For any given schedule, let C_{ij} be the completion time of operation O_{ij}. The objective is to find a schedule that minimizes the *makespan* (the maximum completion time $C_{\max} = \max_{ij} C_{ij}$).

The job shop problem with fixed processing times is strongly NP-hard even if each job has at most three operations and there are only two machines [52]. Williamson et al. [81] proved that when the number of machines, jobs, and operations per job are part of the input there does not exist a polynomial time approximation algorithm with worst case bound smaller than $\frac{5}{4}$ unless $P = NP$. When m and μ are part of the input the best known result [23] is an approximation algorithm with worst case bound $O((\log(m\mu)\log(min(m\mu, p_{max}))/\log\log(m\mu))^2)$, where p_{max} is the largest processing time among all operations. For those instances where m and μ

are fixed (the restricted case we are focusing on in this chapter), Shmoys et al. [75] gave approximation algorithms that compute $(2+\varepsilon)$-approximate solutions in polynomial time for any fixed $\varepsilon > 0$. This result has recently been improved by Jansen, Solis-Oba and Sviridenko [45] who have shown that $(1+\varepsilon)$-approximate solutions of the problem can be computed in polynomial time. The main idea is to divide the set of jobs $\mathcal{J}$ into two groups $\mathcal{L}$ and $\mathcal{S}$ formed by jobs with "large" and "small" total processing time, respectively. The total number of large jobs is bounded by a constant exponentially in m, μ and ε. Then they construct all possible schedules for the large jobs. In any schedule for the large jobs, the starting and completion times of the jobs define a set of time intervals, into which the set of small jobs have to be scheduled. Then for every possible job ordering of large jobs, a linear program is used to assign small jobs to time intervals. The rounded solution of the linear program in combination with an algorithm by Sevastianov [73], gives an approximate schedule in time polynomial in n. In order to speed up the whole algorithm, they suggest [44] a number of improvements: they use the logarithmic potential price decomposition method of Grigoriadis and Khachiyan [28] to compute an approximate solution of the linear program in linear time, and a novel rounding procedure to bring down to a constant the number of fractional assignments in any solution of the linear program. The overall running time is $O(n)$, where the multiplicative constant hidden in the $O(n)$ running time is exponentially in m, μ and ε.

In contrast to the approximation schemes introduced in [45, 44], we present an algorithm that is extremely simple and even faster. We show that we can preprocess in linear time the input to obtain a new instance with a constant number of grouped jobs. This immediately gives a linear time approximation scheme with running time $O(n) + C$, where C is a constant that depends on m, ε and μ. Again, we remark that the multiplicative constant hidden in the $O(n)$ running time of our algorithm is reasonably small and does not depend on the accuracy ε.

In the second part of this chapter we consider two variants of the job shop problem with controllable processing times. The first variant allows *continuous* changes to the processing times of the operations. The second assumes only *discrete* changes. In the case of continuously controllable processing times, we assume that the cost of reducing the time needed to process an operation is a linear function of the processing time. In the case of discretely controllable processing times, there is a finite set of possible processing times and costs for every operation O_{ij}. We observe that, since for problem P2 deciding whether there is a solution of length $T(\sigma, \delta) \leq \tau$ is

already NP-complete, the best result that we might expect to obtain (unless P=NP) is a solution with cost at most the optimal cost and makespan not greater than $\tau(1+\varepsilon)$, $\varepsilon > 0$. Nowicki and Zdrzalka [66] show that the version of problem P3 for the less general flow shop problem with continuously controllable processing times is NP-hard even when there are only two machines.

We present the first known polynomial time approximation schemes for problems P1, P2, and P3, when the number m of machines and the number μ of operations per job are fixed. Our algorithms can handle both, continuously and discretely controllable processing times, and they can be extended to the case of convex piecewise linear processing times and cost functions. These results improve the 4/3-approximation algorithm for problem P3 described in Nowicki [65]. Moreover, the linear time complexity of our PTAS for problem P3 is the best possible with respect to the number of jobs.

Our algorithms are based on the approach described by Jansen, Solis-Oba and Sviridenko [45]. However, a major difficulty with using this approach for our problems is that the processing times and costs of the operations are not fixed, so we must determine their values before we can use the above approach. One possibility is to use a linear program to assign jobs to gaps and to determine the processing times and costs of the operations. But, we must be careful, since, for example, a natural extension of the linear program described in [44] defines a polytope with an exponential number of extreme points, and it does not seem to be possible to solve such linear program in polynomial time. We show how to construct a small polytope with only a polynomial number of extreme points that contains all the optimum solutions of the above linear program. This polytope is defined by a linear program that can be solved exactly in polynomial time and approximately, to within any pre-specified precision, in strongly polynomial time.

Our approach is robust enough so that it can be used to design polynomial time approximation schemes for both the discrete and the continuous versions of problems P1-P3. In this paper we present polynomial time approximation schemes for the continuous version of problems P1-P3, and a PTAS for the discrete version of problem P3. The polynomial time approximation schemes for the discrete version of P1 and P2 can be easily obtained by combining the described techniques.

Let $\varepsilon > 0$ be an arbitrary small rational number, and let $m \geq 2$ and $\mu \geq 1$ be integral values. Throughout this chapter, the values ε, m and μ are considered to be constants and not part of the input.

5.2 Fixed Processing Times

In this section we address the job shop scheduling problem with fixed processing times. We begin by providing some lower and upper bounds of the minimum makespan. Then we show how to reduce the number of jobs to a constant: the reduced set of jobs is computed in linear time by structuring the input and grouping jobs that share the same structure. This directly gives a linear time approximation scheme for the makespan minimization of the job shop scheduling problem.

For a given instance of the job shop scheduling problem, the value of the optimum makespan will be denoted as OPT. Let $d_j = \sum_{i=1}^{\mu} p_{ij}$ be the total processing time of job $J_j \in \mathcal{J}$, and let $D = \sum_{J_j \in \mathcal{J}} d_j$. Clearly, $D/m \leq OPT$ and a schedule of length at most D can be obtained by scheduling one job after the other. Then, we get directly the following bounds: $\frac{D}{m} \leq OPT \leq D$. By dividing all execution times p_{ij} by D/m, we assume, without loss of generality, that $D/m = 1$ and

$$1 \leq OPT \leq m.$$

5.2.1 Negligible Operations

Let us use $\mathcal{N}_j = \left\{ O_{ij} : i = 1, ..., \mu \text{ and } p_{ij} \leq \frac{\varepsilon}{\mu m} d_j \right\}$ to denote the set of *negligible* operations for job J_j, $j = 1, ..., n$. Then we have the following

Lemma 5.1 *With $1 + 2\varepsilon$ loss, we assume that for each job J_j the following holds:*

- $p_{ij} = 0$, *for* $O_{ij} \in \mathcal{N}_j$;
- $p_{ij} = \frac{\varepsilon}{\mu m} d_j (1 + \varepsilon)^{\pi_i}$, *where* $\pi_i \in \mathbb{N}$ *and for* $O_{ij} \notin \mathcal{N}_j$.

Proof. Set $p_{ij} = 0$, for every $O_{ij} \in \mathcal{N}_j$ and $j = 1, ..., n$. Clearly, the corresponding optimal makespan cannot be larger than OPT. Furthermore, if we replace the zero processing times of the negligible operations with the original processing times, we may potentially increase the makespan by at most $\sum_{O_{ij} \in \mathcal{N}} \frac{\varepsilon}{\mu m} d_j = \sum_{j=1}^{n} \sum_{O_{ij} \in \mathcal{N}_j} \frac{\varepsilon}{\mu m} d_j \leq \varepsilon \frac{D}{m} = \varepsilon$.

Any non negligible operation O_{ij} has processing times p_{ij} greater than $\frac{\varepsilon}{\mu m} d_j$. Round each p_{ij} down to the nearest lower value of $\frac{\varepsilon}{\mu m} d_j (1 + \varepsilon)^h$, where $h \in \mathbb{N}$ and $O_{ij} \notin \mathcal{N}_j$. By replacing the rounded values with the original ones we may potentially increases the makespan by at most a factor of $1 + \varepsilon$. ■

Now we define the *profile* of a job J_j to be a $(\mu \cdot m)$-tuple

$$< \pi_{1,1,j}, ..., \pi_{1,m,j}, \pi_{2,1,j}, ..., \pi_{\mu,m,j} >$$

such that $p_{ij} = \frac{\varepsilon}{\mu m} d_j (1+\varepsilon)^{\pi_{i,h,j}}$ if $m_{ij} = h$. We adopt the convention that $\pi_{i,h,j} = -\infty$ if $p_{ij} = 0$ or if $m_{ij} \neq h$.

Lemma 5.2 *The number of distinct profiles is bounded by* $\ell := \lfloor 2 + \log_{1+\varepsilon} \frac{\mu m}{\varepsilon} \rfloor^{\mu} \cdot m^{\mu}$.

Proof. Since each different operation can be processed by one given machine, the number of distinct job profiles can be bounded by $(x \cdot m)^{\mu}$, where x is the number of distinct values in any job profile. By Lemma 5.1, any processing time p_{ij} can be either 0 or $\frac{\varepsilon}{\mu m} d_j (1+\varepsilon)^h$, for some $h \in \mathbb{N}$. Since any processing time is less than $D \leq 1$, we have $h \leq \log_{1+\varepsilon}(\frac{\mu m}{\varepsilon})$. Thus, the number x of different values in any job profile can be bounded by $x \leq 2 + \log_{1+\varepsilon}(\frac{m}{\varepsilon})$. Since x must be integral, we have $x \leq \lfloor 2 + \log_{1+\varepsilon} \frac{\mu m}{\varepsilon} \rfloor$ and the claim follows. ■

5.2.2 Grouping Jobs

Let $\delta := m(\frac{\varepsilon}{4\mu^4 m^2})^{m/\varepsilon}$, and partition the set of jobs in two subsets of large jobs $L = \{J_j : d_j > \delta\}$ and small jobs $S = \{J_j : d_j \leq \delta\}$. Again, we further partition the set of small jobs into subsets S_i of jobs having the same profile, for $i = 1, ..., \ell$. For each subset S_i, we group jobs from S_i as described for the unrelated parallel machines scheduling problem, but with the following difference: here grouping two jobs, J_a and J_b, means to form a composed job for which the processing time of the i-th operation is equal to the sum of the processing times of the i-th operations of J_a and J_b.

Theorem 5.1 *With* $1+2\varepsilon$ *loss, the number of jobs can be reduced in linear time to be at most* $\min\{n, \frac{2m}{\delta} + \ell\}$.

Proof. Consider the transformed instance I' according to Lemma 5.1 (small jobs are not yet grouped). Assume, without loss of generality, that there exists a solution SOL' for I' of value OPT'. It is sufficient to show that, by using the small jobs grouped as described previously, there exists a schedule of value $(1+2\varepsilon)OPT'$.

Partition the set of large jobs into three subsets L_1, L_2, L_3 as follows. Set $\rho = \frac{\varepsilon}{4\mu^4 m^2}$ and let α denote an integer defined later and such that

$$\alpha = 0, 1, ..., m/\varepsilon - 1.$$

Set L is partitioned in the following three subsets:

$$\begin{aligned} L_1 &= \{J_j : m\rho^{\alpha} < d_j\}, \\ L_2 &= \{J_j : m\rho^{\alpha+1} < d_j \leq m\rho^{\alpha}\}, \\ L_3 &= \{J_j : m\rho^{m/\varepsilon} < d_j \leq m\rho^{\alpha+1}\}. \end{aligned}$$

Note that each set size is bounded by a constant, and sets L_1, L_2, L_3 and S establish a partition of all jobs. The number α can be chosen so that

$$\sum_{J_j \in L_2} d_j \leq \varepsilon. \tag{5.1}$$

This is done as follows. Starting from $\alpha := 0$, check each time whether the set L_2 corresponding to the current value of α satisfies inequality (5.1); if it is not the case, set $\alpha := \alpha + 1$ and repeat. Note that for different α-values, the corresponding L_2 sets are disjoint. The total length of all jobs is m, and so there exists a value $\alpha' \leq m/\varepsilon - 1$ for which the corresponding set L_2 satisfies inequality (5.1). We set $\alpha := \alpha'$.

In the following we consider an artificial situation that we use as a tool for proving Theorem 5.1. Focus on solution SOL', remove from SOL' all the jobs except those from L_1. Clearly, this creates gaps in the resulting schedule σ. The starting and finishing times of the operations from L_1 divide the time into intervals: $t_1, t_2, ..., t_g$, where t_1 is the interval whose right boundary is the starting time of the first operation according to σ, and t_g is the interval with left boundary defined by the finishing time of the jobs from L_1. Furthermore, let l_v denotes the length of interval t_v, $1 \leq v \leq g$. It follows that $OPT' = \sum_{s=1}^{g} l_s$. Note that the number g of intervals is bounded by $2\mu|L_1| + 1 \leq 2\mu/\rho^{\alpha}$. Let G be the set of pairs (gaps of σ) (v, i) such that no job from L_1 is processed in interval v and by machine i, for $v = 1, ..., g$ and $i = 1, ..., m$.

Since the total length of jobs from L_2 is at most ε, we can get rid of these jobs by assuming that they are processed at the end of the schedule one after the other; this increases the schedule by at most ε.

Consider the problem of placing jobs from $L_3 \cup S$ into the gaps of σ and such that the length of the resulting schedule is OPT'. In the following we first describe a linear program LP_1 which is a relaxation of this problem. Then we propose another linear program LP_2 which is a relaxation of LP_1. By using the solution of LP_2 we show that the jobs from L_3 and the grouped small jobs can be scheduled into the gaps by increasing the length of the gaps a little; we show that the total increase of the length can be bounded by ε, and the claim follows.

We formulate LP_1 as follows. Consider the set S of (not grouped) small jobs. For each job $J_j \in S \cup L_3$ we use a set of decision variables $x_{j,\tau} \in [0,1]$ for tuples $\tau = (\tau_1, \ldots, \tau_\mu) \in A$, where $A = \{(\tau_1, \ldots, \tau_\mu) | 1 \leq \tau_1 \leq \tau_2 \leq \ldots \leq \tau_\mu \leq g\}$. The meaning of these variables is that $x_{j,\tau}$ represents the fraction of job J_j whose operations are processed according to $\tau = (\tau_1, \ldots, \tau_\mu)$, i.e., the i-th operation is scheduled in interval τ_k for each $1 \leq k \leq \mu$. Note that by the way in which we numbered the operations, any tuple $(\tau_1, \ldots, \tau_\mu) \in A$ represents a valid ordering for the operations. Let the load $L_{v,h}$ on machine h in interval v be defined as the total processing time of operations from small jobs that are executed by machine h during interval s, i.e., $L_{v,h} = \sum_{J_j \in S \cup L_3} \sum_{\tau \in A} \sum_{k=1,\ldots,\mu | \tau_k = v, m_{kj} = h} x_{j,\tau} p_{kj}$. Let us write the load $L_{v,h}$ as the sum of $L^3_{v,h} + L^S_{v,h}$, where $L^3_{v,h}$ is the load of the jobs from L_3 while $L^S_{v,h}$ is the load of jobs from S. By Lemma 5.1, we have that

$$L^S_{v,h} = \sum_{\tau \in A} \sum\nolimits_{k=1,\ldots,\mu | \tau_k = v} \sum_{J_j \in S} x_{j,\tau} \frac{\varepsilon}{\mu m} d_j (1+\varepsilon)^{\pi_{k,h,j}}.$$

Then LP_1 is the following

$$\begin{array}{lll} (1) & L^3_{v,h} + L^S_{v,h} \leq l_v, & (v,h) \in G; \\ (2) & \sum_{\tau \in A} x_{j\tau} = 1, & J_j \in S \cup L_3; \\ (3) & x_{j\tau} \geq 0, & \tau \in A,\ J_j \in S \cup L_3. \end{array}$$

Constraint (1) ensures that the total length of operations assigned to gap (v,h) does not exceed the length of the interval, while constraint (2) ensures that job J_j is completely scheduled.

Let S_ϕ denote the set of small jobs having the same profile, where $\phi = 1, ..., \ell$. For each S_ϕ ($\phi = 1, ..., \ell$) we use a set of decision variables $y_{\phi\tau} \in [0,1]$ for tuples $\tau = (\tau_1, \ldots, \tau_\mu) \in A$. The meaning of these variables is that $y_{\phi\tau}$ represents the fraction of jobs from S_ϕ whose operations are processed according to $\tau = (\tau_1, \ldots, \tau_\mu)$, i.e., the i-th operation is scheduled in interval τ_k for each $1 \leq k \leq \mu$. Let

$$L^*_{v,h} = \sum_{\tau \in A} \sum\nolimits_{k=1,\ldots,\mu | \tau_k = v} \sum_{\phi=1}^{\ell} y_{\phi\tau} \sum_{J_j \in S_\phi} x_{j,\tau} \frac{\varepsilon}{\mu m} d_j (1+\varepsilon)^{\pi_{k,h,j}}.$$

Then LP_2 is the following

$$\begin{array}{lll} (1) & L^3_{v,h} + L^*_{v,h} \leq t_s, & (v,h) \in G; \\ (2) & \sum_{\tau \in A} x_{j\tau} = 1, & J_j \in L_3; \\ (3) & \sum_{\tau \in A} y_{\phi\tau} = 1, & \phi = 1, ..., \ell; \\ (4) & x_{j\tau} \geq 0, & \tau \in A,\ J_j \in S; \\ (5) & y_{\phi\tau} \geq 0, & \tau \in A,\ \phi = 1, ..., \ell. \end{array}$$

By setting $y_{\phi\tau} = \frac{\sum_{J_j \in S_\phi} x_{j,\tau} d_j}{\sum_{J_j \in S_\phi} d_j}$ it is easy to check that any feasible set of values $(x_{j,\tau})$ for LP_1 gives a feasible set of values $(y_{\phi\tau})$ for LP_2. Since by construction a feasible solution for LP_1 exists, a feasible solution for LP_2 exists as well. We show now that by using the optimal solution of LP_2 we can derive a schedule without increasing too much the makespan.

Let $y^*_{\phi\tau}$ $(x^*_{j\tau})$ denote the values of variables $y_{\phi\tau}$ $(x_{j\tau})$ according to the optimal solution of LP_2. For every positive value $y^*_{\phi\tau}$, schedule a subset $H_{\phi\tau}$ of grouped jobs from S_ϕ on machine i until either (a) the jobs from S_ϕ are exhausted or (b) the total fraction (i.e. $\frac{\sum_{J_j \in H_{\phi\tau}} d_j}{\sum_{J_j \in S_\phi} d_j}$) of jobs assigned to i is equal to $y^*_{\phi\tau}$ (if necessary fractionalize one job to use up $y^*_{\phi\tau}$ exactly). We repeat this for the not yet assigned grouped small jobs and for every positive value $y^*_{\phi\tau}$. Note that if $y^*_{\phi\tau}$ is not fractional, then the jobs from S_ϕ are not preempted by the previous algorithm. In general, the number of preempted jobs from S_ϕ is at most $f_\phi - 1$, where $f_\phi = |\left\{y^*_{\phi\tau} : y^*_{\phi\tau} > 0, \tau \in A\right\}|$. According to the optimal solution of LP_2, let us say that job $J_j \in L_3$ is preempted if the corresponding $(x^*_{j\tau})$-values are fractional. Let $f_j = |\left\{x^*_{j\tau} : x^*_{j\tau} > 0, \tau \in A\right\}|$, for $J_j \in L_3$, then we have that the number of preemptions of job $J_j \in L_3$ is $f_j - 1$. Therefore, the total number of preemptions is $f = \sum_{\phi=1}^{\ell}(f_\phi - 1) + \sum_{J_j \in L_3}(f_j - 1)$, and this gives also an upper bound on the number of preempted jobs. Now remove all the preempted jobs J_j from $S \cup L_3$, and schedule these set of jobs at the end of the schedule, one after the other. Since every job from S has a smaller total processing time than any job from L_3, we can bound the total increase of the schedule length by $\Delta = f \cdot m\rho^{\alpha+1}$. A basic feasible solution of LP_2 has the property that the number of positive variables is at most the number of rows in the constraint matrix, $mg + \ell + |L_3|$, therefore $f \leq mg \leq 2m\mu/\rho^\alpha$, and $\Delta = f \cdot m\rho^{\alpha+1} \leq 2m^2\mu\rho$. By the previous algorithm, we have assigned all the jobs from $S \cup L_3$ to gaps with a total increase of the schedule length of $2m^2\mu\rho$. Now we consider the problem of schedule jobs from $S \cup L_3$ within each interval. This is simply a smaller instance of the job shop problem, and by using Sevastianov's algorithm [73] it is possible to find a feasible schedule for each interval t_v of length at most $l_v + \mu^3 m \cdot m\rho^{\alpha+1}$; this increases the total length by at most $\mu^3 mg \cdot m\rho^{\alpha+1} \leq 2m^2\mu^4\rho$. Therefore, the total increase is $2m^2\mu\rho + 2m^2\mu^4\rho \leq 4m^2\mu^4\rho$, and by setting $\rho = \frac{\varepsilon}{4m^2\mu^4}$ the claim follows. ■

By the previous Theorem a $(1+\varepsilon)$-approximate schedule can be obtained by finding the optimum schedule for the reduced set of jobs. It follows that:

Theorem 5.2 *There exists a linear time PTAS for the job shop scheduling problem whose multiplicative constant hidden in the $O(n)$ running time is reasonably small and does not depend on the error ε, whereas the additive constant is exponential in m, μ and $1/\varepsilon$.*

5.3 Controllable Processing Times

In this section we discuss the job shop scheduling problem with controllable processing times. In Sections 5.3.1, 5.3.2 and 5.3.3 we present polynomial time approximation schemes for problem P1, P2 and P3, respectively, with continuous processing times. In Section 5.3.4 we study problem P3 with discrete processing times, and show how to design a linear time PTAS for it.

5.3.1 Problem P1 with Continuous Processing Times

In the case of continuously controllable processing times, we assume that for each job O_{ij} there is an interval $[\ell_{ij}, u_{ij}]$, $0 \leq \ell_{ij} \leq u_{ij}$, specifying its possible processing times. The cost for processing operation O_{ij} in time ℓ_{ij} is denoted as $c_{ij}^{\ell} \geq 0$ and for processing it in time u_{ij} the cost is $c_{ij}^{u} \geq 0$. For any value $\delta_{ij} \in [0,1]$ the cost for processing operation O_{ij} in time $p_{ij}^{\delta_{ij}} = \delta_{ij}\ell_{ij} + (1-\delta_{ij})u_{ij}$ is $c_{ij}^{\delta_{ij}} = \delta_{ij}c_{ij}^{\ell} + (1-\delta_{ij})c_{ij}^{u}$. We assume that ℓ_{ij}, u_{ij}, c_{ij}^{ℓ}, c_{ij}^{u} and δ_{ij} are rational numbers. Moreover, without loss of generality, we assume that for every operation O_{ij}, $c_{ij}^{u} \leq c_{ij}^{\ell}$, and if $c_{ij}^{u} = c_{ij}^{\ell}$ then $u_{ij} = \ell_{ij}$.

Let us consider an instance of problem P1. Divide all c_{ij}^{u} and c_{ij}^{ℓ} values by κ to get an equivalent instance in which the bound on the total cost of the solution is one, i.e. $C(\delta) \leq 1$. If $c_{ij}^{\ell} > 1$ for some operation O_{ij}, we set $c_{ij}^{\ell} = 1$ and make $\ell_{ij} = (u_{ij}(c_{ij}^{\ell} - 1) - \ell_{ij}(c_{ij}^{u} - 1))/(c_{ij}^{\ell} - c_{ij}^{u})$ to get an equivalent instance of the problem in which $c_{ij}^{\ell} \leq 1$ for all operations O_{ij}.

A Simple m-approximation Algorithm

We begin by showing that it is possible to compute in linear time an m-approximate solution for problem P1. This allows us to compute upper and lower bounds for the minimum makespan that we use in the following sections.

Let $U = \sum_{j=1}^{n}\sum_{i=1}^{\mu} u_{ij}$; note that U is a constant value. Consider an optimal solution (σ^*, δ^*) of problem P1 and let us use OPT to denote the optimum makespan. Let $P^* = \sum_{j=1}^{n}\sum_{i=1}^{\mu} p_{ij}^{\delta_{ij}^*} = \sum_{j=1}^{n}\sum_{i=1}^{\mu}(\ell_{ij} - u_{ij})\delta_{ij}^* + U$

be the sum of the processing times of all jobs in this optimum solution. Define $c_{ij} = c^{\ell}_{ij} - c^{u}_{ij}$, so $C(\delta^*) = \sum_{j=1}^{n}\sum_{i=1}^{\mu}(c_{ij}\delta^*_{ij} + c^u_{ij}) \leq 1$. Let $\tilde{x}$ be an optimal solution for the following linear program and $P(\tilde{x})$ the corresponding value.

$$\begin{array}{lll} \min & P = \sum_{j=1}^{n}\sum_{i=1}^{\mu}(\ell_{ij} - u_{ij})x_{ij} + U & \\ \text{s.t.} & \sum_{j=1}^{n}\sum_{i=1}^{\mu} x_{ij}c_{ij} \leq 1 - \sum_{j=1}^{n}\sum_{i=1}^{\mu} c^u_{ij}. & \\ & 0 \leq x_{ij} \leq 1 & j = 1, ..., n \text{ and } i = 1, ..., \mu. \end{array}$$

Observe that $P(\tilde{x}) \leq P^*$ and $C(\tilde{x}) \leq 1$. Note that this linear program is equivalent to the knapsack problem when the integrality constraints have been relaxed, and thus it can be solved in $O(n)$ time [50]. If we schedule all the jobs one after another with processing times as defined by $\tilde{x}$, we obtain a feasible schedule for the jobs $\mathcal{J}$ with makespan at most $P(\tilde{x})$. Since $OPT \geq P^*/m$ (this is the makespan of a schedule which leaves no idle times in the machines), the obtained schedule has cost at most 1 and makespan $P(\tilde{x}) \leq m \cdot OPT$. Therefore, $OPT \in [P(\tilde{x})/m, P(\tilde{x})]$, and by dividing all ℓ_{ij} and u_{ij} values by $P(\tilde{x})$, we get the following bounds for the optimum makespan:

$$1/m \leq OPT \leq 1. \tag{5.2}$$

Large, Medium, and Small Jobs

We partition the set of jobs in three groups in the following way. Let $P^*_j = \sum_{i=1}^{\mu} p_{ij}^{\delta^*_{ij}}$ be the sum of the processing times of the operations of job J_j according to an optimum solution (σ^*, δ^*). Let k and q be two constants, to be defined later, that depend on m, μ and ε. We define the set $\mathcal{L}$ of large jobs consisting of the k longest jobs according to δ^*. The next $(\frac{q\mu m^2}{\varepsilon} - 1)k$ longest jobs define the set $\mathcal{M}$ of medium jobs. The remaining jobs form the set $\mathcal{S}$ of small jobs. Even when we do not know an optimum solution (σ^*, δ^*) for the problem, we show in Section 5.3.1 how to select the set of long and medium jobs in polynomial time. Hence, we assume that we know sets $\mathcal{L}$, $\mathcal{M}$ and $\mathcal{S}$. An operation that belongs to a large, medium, or small job is called a large, medium, or small operation, respectively, regardless of its processing time.

In the following we simplify the input by creating a well-structured set of possible processing times. We begin by showing that it is possible to bound from above the possible processing times of small operations by $\frac{\varepsilon}{qk\mu m}$.

Lemma 5.3 *With no loss, for each small operation O_{ij} we can set $u_{ij} \leftarrow \bar{u}_{ij}$ and $c^u_{ij} \leftarrow \frac{c^{\ell}_{ij} - c^u_{ij}}{u_{ij} - \ell_{ij}}(u_{ij} - \bar{u}_{ij}) + c^u_{ij}$, where $\bar{u}_{ij} = \min\{u_{ij}, \frac{\varepsilon}{qk\mu m}\}$.*

Proof. By Inequality (5.2), the optimal makespan is at most 1, and therefore the sum of all processing times cannot be larger than m. Let p the length of the longest small job according to δ^*. By definition of $\mathcal{L}$ and $\mathcal{M}$, $|\mathcal{M} \cup \mathcal{L}| \cdot p = \frac{qk\mu m^2}{\varepsilon} p \leq \sum_{J_j \in \mathcal{M} \cup \mathcal{L}} P_j^* \leq m$, and so $p \leq \frac{\varepsilon}{qk\mu m}$. Therefore, the length of the schedule is not increased if the largest processing time u_{ij} of any small operation O_{ij} is set to be $\bar{u}_{ij} = \min\{u_{ij}, \frac{\varepsilon}{qk\mu m}\}$. It is easy to check that, in order to get an equivalent instance it is necessary to set $c_{ij}^u \leftarrow \frac{c_{ij}^\ell - c_{ij}^u}{u_{ij} - \ell_{ij}}(u_{ij} - \bar{u}_{ij}) + c_{ij}^u$ (this is the cost to process operation O_{ij} in time $\bar{u}_{ij}$). ■

In order to compute a $1+O(\varepsilon)$-approximate solution for P1 we show that it is sufficient to take into consideration only a constant number of different processing times and costs for medium jobs.

Lemma 5.4 *There exists a $(1 + 2\varepsilon)$-optimal schedule where each medium operation has processing time equal to $\frac{\varepsilon}{m|\mathcal{M}|\mu}(1 + \varepsilon)^i$, for $i \in \mathbb{N}$.*

Proof. Let us use (σ^*, δ^*) to denote an optimal solution. Let A be the set of medium operations for which $p_{ij}^{\delta_{ij}^*} \leq \frac{\varepsilon}{m|\mathcal{M}|\mu}$. Since $c_{ij}^u \leq c_{ij}^\ell$, we observe that by increasing the processing time of any operation then the corresponding cost cannot increase. Therefore, if we increase the processing times for the operations in A up to $\frac{\varepsilon}{m|\mathcal{M}|\mu}$, the makespan may potentially increase by at most $|A|\frac{\varepsilon}{m|\mathcal{M}|\mu} \leq \varepsilon/m$, and by Inequality (5.2) the schedule length may be increased by a factor of $1 + \varepsilon$. Now, consider the remaining medium operations and round every processing time $p_{ij}^{\delta_{ij}^*}$ up to the nearest value of $\frac{\varepsilon}{m|\mathcal{M}|\mu}(1+\varepsilon)^i$, for some $i \in \mathbb{N}$. Since each processing time is increased by a factor $1 + \varepsilon$, the value of an optimum solution potentially increases by the same factor $1 + \varepsilon$. ■

Recall that by Inequality (5.2), every processing time cannot be greater than 1. Then, by the previous lemma, the number of different processing times for medium operations can be bounded by $O(\log(m|\mathcal{M}|\mu)/\varepsilon)$ (clearly, the same bound applies to the number of different costs, since processing times and costs are closely related). Since there is a constant number of medium operations, there is also a constant number of choices for the values of their processing times and costs. We consider all these choices (thus, we also consider the case where the processing times $\bar{p}_{ij}$ and costs $\bar{c}_{ij}$ for the medium operations are chosen as in the $(1 + 2\varepsilon)$-optimal schedule of the previous lemma). In the following we show that when the medium operations are processed according to these $(\bar{p}_{ij}, \bar{c}_{ij})$-values, it is possible to compute a $1 + O(\varepsilon)$-approximate solution for P1 in polynomial time.

Clearly, a $1 + O(\varepsilon)$-approximate algorithm is obtained by considering all possible choices for processing times for the medium operations, and by returning the solution that is of minimum length. From now on, we assume, without loss of generality, that we know these $(\bar{p}_{ij}, \bar{c}_{ij})$-values for medium operations. In order to simplify the following discussion, for each medium operation O_{ij} we set $\ell_{ij} = u_{ij} = \bar{p}_{ij}$ and $c^{\ell}_{ij} = c^{u}_{ij} = \bar{c}_{ij}$ (this settings fix the processing time and cost of O_{ij} to be $\bar{p}_{ij}$ and $\bar{c}_{ij}$, respectively).

Computing Processing Times and Assigning Operations to Snapshots

A *relative schedule* for the large operations is an ordering of the starting and ending times of the operations. A feasible schedule S for the large operations *respects* a relative schedule R if the starting and ending times of the operations as defined by S are ordered as indicated in R (breaking ties in an appropriate way).

Fix a relative schedule R for the large operations. The starting and finishing times of the operations define a set of intervals that we call *snapshots*. Let $M(1), M(2), \ldots, M(g)$, be the snapshots defined by R. Snapshots $M(1)$ and $M(g)$ are empty. The number of snapshots g can be bounded by $g \leq 2k\mu + 1$.

Lemma 5.5 *The number of different relative schedules for the large jobs is at most* $(2ek)^{2k\mu}$.

Proof. The number of possible starting times for the operations of a large job J_j is at most the number of subsets of size μ that can be chosen from a set of $(2k\mu - 1)$ positions (there are $2\mu k - 1$ choices for the starting times of each operations of J_j). Since each large operation can end in the same snapshot in which it starts, the number of ways of choosing the starting and ending times of the operations of a large job is at most the number of subsets of size 2μ that can be chosen from a set of $2(2k\mu - 1)$ positions (we consider two positions associated to each snapshot, one to start and one to end an operation, but both positions denote the same snapshot). For each large job J_j there are at most $\binom{4\mu k-2}{2\mu}$ different choices of snapshots where operations of J_j can start and end. Since $\binom{4\mu k-2}{2\mu} = \frac{(4\mu k-2)(4\mu k-3)\ldots(4\mu k-2\mu-1)}{(2\mu)!} \leq \frac{(4\mu k)^{2\mu}}{(2\mu/e)^{2\mu}} = (2ek)^{2\mu}$ and the number of large jobs is k, then there are at most $(2ek)^{2k\mu}$ different relative schedules. ■

By the previous lemma the number of different relative schedules is bounded by a constant. Our algorithm considers all relative schedules for the large jobs. Therefore, one of them must be equal to the relative schedule R^* defined by some optimum solution. In the following we will show that when R^* is taken into consideration we are able to provide in polynomial time a $1 + O(\varepsilon)$-approximate solution.

Given a relative schedule R^* as described above, to obtain a solution for problem P1 that respects R^* we must select the processing times for the operations and schedule the medium and small operations within the snapshots defined by R^*. We use a linear program to compute the processing times and for deciding the snapshots where the small and medium operations must be placed. Then we find a feasible schedule for the operations in every snapshot.

To design the linear program, we create a variable x_{ij}^{ℓ} for each large operation O_{ij}; this variable defines the processing time and cost of operation O_{ij}. For convenience we define another variable x_{ij}^{u} with value $1 - x_{ij}^{\ell}$. The processing time of operation O_{ij} is then $x_{ij}^{\ell}\ell_{ij} + x_{ij}^{u}u_{ij}$, and its cost is $x_{ij}^{\ell}c_{ij}^{\ell} + x_{ij}^{u}c_{ij}^{u}$. Let α_{ij} be the snapshot where the large operation O_{ij} starts processing in the relative schedule R^* and let β_{ij} be the snapshot where it finishes processing.

Let $Free(R^*) = \{(\ell, h) \mid \ell = 1, ..., g,\ h = 1, ..., m$ and no long operation is scheduled by R^* in snapshot $M(\ell)$ on machine $h\}$ be the set of snapshots and machines not used by the large jobs in relative schedule R^*. We represent the processing times and costs for the medium and small operations as follows. For every job $J_j \in \mathcal{S} \cup \mathcal{M}$, let Λ_j be the set of tuples of the form $\Lambda_j = \{(s_1, s_2, \ldots, s_\mu) \mid 1 \leq s_1 \leq s_2 \leq \ldots, \leq s_\mu \leq g$, and $(s_i, m_{ij}) \in Free(R^*)$, for all $i = 1, \ldots, \mu\}$. Set Λ_j defines the set of μ-snapshots where it is possible to put the operations of job J_j.

Let $\Delta = \{(\delta_1, \delta_2, \ldots, \delta_\mu) \mid \delta_k \in \{0, 1\}$ for all $k = 1, \ldots, \mu\}$. For each job $J_j \in \mathcal{S} \cup \mathcal{M}$ we define a set of at most $(2g)^\mu$ variables $x_{j,(s,\delta)}$, where $s \in \Lambda_j$ and $\delta \in \Delta$. To explain the meaning of these variables, let us define

$$x_{ij}(w, 1) = \sum_{(s,\delta)\in\Lambda_j\times\Delta \mid s_i=w,\ \delta_i=1} x_{j,(s,\delta)},$$

$$x_{ij}(w, 0) = \sum_{(s,\delta)\in\Lambda_j\times\Delta \mid s_i=w,\ \delta_i=0} x_{j,(s,\delta)},$$

for each operation i, job J_j, and snapshot w. Given a set of values for the variables $x_{j,(s,\delta)}$, they define the processing times and an assignment of medium and small operations to snapshots in which the amount of time

that operation O_{ij} is processed within snapshot $M(w)$ is $x_{ij}(w,1) \cdot \ell_{ij} + x_{ij}(w,0) \cdot u_{ij}$, and the fraction of O_{ij} that is assigned to this snapshot is $x_{ij}(w,0) + x_{ij}(w,1)$.

For each snapshot $M(\ell)$ we use a variable t_ℓ to denote its length. For any $(\ell,h) \in Free(R^*)$, we define the load $L_{\ell,h}$ on machine h in snapshot $M(\ell)$ as the total processing time of the operations from small and medium jobs that get assigned to h during $M(\ell)$, i.e.,

$$L_{\ell,h} = \sum_{J_j \in \mathcal{S} \cup \mathcal{M}} \sum_{(s,\delta) \in \Lambda_j \times \Delta} \left(\sum_{\substack{i=1 \\ s_i=\ell, m_{ij}=h \\ \delta_i=1}}^{\mu} x_{j,(s,\delta)} \ell_{ij} + \sum_{\substack{i=1 \\ s_i=\ell, m_{ij}=h \\ \delta_i=0}}^{\mu} x_{j,(s,\delta)} u_{ij} \right).$$

Let the cost be

$$C = \sum_{J_j \in \mathcal{S} \cup \mathcal{M}} \sum_{(s,\delta) \in \Lambda_j \times \Delta} \left(\sum_{\substack{i=1 \\ \delta_i=1}}^{\mu} x_{j,(s,\delta)} c^{\ell}_{ij} + \sum_{\substack{i=1 \\ \delta_i=0}}^{\mu} x_{j,(s,\delta)} c^{u}_{ij} \right) + \sum_{J_j \in \mathcal{L}} \sum_{i=1}^{\mu} (x^{\ell}_{ij} c^{\ell}_{ij} + x^{u}_{ij} c^{u}_{ij}).$$

We use the following linear program $LP(R^*)$ to determine processing times and to allocate operations to snapshots.

$$\begin{array}{lll}
\min & T = \sum_{\ell=1}^{g} t_\ell & \\
\text{s.t.} & C \leq 1, & (1) \\
& \sum_{\ell=\alpha_{ij}}^{\beta_{ij}} t_\ell = x^{\ell}_{ij} \ell_{ij} + x^{u}_{ij} u_{ij}, \quad J_j \in \mathcal{L},\ i = 1, \ldots, \mu, & (2) \\
& x^{\ell}_{ij} + x^{u}_{ij} = 1, \quad J_j \in \mathcal{L},\ i = 1, \ldots, \mu, & (3) \\
& \sum_{(s,\delta) \in \Lambda_j \times \Delta} x_{j,(s,\delta)} = 1, \quad J_j \in \mathcal{S} \cup \mathcal{M}, & (4) \\
& L_{\ell,h} \leq t_\ell, \quad (\ell,h) \in Free(R^*), & (5) \\
& x^{\ell}_{ij}, x^{u}_{ij} \geq 0, \quad J_j \in \mathcal{L},\ i = 1, \ldots, \mu, & (6) \\
& x_{j,(s,\delta)} \geq 0, \quad J_j \in \mathcal{S} \cup \mathcal{M},\ (s,\delta) \in \Lambda_j \times \Delta, & (7) \\
& t_\ell \geq 0, \quad \ell = 1, \ldots, g. & (8)
\end{array}$$

In this linear program the value of the objective function T is the length of the schedule, which we want to minimize. Constraint (1) ensures that the total cost of the solution is at most one. Condition (2) requires that the

total length of the snapshots where a long operation is scheduled is exactly equal to the length of the operation. Constraint (4) assures that every small and medium operation is completely assigned to snapshots, while constraint (5) checks that the total load of a machine h during some snapshot ℓ does not exceed the length of the snapshot.

Let (σ^*, δ^*) denote an optimal schedule when the processing times and costs of medium jobs are fixed as described in the previous section.

Lemma 5.6 *The optimal solution of $LP(R^*)$ has value no larger than the makespan of (σ^*, δ^*).*

Proof. We only need to show that for any job $J_j \in \mathcal{S} \cup \mathcal{M}$ there is a feasible solution of $LP(R^*)$ that schedules all operations O_{ij} in the same snapshots and with the same processing times and costs as the optimum schedule (σ^*, δ^*).

For any operation O_{ij}, let $p_{ij}^{\delta_{ij}^*}(w)$ be the amount of time that O_{ij} is assigned to snapshot $M(w)$ by the optimum schedule (σ^*, δ^*). Let $x_{ij}^*(w,1) = \delta_{ij}^* p_{ij}^{\delta_{ij}^*}(w)/p_{ij}^{\delta_{ij}^*}$ and $x_{ij}^*(w,0) = (1-\delta_{ij}^*)p_{ij}^{\delta_{ij}^*}(w)/p_{ij}^{\delta_{ij}^*}$. The processing time $p_{ij}^{\delta_{ij}^*}(w)$ and cost $c_{ij}^{\delta_{ij}^*}(w)$ of O_{ij} can be written as $x_{ij}^*(w,1)\cdot\ell_{ij} + x_{ij}^*(w,0)\cdot u_{ij}$ and $x_{ij}^*(w,1)\cdot c_{ij}^{\ell} + x_{ij}^*(w,0)\cdot c_{ij}^{u}$, respectively.

Now we show that there is a feasible solution of $LP(R^*)$ such that
(*i*) $x_{ij}^*(w,1) = \sum_{(s,\delta)\in\Lambda_j\times\Delta \mid s_i=w,\ \delta_i=1} x_{j,(s,\delta)}$ and
(*ii*) $x_{ij}^*(w,0) = \sum_{(s,\delta)\in\Lambda_j\times\Delta \mid s_i=w,\ \delta_i=0} x_{j,(s,\delta)}$.

Therefore, for this solution $p_{ij}^{\delta_{ij}^*}(w)$ and $c_{ij}^{\delta_{ij}^*}(w)$ are linear combinations of the variables $x_{j,(s,\delta)}$. We assign values to the variables $x_{j,(s,\delta)}$ as follows.

1. For each job $J_j \in \mathcal{S} \cup \mathcal{M}$ do
2. (a) Compute $S_j = \{(i,\ell,d) \mid x_{ij}^*(\ell,d) > 0, i = 1,\ldots,\mu,\ \ell = 1,\ldots,g,\ d = 0,1\}$.
3. (a) If $S_j = \emptyset$ then exit.
4. (a) Let $f = \min\{x_{ij}^*(\ell,d) \mid (i,\ell,d) \in S_j\}$ and let i', ℓ', and d' be such that $x_{i'j}^*(\ell',d') = f$.
5. (a) Let $s = (s_1, s_2, \ldots, s_\mu) \in \Lambda_j$ and $\delta = (d_1, d_2, \ldots, d_\mu) \in \Delta$ be such that $s_{i'} = \ell'$, $d_{i'} = d'$ and $x_{ij}^*(s_i, d_i) > 0$, for all $i = 1,\ldots,\mu$.
6. (a) $x_{j,(s,\delta)} \leftarrow f$

7. (a) $x^*_{ij}(s_i, d_i) \leftarrow x^*_{ij}(s_i, d_i) - f$ for all $i = 1, 2, \ldots, \mu$.

8. (a) Go back to step 2.

With this assignment of values to the variables $x_{j,(s,\delta)}$, equations (i) and (ii) hold for all jobs $J_j \in \mathcal{S} \cup \mathcal{M}$, all operations i, and all snapshots w. Therefore, this solution of $LP(R^*)$ schedules the jobs in the same positions and with the same processing times as the optimum schedule. ■

Finding a Feasible Schedule

Linear program $LP(R^*)$ has at most $1+2\mu k+n-k+mg-\mu k$ constraints. By condition (3) every large operation O_{ij} must have at least one of its variables x^ℓ_{ij} or x^u_{ij} set to a non-zero value. By condition (4) every small and medium job J_j must have at least one of its variables $x_{j,(s,\delta)}$ set to a positive value. Furthermore, there is at least one snapshot ℓ for which $t_\ell > 0$. Since in any basic feasible solution of $LP(R^*)$ the number of variables that receive positive values is at most equal to the number of rows of the constraint matrix, then in a basic feasible solution there are at most mg variables with fractional values. This means that in the schedule defined by a basic feasible solution of $LP(R^*)$ at most mg medium and small jobs receive fractional assignments, and therefore, there are at most that many jobs from $\mathcal{M} \cup \mathcal{S}$ for which at least one operation is assigned to at least two different snapshots. Let $\mathcal{F}$ be the set of jobs that received fractional assignments. For the time being let us forget about those jobs. We show later how to schedule them.

Note that this schedule is still not feasible because there might be ordering conflicts among the small or medium operations assigned to a snapshot. To eliminate these conflicts, we first remove the set $\mathcal{V}$ of jobs from $\mathcal{M} \cup \mathcal{S}$ which have at least one operation with processing time larger than $\frac{\varepsilon}{\mu^3 m^2 g}$ according to the solution of the linear program. Since the sum of the processing times computed by the linear program is at most m, then $|\mathcal{V}| \leq \mu^3 m^3 g/\varepsilon$, so we remove only a constant number of jobs.

Let $\mathcal{O}(\ell)$ be the set of operations from $\mathcal{M} \cup \mathcal{S}$ that remain in snapshot $M(\ell)$. Let $p_{max}(\ell)$ be the maximum processing time among the operations in $\mathcal{O}(\ell)$. Every snapshot $M(\ell)$ defines an instance of the classical job shop problem, since the solution of $LP(R^*)$ determined the processing time of every operation. Hence, we can use Sevastianov's algorithm [74] to find in $O(n^2\mu^2m^2)$ time a feasible schedule for the operations in $\mathcal{O}(\ell)$; this schedule has length at most $\bar{t}_\ell = t_\ell + \mu^3 m p_{max}(\ell)$. We must increase the length of every snapshot $M(\ell)$ to $\bar{t}_\ell$ to accommodate the schedule produced by Sevastianov's algorithm. Summing up all these snapshot enlargements, we

get a solution of length at most the value of the solution of $LP(R^*)$ plus ε/m.

It remains to show how to schedule the jobs $\mathcal{V} \cup \mathcal{F}$. First we choose the value for parameter q:

$$q = 6\mu^4 m^3/\epsilon. \tag{5.3}$$

Then the number of jobs in $\mathcal{V} \cup \mathcal{F}$ is

$$|\mathcal{V} \cup \mathcal{F}| \leq \mu^3 m^3 g/\epsilon + mg \leq qk. \tag{5.4}$$

Lemma 5.7 *Consider solution (σ^*, δ^*). Let $P_j^* = \sum_{i=1}^{\mu} p_{ij}^{\delta_{ij}^*}$ denote the length of job J_j according to δ^*. There exists a positive constant k such that if the set of large jobs contains the k jobs J_j with the largest P_j^* value, then $\sum_{J_j \in \mathcal{V} \cup \mathcal{F}} P_j^* \leq \varepsilon/m$.*

Proof. Let us sort the jobs J_j non-increasingly by P_j^* values, and assume for convenience that $P_1^* \geq P_2^* \geq ... \geq P_n^*$. Partition the jobs into groups $G_1, G_2, ..., G_d$ as follows $G_i = \{J_{(1+qb)^{i-1}+1}, ..., J_{(1+qb)^i}\}$, and let $P(G_j) = \sum_{J_i \in G_j} P_j^*$. Let $G_{\rho+1}$ be the first group for which $P(G_{\rho+1}) \leq \varepsilon/m$. Since $\sum_{J_j \in \mathcal{J}} P_j^* \leq m$ and $\sum_{i=1}^{\rho} P(G_i) > \rho\varepsilon/m$ then $\rho < \frac{m^2}{\varepsilon}$. We choose $\mathcal{L}$ to contain all jobs in groups G_1 to G_ρ, and so $k = (1+q)^\rho$. Since $|\mathcal{V} \cup \mathcal{F}| \leq qk$, then with this choice of $\mathcal{L}$, $|G_{\rho+1}| = qk \geq |\mathcal{V} \cup \mathcal{F}|$ and therefore $\sum_{J_j \in \mathcal{V} \cup \mathcal{F}} P_j^* \leq \sum_{J_j \in G_\rho} P_j^* \leq \varepsilon/m$. ■

We choose the set of large jobs by considering all subsets of k jobs, for all integer values $1 \leq k \leq \frac{m^2}{\varepsilon}$. For each choice of k the set of medium jobs is obtained by considering all possible subsets of $\left(q\mu m^2/\varepsilon - 1\right) k$ jobs. Since there is only a polynomial number of choices for large and medium jobs, we require only a polynomial number of attempts before detecting the set of large and medium jobs as defined by an optimum solution.

The processing time of every small operation O_{ij} in $\mathcal{V} \cup \mathcal{F}$, is chosen as $p_{ij} = u_{ij}$ and so its cost is $c_{ij} = c_{ij}^u$. Furthermore, recall that we are assuming that each medium operation O_{ij} is processed in time $p_{ij} = \bar{p}_{ij}$ and cost $c_{ij} = \bar{c}_{ij}$ (see section 5.3.1). Let $P_j = \sum_{i=1}^{\mu} p_{ij}$. Then $\sum_{J_j \in \mathcal{V} \cup \mathcal{F}} P_j = \sum_{J_j \in \mathcal{M} \cap (\mathcal{V} \cup \mathcal{F})} P_j + \sum_{J_j \in \mathcal{S} \cap (\mathcal{V} \cup \mathcal{F})} P_j$. Note that by Lemma 5.3 and inequality (5.4), $\sum_{J_j \in \mathcal{S} \cap (\mathcal{V} \cup \mathcal{F})} P_j \leq qk\mu\varepsilon/(qk\mu m) = \varepsilon/m$. By the arguments of section 5.3.1, $\bar{p}_{ij} \leq \max\{\, p_{ij}^{\delta_{ij}^*}(1+\varepsilon), \varepsilon/(m|\mathcal{M}|\mu)\,\}$ and, therefore, $\sum_{J_j \in \mathcal{M} \cap (\mathcal{V} \cup \mathcal{F})} P_j \leq \sum_{J_j \in \mathcal{M} \cap (\mathcal{V} \cup \mathcal{F})} P_j^*(1+\varepsilon) + \varepsilon/m \leq \varepsilon(2+\varepsilon)/m$, by Lemma 5.7. Therefore, we can schedule the jobs from $\mathcal{V} \cup \mathcal{F}$ one after the other at the end of the schedule without increasing too much the length of the schedule.

Theorem 5.3 *For any fixed m and μ, there exists a polynomial-time approximation scheme for P1 that computes a feasible schedule with makespan at most $(1+\epsilon)$ times the optimal makespan and cost not greater than κ, for any fixed $\epsilon > 0$.*

5.3.2 Problem P2 with Continuous Processing Times

Given a value $\tau \geq 0$, let (σ^*, δ^*) be an optimum solution for problem P2, if such a solution exists. Note that for some values of τ problem P2 might not have a solution. Given a value $\varepsilon > 0$, we present an algorithm that either finds a schedule for $\mathcal{J}$ of length at most $(1+\varepsilon)\tau$ and cost at most $C(\delta^*)$, or it decides that a schedule of length at most τ does not exist.

We embed the PTAS for P1, described in the previous section, within a binary search procedure. Assume, without loss of generality, that the input instance has integral data. Let $C_\ell = \sum_{j=1}^{n} \sum_{i=1}^{\mu} c_{ij}^{\ell}$ and $C_u = \sum_{j=1}^{n} \sum_{i=1}^{\mu} c_{ij}^{u}$. Clearly, the value of the optimum solution for problem P2 is in the interval $[C_u, C_\ell]$. Thus we can use $UB := C_\ell$ and $LB := C_u$ as initial upper and lower bounds, respectively, for the binary search; in each iteration the algorithm performs the following steps:

a) it uses the PTAS for P1 to find a schedule of length $T(H)$ at most $1+\varepsilon$ times the minimum length and cost at most $H = (LB+UB)/2$;

b) if $T(H) \leq (1+\varepsilon)\tau$ then update the upper bound to H, otherwise update the lower bound to $H+1$.

The algorithm terminates when $LB = UB$, and outputs the best schedule found. A straightforward proof by induction shows that, throughout the execution of the binary search, the inequality $LB \leq C(\delta^*)$ always holds, and if no schedule with length at most $(1+\varepsilon)\tau$ is found then a schedule of length at most τ does not exist. After $\log_2(C_\ell - C_u)$ iterations the search converges and we have obtained PTAS for P2.

5.3.3 Problem P3 with Continuous Processing Times

Our PTAS for problem P1 can be modified to work also for P3. It is enough to provide the following observations.

For each operation O_{ij}, let $d_{ij} = \min_{0 \leq \delta_{ij} \leq 1} \{p_{ij}^{\delta_{ij}} + c_{ij}^{\delta_{ij}}\} = \min\{\ell_{ij} + c_{ij}^{\ell}, u_{ij} + c_{ij}^{u}\}$. For every job J_j let $d_j = \sum_{i=1}^{\mu} d_{ij}$. We partition the set of jobs into large $\mathcal{L}$ and small $\mathcal{S}$ jobs. Differently from before, there is no set of medium job and $\mathcal{L}$ and $\mathcal{S}$ can be computed in linear time. The set $\mathcal{L}$ of large

jobs includes the k jobs of largest value d_j, where k is a constant chosen as in Lemma 5.7 to ensure that $\sum_{J_i \in \mathcal{L}} d_i \leq \varepsilon/m$.computed by using similar arguments as before. By using normalization ($c_{ij} \leftarrow c_{ij}/\alpha$), we assume, without loss of generality, that the objective function is $\min T(\sigma, \delta) + C(\delta)$. Let $T^* + C^*$ be the optimum objective function value. It is easy to see that $T^* + C^* \leq D$, where $D = \sum_j d_j$. Furthermore, $T^* + C^* \geq D/m$ since

$$\begin{aligned} T^* + C^* &\geq \frac{1}{m} \sum_{ij} \left[\delta^*_{ij} \ell_{ij} + (1 - \delta^*_{ij}) u_{ij}\right] + \sum_{ij} \left[\delta^*_{ij} c^{\ell}_{ij} + (1 - \delta^*) c^{u}_{ij}\right] \\ &\geq \frac{1}{m} \sum_{ij} \left[\delta^*_{ij} (\ell_{ij} + c^{\ell}_{ij}) + (1 - \delta^*_{ij})(u_{ij} + c^{u}_{ij})\right] \\ &\geq \frac{1}{m} \sum_{ij} \left[\delta^*_{ij} d_{ij} + (1 - \delta^*_{ij}) d_{ij}\right] = \frac{D}{m}. \end{aligned}$$

By dividing all execution times and costs by D, we may assume that $D = 1$ and $\frac{1}{m} \leq T^* + C^* \leq 1$.

The linear program $LP(R)$ has to be modified as follows. The objective function is changed to $\min \sum_{\ell=1}^{g} t_\ell + C$ and we discard constraint (1). Again, the number of fractional values can be bounded by a constant and the algorithm is as before.

We observe that the most time consuming part of this approach is solving the linear program. However, since we want to get an approximate solution, it is not necessary to find an optimum solution for $LP(R)$, an approximate solution would suffice. To speed up our algorithm to run in linear time we can use the ideas described in Section 5.3.4.

5.3.4 Problem P3 with Discrete Processing Times

In the case of discretely controllable processing times, the possible processing times and costs of an operation O_{ij} are specified by a set of values $\Delta_{ij} = \{\delta_1, \delta_2, \ldots, \delta_{w(i,j)}\}$, $0 \leq \delta_k \leq 1$ for all $k = 1, 2, \ldots, w(i, j)$. When the processing time for operation O_{ij} is $p^{\delta_k}_{ij} = \delta_k \ell_{ij} + (1 - \delta_k) u_{ij}$, the cost is equal to $c^{\delta_k}_{ij} = \delta_k c^{\ell}_{ij} + (1 - \delta_k) c^{u}_{ij}$.

For each operation O_{ij}, let $d_{ij} = \min_{\delta_{ij} \in \Delta_{ij}} \{p^{\delta_{ij}}_{ij} + c^{\delta_{ij}}_{ij}\}$. For every job J_j let $d_j = \sum_{i=1}^{\mu} d_{ij}$, and let $D = \sum_j d_j$. We partition the set of jobs into large $\mathcal{L}$ and small $\mathcal{S}$ jobs, where the set $\mathcal{L}$ includes the k jobs with the largest d_j values, and k is a constant computed as in Lemma 5.7 so that the set T containing the qk jobs with the next d_j values has

$\sum_{J_j \in T} d_j \leq \varepsilon/m$. The set of large jobs can be computed in $O(n\mu|\Delta_{\max}|)$ time, where $|\Delta_{\max}| = \max_{ij} |\Delta_{ij}|$.

By dividing all $c_{ij}^{\delta_{ij}}$ values by parameter α, we can assume without loss of generality that the objective function for problem P3 is: $\min T(\sigma, \delta) + C(\delta)$. Let $p_{ij}^{\delta_{ij}^*}$ and $c_{ij}^{\delta_{ij}^*}$ be the processing time and cost of operation O_{ij} in an optimal solution. Let F^* be the value of an optimal solution for P3. It is easy to see that $F^* \leq D$ and $F^* \geq \frac{1}{m} \sum_{ij} p_{ij}^{\delta_{ij}^*} + \sum_{ij} c_{ij}^{\delta_{ij}^*} \geq \frac{D}{m}$. By dividing all execution times and costs by D, we may assume that $D = 1$ and

$$\frac{1}{m} \leq F^* \leq 1. \tag{5.5}$$

The following lemma shows that with $1 + 2\varepsilon$ loss we can reduce the number of different costs and processing times for each operation (proof in appendix).

Lemma 5.8 *With $1 + 2\varepsilon$ loss, we assume that $|\Delta_{ij}| = O(\log n)$ for every O_{ij}.*

Proof. To prove this claim, divide the interval $[0, 1]$ into b subintervals as follows, $I_1 = [0, \frac{\varepsilon}{\mu nm}], I_2 = (\frac{\varepsilon}{\mu nm}, \frac{\varepsilon}{\mu nm}(1+\varepsilon)], ..., I_b = (\frac{\varepsilon}{\mu nm}(1+\varepsilon)^{b-1}, 1]$, where b is the largest integer such that $\frac{\varepsilon}{\mu nm}(1+\varepsilon)^{b-1} < 1$. Clearly $b = O(\log n)$. We say that d is a *choice* for operation O_{ij} if $d \in \Delta_{ij}$. For each operation O_{ij}, partition the set of choices Δ_{ij} into b groups $g_1, g_2, ..., g_b$, such that $d \in \Delta_{ij}$ belongs to group g_h if and only if c_{ij}^d falls in interval I_h, $h \in \{1, ..., b\}$.

For each group take the choice (if any) with the lowest processing time and delete the others. The new set of choices has at most $O(\min\{|\Delta_{ij}|, \log n\})$ elements and by using arguments similar to those used in the proof of Lemma 5.4 we can prove that with this transformation the cost of an optimum solution can be up to $1 + 2\varepsilon$ times the optimum value for the original problem. The transformed instance can be computed in $O(n\mu|\Delta_{\max}|)$ time. ■

By using arguments similar to those used to prove Lemma 5.8 we can obtain, with $1 + 2\varepsilon$ loss, a new instance with $O(\log k)$ different costs and processing times for each large operation. Since there is a constant number of large operations, there is only a constant number of possible assignments of costs and processing times for them.

By trying all possible assignments of cost and processing times, we can find for each large operation O_{ij} a processing time $\bar{p}_{ij}$ and cost $\bar{c}_{ij}$ such that

$\bar{p}_{ij} \leq \max\{\, p_{ij}^{\delta_{ij}^*}(1+\varepsilon), \varepsilon/(mk\mu)\,\}$ and $\bar{c}_{ij} \leq c_{ij}^{\delta_{ij}^*}$. Let us use the same definition of relative schedule given for the continuous case. Let R denote a relative schedule that respects the ordering of the large operations in some optimal schedule. For each small job J_j we define a set of at most $O((g \log n)^\mu)$ variables $x_{j,(s,\delta)}$, where $s \in \Lambda_j$ and $\delta \in \Delta_j = \{(\delta_{1j}, \delta_{2j}, \ldots, \delta_{\mu j}) \mid \delta_{ij} \in \Delta_{ij}$ for all $i = 1, \ldots, \mu\}$.

As in the continuous case we define a linear program $LP'(R)$ to compute the processing times and snapshots for the small jobs. $LP'(R)$ is obtained from $LP(R)$ by deleting constraints (1), (3), and (6), and considering the following changes. Variable $x_{j,(s,\delta)}$ takes value $0 \leq f \leq 1$ to indicate that a fraction f of operation O_{ij}, $i = 1, \ldots, \mu$ is scheduled in snapshot s_i with processing time $p_{ij}^{\delta_i}$. Let C be the cost function, i.e., $C = \sum_{J_j \in \mathcal{S}} \sum_{(s,\delta) \in \Lambda_j \times \Delta} \sum_{i=1}^{\mu} x_{j,(s,\delta)} c_{ij}^{\delta_k} + \sum_{J_j \in \mathcal{L}} \sum_{i=1}^{\mu} \bar{c}_{ij}$. The objective function is now to minimize $\sum_{\ell=1}^{g} t_\ell + C$. Constraint (2) is replaced with

$$\sum_{\ell=\alpha_{ij}}^{\beta_{ij}} t_\ell = \bar{p}_{ij}, \quad \text{for all } J_j \in \mathcal{L}, i = 1, \ldots, \mu.$$

As in Lemma 5.6, we can prove that any optimum solution of problem P3 is a feasible solution for $LP'(R)$. The rest of the algorithm is as that described in the previous subsection 5.3.1.

By using interior point methods to solve the linear program, we get a total running time for the above algorithm that is polynomial in the input size [2]. It is easy to check that similar results can be obtained if, instead of finding the optimum solution of the linear program, we solve it with a given accuracy $\varepsilon > 0$. In the following section we show that we can solve approximately the linear program in $O(n|\Delta_{\max}|)$ time. Therefore, for every fixed m, μ and ϵ, all computations (including Sevastianov's algorithm [44]) can be carried out in time $O(n|\Delta_{\max}| + n \min\{\log n, |\Delta_{\max}|\} \cdot f(\varepsilon, \mu, m))$, where $f(\varepsilon, \mu, m)$ is a function that depends on ε, μ and m. This running time is linear in the size of the input.

Theorem 5.4 *For any fixed m and μ, there exists a linear time approximation scheme for P3 with discretely controllable processing times.*

Approximate Solution of the Linear Program

In this section we show how to find efficiently a solution for $LP'(R)$ of value no more than $1 + O(\epsilon)$ times the value of the optimum solution for problem P3. To find an approximate solution for the linear program we rewrite it as a

convex block-angular resource-sharing problem, and then use the algorithm of [28] to solve it with a given accuracy. A *convex block-angular resource sharing problem* has the form:
$\lambda^* = \min\left\{\lambda \left| \sum_{j=1}^{K} f_i^j(x^j) \leq \lambda, \forall i = 1,\ldots,N, \text{and } x^j \in \mathcal{B}^j, j = 1,\ldots,K \right.\right\}$, where $f_i^j : \mathcal{B}^j \to \mathbb{R}^+$ are N non-negative continuous convex functions, and $\mathcal{B}^j$ are disjoint convex compact nonempty sets called *blocks*. The algorithm in [28] finds a $(1+\rho)$-approximate solution for this problem for any $\rho > 0$ in $O(N(\rho^{-2}\ln\rho^{-1} + \ln N)(N\ln\ln(N/\rho) + KF))$ time, where F is the time needed to find a ρ-approximate solution to the problem: $\min\left\{\sum_{i=1}^{N} p_i f_i^j(x^j) \left| x^j \in \mathcal{B}^j \right.\right\}$, for some vector $(p_1,\ldots,p_N) \in \mathbb{R}^N$.

We can write $LP'(R)$ as a convex block-angular resource sharing problem as follows. First we guess the value V of an optimum solution of problem P3, and add the constraint: $\sum_{\ell=1}^{g} t_\ell + C + 1 - V \leq \lambda$, to the linear program, where λ is a non-negative value. Since $1/m \leq V \leq 1$, we can use binary search on the interval $[1/m, 1]$ to guess V with a given accuracy $\varepsilon > 0$. This search can be completed in $O(\log(\frac{1}{\varepsilon}\log m))$ iterations by doing the binary search on the values: $\frac{1}{m}(1+\varepsilon), \frac{1}{m}(1+\varepsilon)^2, ..., 1$. We replace constraint (5) of $LP'(R)$ by

(5') $\quad L_{\ell,h} + 1 - t_\ell \leq \lambda, \quad$ for all $(\ell,h) \in Free(R)$

where

$$L_{\ell,h} = \sum_{J_j \in \mathcal{S}} \sum_{(s,\delta)\in\Sigma_j\times\Delta} \sum_{\substack{q=1 \\ s_q=\ell, m_{qj}=h}}^{\mu} x_{j,(s,\delta)} p_{qj}^{\delta_q}.$$

This new linear program, that we denote as $LP(R,V,\lambda)$, has the above block-angular structure. The blocks $\mathcal{B}^j = \{x_{j,(s,\delta)} \mid (s,\delta) \in \Sigma_j \times \Delta$, constraints (4) and (8) hold$\}$, for $J_j \in \mathcal{S}$ are $(g|\Delta_{\max}|)^\mu$-dimensional *simplicies*. The block $\mathcal{B}^0 = \{< t_1, t_2, .., t_g > |$constraints (2) and (9) hold$\}$ has constant dimension. Let $f_{\ell,h} = L_{\ell,h} + 1 - t_\ell$. Since $t_\ell \leq V \leq 1$, these functions are non-negative.

For every small job J_j, let $f_0^j(x^j) = \sum_{(s,\delta)\in\Lambda_j\times\Delta} \sum_{i=1}^{\mu} x_{j,(s,\delta)} c_{ij}^{\delta_i}$. For every $(\ell,h) \in Free(R)$, let $f_{\ell h}^j(x^j) = \sum_{(s,\delta)\in\Lambda_j\times\Delta} \sum_{\substack{q=1 \\ s_q=\ell, m_{qj}=h}}^{\mu} x_{j,(s,\delta)} p_{qj}^{\delta_q}$. For every $x^0 \in \mathcal{B}^0$ let $f_0^0(x^0) = \sum_{J_j\in\mathcal{L}} \sum_{i=1}^{\mu} c_{ij} + \sum_{\ell=1}^{g} t_\ell + 1 - V$, and for every $(\ell,h) \in Free(R)$, let $f_{\ell h}^0(x^0) = 1 - t_\ell$. All these functions are

nonnegative. Now $LP(R, V, \lambda)$ is to find the smallest value λ such that

$$\begin{aligned} \textstyle\sum_{J_j \in \mathcal{S}} \quad & f_0^j(x^j) + f_0^0(x^0) \leq \lambda, \qquad \text{for all } x^k \in \mathcal{B}^k, \\ \textstyle\sum_{J_j \in \mathcal{S}} \quad & f_{\ell h}^j(x^j) + f_{\ell h}^0(x^0) \leq \lambda, \quad \text{for all } (\ell, h) \in Free(R) \text{ and } x^k \in \mathcal{B}^k. \end{aligned}$$

Using the algorithm in [28], a $1+\rho$, $\rho > 0$ approximation for the problem can be obtained by solving on each block $\mathcal{B}^j$ a constant number of block optimization problems of the form: $\min\{p^T f^j(x) \mid x \in \mathcal{B}^j\}$, where p is a $(g|\Delta_{\max}| + 1)$-dimensional positive price vector, and f^j is a $(g|\Delta_{\max}| + 1)$-dimensional vector whose components are the functions $f_0^j, f_{\ell h}^j$.

Note that $\mathcal{B}^0$ has constant dimension, and the corresponding block optimization problem can be solved in constant time. But, the blocks $\mathcal{B}^j$ for $J_j \in \mathcal{S}$ do not have a constant dimension. To solve the block optimization problem on $\mathcal{B}^j$ we must find a snapshot where to place each operation of J_j and determine its processing time, so that the total cost plus processing time of all operations times the price vector is minimized. To choose the snapshots we select for each operation the snapshot in which the corresponding component of the price vector is minimum. Then we select for each O_{ij} the value δ_{ij} that minimizes its cost plus processing time. This can be done in $O(|\Delta_{\max}|)$ time for each block, so the algorithm of [28] finds a feasible solution for $LP(R, V, 1+\rho)$ in $O(nw)$ time. Linear program $LP(R, V, 1+\rho)$ increases the length of each snapshot by ρ, and therefore the total length of the solution is $V + g\rho \leq (1 + 2\varepsilon)V^*$, for $\rho = \frac{\epsilon}{mg}$, where V^* is the optimal solution value.

There is a problem with this method: we cannot guarantee that the solution found by the algorithm is basic feasible. Hence it might have a large number of fractional assignments. In the following we show that the number of fractional assignments is $O(n)$. Since the number of fractional assignments is $O(n)$, using the rounding technique described in [44], we can obtain in linear time a new feasible solution with only a constant number of fractional assignments.

The algorithm in [28] works by choosing a starting solution $x_0 \in \mathcal{B}^j$ and then it repeats the following three steps for at most $O(mg \log(mg))$ times: (step 1) use a deterministic or randomized procedure to compute a price vector p; (step 2) use a block solver to compute an optimal solution of each block problem, (step 3) replace the current solution by a convex combination of the previous solutions. By starting from a solution x_0 in which every vector $x_0^j \in \mathcal{B}^j$ is an integer vector, for $j \neq 0$, we get at the end at most $O(n \cdot mg \log(mg))$ fractional assignments. To achieve the promised running

time we additionally need that $\lambda(x_0) \leq c\lambda^*$ [28], where c is a constant and $\lambda(x_0)$ is the value of λ corresponding to x_0. This is accomplished as follows.

For convenience, let us rename jobs such that $J_1, ..., J_{\bar{n}}$ are the small jobs, where $\bar{n} = n - k$. Choose the processing time $p_{ij}^{\delta_{ij}}$ and cost $c_{ij}^{\delta_{ij}}$ for every small operation O_{ij} so that $d_{ij} = p_{ij}^{\delta_{ij}} + c_{ij}^{\delta_{ij}}$. Put the small jobs one after the other in the last snapshot. Set $t_g = \sum_{J_j \in \mathcal{S}} \sum_{i=1}^{\mu} p_{ij}^{\delta_{ij}}$. The large operations are scheduled as early as possible, according to the optimal relative schedule R. Set each $t_\ell \in \{t_1, t_2, ..., t_{g-1}\}$ equal to the maximum load of snapshot ℓ according to the described schedule. By inequality (5.5), we know that $\sum_{J_j \in \mathcal{S}} \sum_{i=1}^{\mu} d_{ij} \leq 1$. Furthermore, we have $\sum_{\ell=1}^{g-1} t_\ell + \sum_{J_j \in \mathcal{L}} \sum_{i=1}^{\mu} c_{ij} \leq V$, since by construction $\sum_{\ell=1}^{g-1} t_\ell$ cannot be greater than the optimal length and the costs of large operations are chosen according to the optimal solution. Hence, $\sum_{\ell=1}^{g} t_\ell + C \leq 1 + V$, and $\sum_{\ell=1}^{g} t_\ell + C + 1 - V \leq 2$, $L_{\ell,h} + 1 - t_\ell \leq 1$; so $\lambda(x_0) \leq 2$. Since $\lambda^* = 1$, it follows that $\lambda(x_0) \leq 2\lambda^*$.

Chapter 6

Flexible Job Shop Problems

6.1 Introduction

In this chapter we study a generalization of the job shop scheduling problem (see Chapter 5) called the *flexible job shop problem* [62], which models a wide variety of problems encountered in real manufacturing systems [18, 78]. In the flexible job shop problem an operation O_{ij} can be processed by any machine from a given group $M_{ij} \subseteq M$. The processing time of operation O_{ij} on machine $k \in M_{ij}$ is p_{ij}^k. The goal is to choose for each operation O_{ij} an eligible machine and a starting time so that the maximum completion time $C_{\max}$ over all jobs is minimized. $C_{\max}$ is called the *makespan* or the *length* of the schedule.

The flexible job shop problem is more complex than the job shop problem because of the additional need to determine the assignment of operations to machines. Following the three-field $\alpha|\beta|\gamma$ notation suggested by Vaessens [78] and based on the one given by Graham et al. [25], we denote our problem as $m1m|chain, op \leq \mu|C_{\max}$. In the first field m specifies that the number of machines is a constant, 1 specifies that any operation requires at most one machine to be processed, and the second m gives an upper bound on the number of machines that can process an operation. The second field states the precedence constraints and the maximum number of operations per job, while the third field specifies the objective function. The following special cases of the problem are already NP-hard (see [78] for a survey): 2 1 2$|chain, n = 3|C_{\max}$, 3 1 2$|chain, n = 2|C_{\max}$, 2 1 2$|chain, op \leq 2|C_{\max}$.

The flexible job shop problem is equivalent to the problem of scheduling jobs with chain precedence constraints on unrelated parallel machines. For this later problem, Shmoys et al. [75] have designed a polynomial-time

randomized algorithm that, with high probability, finds a schedule of length at most $O((\log^2 n/\log\log n)C^*_{\max})$, where $C^*_{\max}$ is the optimal makespan.

In this work we study the preemptive and non-preemptive versions of the flexible job shop scheduling problem when the number of machines m and the number of operations per job μ are fixed. We generalize the techniques described in [44] for the job shop scheduling problem and design a linear time approximation scheme for the flexible job shop problem. Our algorithm works also for the case when each job J_j has a *delivery time* q_j. If in some schedule, job J_j completes its processing at time C_j, then its *delivery completion time* is equal to $C_j + q_j$. The problem now is to find a schedule that minimizes the maximum delivery completion time $L^*_{\max}$. If in addition each job has a release time r_j when it becomes available for processing, our algorithm finds in linear time a solution of length no more than $(2+\varepsilon)$ times the length of an optimum schedule.

We present a linear time approximation scheme for the preemptive version of the flexible job shop problem without migration. No migration means that each operation must be processed by a unique machine. Hence if an operation is preempted, its processing can only be resumed on the same machine on which it was being processed before the preemption. We also study the preemptive flexible job shop problem with migration and in which every operation has both a release time and a delivery time. We present a $(2+\varepsilon)$-approximation algorithm for it. Both algorithms produce solutions with only a constant number of preemptions.

The job shop problem with multi-purpose machines is a special instance of the flexible job shop problem in which the processing time of an operation is the same regardless of the machine in which it is processed. We present a linear time approximation scheme for this problem that works for the case when each operation has a release time and a delivery time.

6.2 Preliminaries

Consider an instance of the flexible job shop problem with release and delivery times. Let $L^*_{\max}$ be the length of an optimum schedule. For every job J_j, let

$$P_j = \sum_{i=1}^{\mu} [\min_{s \in M_{ij}} p^s_{ij}] \tag{6.1}$$

denote its minimum processing time. Let $P = \sum_{J_j \in \mathcal{J}} P_j$. Let r_j be the release time of job J_j and q_j be its delivery time. We define $t_j = r_j + P_j + q_j$ for all jobs J_j and $t_{\max} = \max_j t_j$.

Lemma 6.1

$$\max\left\{\frac{P}{m},\, t_{\max}\right\} \leq L^*_{\max} \leq P + t_{\max}.$$

Proof. $L^*_{\max} \geq \frac{P}{m}$, since $\frac{P}{m}$ is the length of a schedule with no idle times, in which every operation is processed on the fastest machine, and in which all release and delivery times are 0. Clearly, $L^*_{\max} \geq t_{\max}$. To show that $L^*_{\max} \leq P + t_{\max}$ we describe a simple algorithm that finds a schedule of length $L \leq P + t_{\max}$: take the operations in non-decreasing order of release time, and schedule each operation O_{ij} on the fastest machine m_{ij} that can process it as soon as m_{ij} becomes available after time r_{ij}. Let J_k be the job with largest delivery completion time in this schedule, i.e., $L = s_k + P_k + q_k$, where s_k is the time when job J_k starts its processing. Then, $L = s_k + P_k + q_k \leq r_k + P + P_k + q_k \leq P + t_{\max}$. ■

Let us divide all processing, release, and delivery times by $\max\left\{\frac{P}{m},\, t_{\max}\right\}$, thus by Lemma 6.1,

$$1 \leq L^*_{\max} \leq m + 1, \text{ and } t_{\max} \leq 1. \tag{6.2}$$

We observe that Lemma 6.1 holds also for the preemptive version of the problem with or without migration.

6.3 Non-preemptive Problem

In this section we describe our linear time approximation scheme for the flexible job shop problem with delivery times. We assume that all release times are zero. The algorithm works as follows. First we show how to transform an instance of the flexible job shop problem into another instance without delivery times. Then we define a set of time intervals and assign operations to the intervals in such a way that operations from the same job that are assigned to different intervals appear in the correct order, and the total length of the intervals is no larger than the length of an optimum schedule. We perform this step by first fixing the position of the operations from a constant number of jobs (which we call the long jobs), and then using linear programming to determine the position of the remaining operations.

Next we use an algorithm by Sevastianov [74] to find a feasible schedule for the operations within each interval. Sevastianov's algorithm finds for each interval a schedule of length equal to the length of the interval plus $m\mu^3 p_{\max}$, where $p_{\max}$ is the largest processing time of any operation in the interval. In order to keep this enlargement small we remove from

each interval a subset $\mathcal{V}$ of jobs with the largest operations before running Sevastianov's algorithm. Those operations are scheduled at the beginning of the solution, and by choosing carefully the set of long jobs we can show that the total length of the operations in $\mathcal{V}$ is very small compared to the overall length of the schedule.

Getting Rid of the Delivery Times We can use a technique by Hall and Shmoys [29] to transform an instance of the flexible job shop problem into another instance with only a constant number of different delivery times. Any solution for this later problem can be easily transformed into a solution for the former one, and this transformation increases the length of the solution by only $\varepsilon/2$.

Let $q_{\max}$ be the maximum delivery time and let $\varepsilon > 0$ be a constant value. The idea is to round each delivery time down to the nearest multiple of $\frac{\varepsilon}{2}q_{\max}$ to get at most $1 + 2/\varepsilon$ distinct delivery times. Next, apply a $(1+\varepsilon/2)$-approximation algorithm for the flexible job shop problem that can handle $1 + 2/\varepsilon$ distinct delivery times (this algorithm is described below). Finally, add $\frac{\varepsilon}{2}q_{\max}$ to the completion time of each job; this increases the length of the solution by $\frac{\varepsilon}{2}q_{\max}$. The resulting schedule is feasible for the original instance, so this is a $(1+\varepsilon)$-approximation algorithm for the original problem. In the remainder of this paper, we shall restrict our attention to instances of the problem in which the delivery times $q_1, ..., q_n$ can take only $\chi \leq 1 + \frac{2}{\varepsilon}$ distinct values, which we denote $\delta_1 > ... > \delta_\chi$.

The delivery time of a job can be interpreted as an additional *delivery operation* that must be processed on a *non-bottleneck machine* after the last operation of the job. A non-bottleneck machine is a machine that can process simultaneously any number of operations. Let $\mathcal{D} = \{d_1, ..., d_\chi\}$ be a set of *delivery operations*; each operation d_i has processing time δ_i and it must be processed by a non-bottleneck machine. Moreover, every feasible schedule for the jobs $\mathcal{J}$ can be transformed into another feasible schedule, in which all delivery operations finish at the same time, without increasing the schedule length: simply shift the delivery operations to the end of the schedule. Because of the above interpretation, for every instance of the flexible job shop problem we can get an equivalent instance without delivery times: just add to the set of machines a non-bottleneck machine, and add to every job a delivery operation of length equal to the delivery time of the job. For the rest of this section we assume that every job has a delivery operation, all delivery operations end at the same time, and there is a non-bottleneck machine to process the delivery operations. Let $m + 1$ be the

non-bottleneck machine.

Relative Schedules Assume that the jobs are indexed so that $P_1 \geq P_2 \geq ... \geq P_n$. Let $\mathcal{L} \subset \mathcal{J}$ be the set formed by the first k jobs, i.e., the k jobs with longest minimum processing time, where k is a constant to be defined later. We call $\mathcal{L}$ the set of *long* jobs. An operation from a long job is called a long operation, regardless of its processing time. Let $\mathcal{S} = \mathcal{J} \setminus \mathcal{L}$ be the set of *short* jobs. We create a set $J_{\mathcal{D}}$ of χ *delivery jobs* such that every job $J_j \in J_{\mathcal{D}}$ consists of a single delivery operation d_j.

Consider any feasible schedule for the jobs in $\mathcal{J}$. This schedule assigns a machine to every operation and it also defines a relative ordering for the starting and finishing times of the operations. A *relative schedule* R for $\cup J_{\mathcal{D}}$ is an assignment of machines to long operations and a relative ordering of the starting and finishing times of the long and delivery operations, such that there is a feasible schedule for $\mathcal{J}$ that respects R. This means that for every relative schedule R there is a feasible schedule for $\mathcal{J}$ that assigns the same machines as R to the long operations, that schedules the long operations in the same relative order as R, and in which all delivery operations end at the same time.

Note that since there is a constant number of long and delivery jobs, there is only a constant number of different relative schedules.

Lemma 6.2 *The number of relative schedules for* $\mathcal{L} \cup J_{\mathcal{D}}$ *is at most*

$$m^{k\mu}(2\mu k)!(2\mu k + 1)^{\chi}.$$

Proof. Fix the position of a long operation. The start and finishing times of this operation divide the time into three intervals. Take a second long operation: there are at most 3^2 choices for the starting and finishing intervals of the operation. These two operations define five time intervals, and so for the next long operation there are at most 5^2 possibilities for choosing its starting and ending intervals, and so on. Moreover, for every long operation there are at most m machines that can process it, therefore the number of relative schedules is at most

$$m^{k\mu}(1 \cdot 3^2 \cdot 5^2 \cdots (2\mu k - 1)^2)(2\mu k + 1)^{\chi} \leq m^{k\mu}(2\mu k)!(2\mu k + 1)^{\chi}.$$

■

If we build all relative schedules for $\mathcal{L} \cup J_{\mathcal{D}}$, one of them must be equal to the relative schedule defined by some optimum solution. Since it is possible to build all relative schedules in constant time, we might assume without

loss of generality that we know how to find a relative schedule R such that some optimum schedule for $\mathcal{J}$ respects R.

Fix a relative schedule R as described above. The ordering of the starting and finishing times of the long operations divide the time into intervals that we call *snapshots*. We can view a relative schedule as a sequence of snapshots $M(1), M(2), \ldots, M(g)$, where $M(1)$ is the unbounded snapshot whose right boundary is the starting time of the fist operation according to R, and $M(g)$ is the snapshot with right boundary defined by the finishing time of the delivery operations. Note that the number of snapshots g is

$$g \leq 2\mu k + 1 + \chi \leq (2\mu + 1 + \chi)k \tag{6.3}$$

for each $k \geq 1$.

6.3.1 Scheduling the Small Jobs

Given a relative schedule R as described above, to obtain a solution for the flexible job shop problem we need to schedule the small operations within the snapshots defined by R. We do this in two steps. First we use a linear program $LP(R)$ to assign small operations to snapshots, and second, we find a feasible schedule for the small operations within every snapshot.

To formulate the linear program we need first to define some variables. For each snapshot $M(\ell)$ we use a variable t_ℓ to denote its length. For each $J_j \in \mathcal{S}$ we define a set of decision variables $x_{j,(i_1,\ldots,i_\mu),(s_1,\ldots,s_\mu)}$ with the following meaning: $x_{j,(i_1,\ldots,i_\mu),(s_1,\ldots,s_\mu)} = f$ if and only if for all $q = 1, \ldots, \mu$, an f fraction of the q-th operation of job J_j is completely scheduled in the i_q-th snapshot on machine s_q.

Let α_j be the snapshot where the delivery operation of job J_j starts. For every variable $x_{j,(i_1,\ldots,i_\mu),(s_1,\ldots,s_\mu)}$ we need $1 \leq i_1 \leq i_2 \leq \cdots \leq i_\mu < \alpha_j$ to ensure that the operations of J_j are scheduled in the proper order. Let $A_j = \{(i,s) \mid i = (i_1, \ldots, i_\mu)\ 1 \leq i_1 \leq \ldots \leq i_\mu < \alpha_j,\ s = (s_1, \ldots, s_\mu)\ s_q \in M_{qj}$ and no long operation is scheduled by R at snapshot i_q on machine s_q, for all $q = 1, \ldots, \mu\}$.

The load $L_{\ell,h}$ on machine h in snapshot $M(\ell)$ is the total processing time of the operations from small jobs assigned to h during $M(\ell)$, i.e.,

$$L_{\ell,h} = \sum_{J_j \in \mathcal{S}} \sum_{(i,s) \in A_j} \sum_{\substack{q=1 \\ i_q = \ell, s_q = h}}^{\mu} x_{jis} p_{qj}^{s_q}, \tag{6.4}$$

where i_q and s_q are the q-th components of tuples i and s respectively.

For every long operation O_{ij} let α_{ij} and β_{ij} be the indices of the first and last snapshots where the operation is scheduled. Let p_{ij} be the processing time of long operation O_{ij} according to the machine assignment defined by the relative schedule R. We are ready to describe the linear program $LP(R)$ that assigns small operations to snapshots.

$$
\begin{array}{lll}
\textbf{Minimize} & \sum_{\ell=1}^{g} t_\ell & \\
\textbf{s.t. (1)} & \sum_{\ell=\alpha_{ij}}^{\beta_{ij}} t_\ell = p_{ij}, & \text{for all } J_j \in \mathcal{L},\ i = 1, \ldots, \mu, \\
\textbf{(2)} & \sum_{\ell=\alpha_j}^{g} t_\ell = \delta_j, & \text{for all } d_j,\ j = 1, \ldots, \chi \\
\textbf{(3)} & \sum_{(i,s)\in A_j} x_{jis} = 1\,, & \text{for all } J_j \in \mathcal{S}, \\
\textbf{(4)} & L_{\ell,h} \le t_\ell, & \text{for all } \ell = 1, \ldots, g,\ h = 1, \ldots, m, \\
\textbf{(5)} & t_\ell \ge 0\,, & \text{for all } \ell = 1, \ldots, g, \\
\textbf{(6)} & x_{jis} \ge 0\,, & \text{for all } J_j \in \mathcal{S},\ (i,s) \in A_j.
\end{array}
$$

Lemma 6.3 *An optimum solution of $LP(R)$ has value no larger than the length of an optimum schedule S^* that respects the relative schedule R.*

Proof. To prove the lemma we only need to show how to build a feasible solution for $LP(R)$ that schedules the jobs in exactly the same way as S^*. First we assign to each variable t_ℓ value equal to the length of snapshot $M(\ell)$ in schedule S^*. Then, we assign values to the variablesx_{jis} as follows.

1. For each operation O_{kj}, snapshot $M(\ell)$, $\ell = 1, \ldots, g$, and machine $h = 1, \ldots, m$, initialize $f_{kj}(\ell, h)$ to be the fraction of operation O_{kj} that is scheduled in snapshot $M(\ell)$ and on machine h according to S^*. Initialize the variables x_{jis} to 0.

2. For each job J_j and each $(i,s) \in A_j$:

3. set $x_{jis} = f$, where $f := \min\{f_{kj}(i_k, s_k) \mid k = 1, \ldots, \mu\}$, and

4. set $f_{kj}(i_k, s_k) := f_{kj}(i_k, s_k) - f$ for each $k = 1, \ldots, \mu$.

First, note that for any given feasible solution it is $\sum_{\ell=1}^{g} \sum_{h=1}^{m} f_{kj}(\ell, h) = 1$, for each $J_j \in \mathcal{J}$ and $1 \le k \le \mu$. Each time steps (2), (3) and (4) are completed, the same fraction f of every operation of job J_j is assigned to the same snapshot and machine as in S^*; furthermore for at least one operation O_{kj} the new value of $f_{kj}(i_k, s_k)$ will be set to zero. Hence, $\sum_{(i,s)\in A_j} x_{jis} \le 1$. To show that at the end of the above procedure $\min\{f_{kj}(i_k, s_k) \mid k = 1, \ldots, \mu\} = 0$ for all $(i_1, \ldots, i_\mu), (s_1, \ldots, s_\mu) \in A_j$, suppose the contrary, i.e. that there is a fraction $\hat{f} > 0$ of an operation of job J_j that is not assigned to any snapshot and machine, i.e., $\sum_{(i,s)\in A_j} x_{jis} < 1$. But then, the same

fraction $\hat{f}$ of every operation of job J_j is not assigned by this procedure to any snapshot and to any machine, and therefore, there is at least a pair $(i_1, \ldots, i_\mu), (s_1, \ldots, s_\mu) \in A_j$ for which $\min\{f_{kj}(i_k, s_k) \mid k = 1, \ldots, \mu\} > 0$, which is a contradiction. ■

One can solve $LP(R)$ optimally in polynomial time and get only a constant number of jobs with fractional assignments since a basic feasible solution of $LP(R)$ has at most $k\mu + n - k + mg + \chi$ variables with positive value. By constraint (3) every small job has at least one positive variable associated with it, and so there are at most $mg + k\mu + \chi$ jobs with fractional assignments. We show later how to get rid of any constant number of fractional assignments by only slightly increasing the length of the solution.

The drawback of this approach is that solving the linear program might take a very long time. Since we want to get an approximate solution to the flexible job shop problem, it is not necessary to find an optimum solution for the linear program. An approximate solution for $LP(R)$ would suffice for our purposes. We can use an algorithm by Grigoriadis and Khachiyan [28] to find an approximate solution for $LP(R)$ in linear time.

Approximate Solution of the Linear Program

A *convex block-angular resource sharing problem* has the form:

$$\min \left\{ \lambda \;\middle|\; \sum_{k=1}^{K} f_m^k(x^k) \leq \lambda, \text{for all } m = 1, \ldots, M, \text{and } x^k \in B^k, k = 1, \ldots, K \right\}$$

where $f_m^k : B^k \to \Re^+$ are M non-negative continuous convex functions, and B^k are K disjoint convex compact sets called *blocks*. The Potential Price Directive Decomposition Method of Grigoriadis and Khachiyan [28] can be used to find a $(1+\rho)$-approximate solution to this problem for any value $\rho > 0$. This algorithm needs $O(M(\rho^{-2} \ln \rho^{-1} + \ln M)(M \ln \ln(M/\rho) + KF))$ time, where F is the time needed to find a ρ-approximate solution to the following problem on any block B^k:

$$\min \left\{ \sum_{i=1}^{M} p_i f_i^k(x^k) \;\middle|\; x^k \in B^k \right\} \tag{6.5}$$

for some vector $(p_1, \ldots, p_M) \in \Re^M$.

We can write $LP(R)$ as a convex block-angular resource sharing problem as follows. First we guess the value s of an optimum solution for $LP(R)$,

and add the constraint: $\sum_{\ell=1}^{g} t_\ell \leq s$ to the linear program. Note that $s \leq m+1$. Then we replace constraint (4) by constraint (4'), where λ is a non-negative value:

(4') $\quad L_{\ell,h} - t_\ell + m + 1 \leq \lambda, \quad$ for all $\ell = 1, \ldots, g,\ h = 1, \ldots, m.$

This new linear program, that we denote as $LP(R, s, \lambda)$, has the above block-angular structure. The blocks $B_j = \{< x_{jis} >_{(is)\in A_j} |$ constraints (3) and (6) hold$\}$, are $(mg)^\mu$-dimensional *simplicies*. The block $B_{|\mathcal{S}|+1} = \{t_\ell \mid \sum_{\ell=1}^{g} t_\ell \leq s$ and constraints (1), (2), and (5) hold$\}$ has also constant dimension. Let $f_{\ell,h} = L_{\ell,h} - t_\ell + m + 1$. Since $t_\ell \leq s \leq m+1$, these functions are non-negative.

Each block B_i has constant dimension, and so the block optimization problem (6.5) can be solved in constant time. Therefore, the algorithm of [28] finds a $(1+\rho)$-approximate solution for $LP(R, s, \lambda)$ in $O(n)$ time for any value $\rho > 0$. By choosing $\lambda = m+1$ and $\rho = \rho'/(m+1)$ we get a feasible solution of $LP(R, s, (m+1+\rho'))$. Or in other words we get a feasible assignment of operations of small jobs to snapshots in which the total load on any machine in any snapshot $M(\ell)$ is at most $t_\ell + \rho'$.

Let $L^*_{\max}$ be the length of an optimum schedule and let us assume that R is a relative schedule for $\mathcal{L}$ in an optimum schedule. By Lemma 2, $1 \leq L^*_{\max} \leq m+1$. Thus we can use binary search on the interval $[1, 1+m]$ to find a value $s \leq (1+\frac{\varepsilon}{8})L^*_{\max}$ such that $LP(R, s, (m+1+\rho'))$ has a solution for $\rho' = \frac{\varepsilon}{8g}$. This search can be done in $O(\log(\frac{1}{\varepsilon}\log m))$ iterations by performing the binary search only on the following values,

$$(1+\frac{\varepsilon}{8}), (1+\frac{\varepsilon}{8})^2, ..., (1+\frac{\varepsilon}{8})^{b-1}, m+1 \tag{6.6}$$

where b is the smallest integer such that $(1+\varepsilon/8)^b \geq m+1$. Thus, $b \leq \ln(m+1)/\ln(1+\varepsilon/8)+1 = O(\frac{1}{\varepsilon}\log m)$, since $\ln(1+\varepsilon/8) \geq \frac{\varepsilon/8}{1+\varepsilon/8}$. To see that this search yields the desired value for s, note that there exists a nonnegative integer $i \leq b$ such that $L^*_{\max} \in [(1+\frac{\varepsilon}{8})^i, (1+\frac{\varepsilon}{8})^{i+1}]$ and therefore with the above search we find a value $s \leq (1+\frac{\varepsilon}{8})L^*_{\max}$ for which $LP(R, s, m+1+\rho')$ has a feasible solution. Linear program $LP(R, s, m+1+\rho')$ assumes that the length of each snapshot is increased by ρ', and therefore the total length of the solution is $(1+\frac{\varepsilon}{8})L^*_{\max} + g\rho' \leq (1+\frac{\varepsilon}{4})L^*_{\max}$.

Lemma 6.4 *A solution for $LP(R, s, m+1+\rho')$, with $s \leq (1+\frac{\varepsilon}{8})L^*_{max}$ and $\rho' = \frac{\varepsilon}{8g}$, of value at most $(1+\frac{\varepsilon}{4})L^*_{\max}$ can be found in linear time.*

Rounding Step

In this section we show how we can modify any feasible solution for $LP(R, s, m+1+\rho')$ to get a new feasible solution in which all but a constant number of variables $x_{j,(i_1,\ldots,i_\mu),(s_1,\ldots,s_\mu)}$ have value 0 or 1. Moreover we can do this rounding step in linear time. Let $n' = |\mathcal{S}|$.

Let us write the linear program $LP(R, s, m+1+\rho')$ in matrix form as $Bx = b$, $x \geq 0$. We arrange the columns of B so that the last g columns contain the coefficients for variables t_ℓ and the first $n'(mg)^\mu$ columns have the coefficients for variables $x_{j,(i_1,\ldots,i_\mu),(s_1,\ldots,s_\mu)}$. The key observation that allows us to perform the rounding step in linear time is to note that matrix B is sparse. In particular, each one of the first $n'(mg)^\mu$ columns has at most $\mu+1$ non-zero entries: one entry comes from constraint (3) and at most μ entries come from constraint (4')), and we show below that there exists a constant size subset B' of these columns in which the number of non-zero rows is smaller than the number of columns. The non-zero entries of B' induce a singular matrix of constant size, so we can find a non-zero vector y in the null space of this matrix, i.e., $B'y = 0$.

Let $\delta > 0$ be the smallest value such that some component of the vector $x+\delta y$ is either zero or one (if the dimension of y is smaller than the dimension of x we augment it by adding an appropriate number of zero entries). Note that the vector $x+\delta y$ is a feasible solution of $LP(R, s, m+1+\rho)$. Let $x^0$and x^1 be respectively the zero and one components of vector $x+\delta y$. We update the linear program by making $x = x + \delta y$ and then removing from x all variables in x^0 and x^1 and all columns of B corresponding to such variables. If $x^1 \neq \emptyset$ then vector b is set to $b - \sum_{i \in x^1} B[*, i]$, where $B[*, i]$ is the column of B corresponding to variable i.

This process rounds the value of at least one variable $x_{j,(i_1,\ldots,i_\mu),(s_1,\ldots,s_\mu)}$ to either 0 or 1 and it can be carried out in constant time since the sizes of the submatroids involved are constant. We note that the value of δ can be found in constant time also since y has constant size. We can repeat this process until only a constant number of variables $x_{j,(i_1,\ldots,i_\mu),(s_1,\ldots,s_\mu)}$ have fractional values. Since there is a linear number of these variables then the overall time is linear.

Now we describe the rounding algorithm in more detail. Let us assume that the first $n'(mg)^\mu$ columns of B are indexed so that the columns corresponding to variables $x_{j,(i_1,\ldots,i_\mu),(s_1,\ldots,s_\mu)}$ for each job J_j appear in adjacent positions. We might assume that at all times during the rounding procedure each job J_j has associated at least two columns in B. This assumption can be made since if job J_j has only one associated column, then the cor-

responding variable $x_{j,(i_1,\ldots,i_\mu),(s_1,\ldots,s_\mu)}$ must have value either zero or one. Let B' be the set formed by the first $2mg+1$ columns of B. Note that at most $2mg$ rows of B' have non-zero entries. To see this observe that at most $mg+1$ of these entries come from constraint (3) because of the above assumption on the number of columns for each job, while at most $mg-1$ non-zero entries come from constraint (4′) because only a delivery operation can be scheduled in the last snapshot.

To avoid introducing more notation let B' be the matrix induced by the non-zero rows of the first $2mg+1$ columns of B. Since B' has at most $2mg$ rows and exactly $2mg+1$ columns then B' is singular and hence its null space has at least one non-zero vector y such that $B'y=0$. Since the size of B' is constant, vector y can be found in constant time by using simple linear algebra.

After updating x, B, and b as described above, the procedure is repeated. This is done until there are at most $2mg$ columns in B corresponding to variables $x_{j,(i_1,\ldots,i_\mu),(s_1,\ldots,s_\mu)}$. Hence the total number of iterations is at most $n'(mg)^\mu - 2mg$ and each iteration can be done in constant time.

Lemma 6.5 *A solution for $LP(R,s,m+1+\rho')$ can be transformed in linear time into another feasible solution for $LP(R,s,m+1+\rho')$ in which the set of jobs $\mathcal{F}$ that have fractional assignments in more than one snapshot has size $|\mathcal{F}| \leq mg$.*

Proof. By the above argument at most $2mg$ variables $x_{j,(i_1,\ldots,i_\mu),(s_1,\ldots,s_\mu)}$ might have value different from 0 and 1. Since at the end of the rounding procedure each job has either 0 or at least 2 columns in B, then at most mg jobs receive fractional assignments. ■

6.3.2 Generating a Feasible Schedule

We assume, without loss of generality, that our solution for $LP(R,s,m+1+\rho')$ has value no larger than $m+1$, otherwise the solution obtained by scheduling the jobs sequentially on the machines with the smallest processing time has a better makespan in the worst case. To get a feasible schedule from the solution of the linear program we first need to remove all jobs $\mathcal{F}$ that received fractional assignment. These jobs are placed sequentially at the beginning of the schedule.

For every operation of the remaining small jobs, consider its processing time according to the machine selected for it by the solution of the linear program. Let $\mathcal{V}$ be the set formed by all operations of small jobs with

processing time larger than $\tau = \frac{\varepsilon}{8\mu^3 mg}$. Note that

$$|\mathcal{V}| \leq \frac{m(m+1)}{\tau} = \frac{8\mu^3 m^2(m+1)g}{\varepsilon}. \tag{6.7}$$

We remove from the snapshots all operations in $\mathcal{V}$ and place them sequentially at the beginning of the schedule.

Let $O(\ell)$ be the set of operations from small jobs that remain in snapshot $M(\ell)$. Let $p_{max}(\ell)$ be the maximum processing time among the operations in $O(\ell)$. Observe that the set of jobs in every snapshot $M(\ell)$ defines an instance of the job shop problem, since the linear program assigns a unique machine to every operation of those jobs. Hence we can use Sevastianov's algorithm [74] to find in $O(n^2\mu^2m^2)$ time a feasible schedule for the operations $O(\ell)$; this schedule has length at most $\bar{t}_\ell = t_\ell + \rho' + \mu^3 m p_{max}(\ell)$. We must further increase the length of every snapshot $M(\ell)$ to $\bar{t}_\ell$ to accommodate the schedule produced by Sevastianov's algorithm. Summing up all these enlargements, we get:

Lemma 6.6

$$\sum_{\ell=1}^{g} \mu^3 m p_{max}(\ell) \leq \mu^3 mg\tau = \frac{\varepsilon}{8} \leq \frac{\varepsilon}{8} L^*_{\max}. \tag{6.8}$$

Note that the total length of the snapshots $M(\alpha_{ij}), \ldots, M(\beta_{ij})$ containing a long operation O_{ij} might be larger than p_{ij}. This creates some idle times on machine m_{ij}. We start operations O_{ij} for long jobs $\mathcal{L}$ at the beginning of the enlarged snapshot $M(\alpha_{ij})$. The resulting schedule is clearly feasible. Let $S(J') = \sum_{J_j \in J'} P_j$ be the total processing time of all jobs in some set $J' \subset \mathcal{J}$ when the operations of those jobs are assigned to their fastest machines.

Lemma 6.7 *A feasible schedule for the jobs $\mathcal{J}$ of length at most $(1 + \frac{3}{8}\varepsilon)L^*_{\max} + S(\mathcal{F} \cup \mathcal{V})$ can be found in $O(n^2)$ time.*

Proof. The small jobs $\mathcal{F}$ that received fractional assignments and the jobs in $\mathcal{V}$ are scheduled sequentially at the beginning of the schedule on their fastest machines. By Lemmas 6.4 and 6.6 the claim follows. ■

Now we show that we can choose the number k of long jobs so that $S(\mathcal{F} \cup \mathcal{V}) \leq \frac{\varepsilon}{8} L^*_{\max}$.

Lemma 6.8 *[43] Let $\{d_1, d_2, \ldots, d_n\}$ be positive values and $\sum_{j=1}^{n} d_j \leq m$. Let q be a nonnegative integer, $\alpha > 0$, and $n \geq (q+1)^{\lceil \frac{1}{\alpha} \rceil}$. There exists an integer $k \geq 1$ such that $d_{k+1} + \ldots + d_{k+qk} \leq \alpha m$ and $k \leq (q+1)^{\lceil \frac{1}{\alpha} \rceil}$.*

Let us choose $\alpha = \frac{\varepsilon}{8m}$ and $q = \left(\frac{8\mu^3 m(m+1)}{\varepsilon} + 1\right) m(2\mu + 1 + \chi)$. By Lemma 6.5 and inequalities (6.3) and (6.7) , $|\mathcal{F} \cup \mathcal{V}| \leq mg + \frac{8\mu^3 m^2(m+1)}{\varepsilon} g \leq qk$. By Lemma 6.8 it is possible to choose a value $k \leq (q+1)^{\lceil \frac{1}{\alpha} \rceil}$ so that the total processing time of the jobs in $\mathcal{F} \cup \mathcal{V}$ is at most $\frac{\varepsilon}{8} \leq \frac{\varepsilon}{8} L^*_{\max}$. This value of k can clearly be computed in constant time. We select the set $\mathcal{L}$ of long jobs as the set consisting of the k jobs with largest processing times $P_j = \sum_{i=1}^{\mu} [\min_{s \in M_{kj}} p^s_{ij}]$.

Lemma 6.9

$$S(\mathcal{F} \cup \mathcal{V}) \leq \frac{\varepsilon}{8} L^*_{\max}. \tag{6.9}$$

Theorem 6.1 *For any fixed m and μ, there is a linear-time approximation scheme for the flexible job shop scheduling problem that computes for any value $\varepsilon > 0$, a feasible schedule with maximum delivery completion time of value at most $(1+\varepsilon)L^*_{\max}$ in $O(n)$ time.*

Proof. By Lemmas 6.7 and 6.9, the above algorithm finds in $O(n^2)$ a schedule of length at most $(1 + \frac{1}{2}\varepsilon)L^*_{\max}$. This algorithm can handle $1 + \frac{2}{\varepsilon}$ distinct delivery times. By the discussion at the beginning of Section 3.1 it is possible to modify the algorithm so that it handles arbitrary delivery times and it yields a schedule of length at most $(1+\varepsilon)C^*_{\max}$. For every fixed m and ε, all computations can be carried out in $O(n)$ time, with exception of the algorithm of Sevastianov that runs in $O(n^2)$ time. The latter can be sped up to get linear time by "merging" pairs of small jobs together as described in [44]. ■

We note that our algorithm is also a $(2+\varepsilon)$-approximation algorithm when jobs have release and delivery times. Indeed by adding release times to jobs in the schedule found by the algorithm, the maximum delivery completion time cannot increase by more than $r_{\max}$, where $r_{\max}$ is the maximum release time. Hence the length of the schedule is at most $(2+\varepsilon)$ times the optimal value, since $r_{\max} \leq L^*_{\max}$.

6.4 Preemptive Problem without Migration

In the preemptive flexible job shop problem without migration the processing of an operation may be interrupted and resumed later on the same machine. We describe our algorithm only for the case when all release times are zero. As in the non-preemptive case we divide the set of jobs $\mathcal{J}$ into

long jobs $\mathcal{L}$ and short jobs $\mathcal{S}$, with set $\mathcal{L}$ having a constant number of jobs. One can associate a relative schedule to each preemptive schedule of $\mathcal{L}$ by assigning a machine to each long operation, and by determining a relative order for the starting and finishing times of operations from $\mathcal{L} \cup J_{\mathcal{D}}$. The only difference with the non-preemptive case is that some unfeasible relative schedules for the non-preemptive case are now feasible. Namely, consider two long operations a and b assigned to the same machine according to R. Now, R is feasible even if a starts before b but ends after b.

By looking at every time in the schedule when an operation from $\mathcal{L} \cup J_{\mathcal{D}}$ starts or ends we define a set of time intervals, similar to those defined in the non-preemptive case by the snapshots. For convenience we also call these time intervals snapshots. Since $\mathcal{L} \cup J_{\mathcal{D}}$ has a constant number of operations (and hence there is a constant number of snapshots), we can build all relative schedules for $\mathcal{L} \cup J_{\mathcal{D}}$ in constant time. Thus we might assume without loss of generality that we know how to find a relative schedule R that schedules the long operations within the same snapshots as in an optimum schedule.

An operation of a long job is scheduled in consecutive snapshots $i, i+1, \ldots, i+t$, *but* only a fraction (possibly equal to zero) of the operation might be scheduled in any one of these snapshots. However, and this is crucial for the analysis, in every snapshot there can be at most one operation from any given long job.

Now we define a linear program as in the case of the non-preemptive flexible job shop. For each long operation O_{ij} we define variables $y_{ij\ell}$ for every $\alpha_{ij} \leq \ell \leq \beta_{ij}$, where relative schedule R places operation O_{ij} within the snapshots α_{ij} to β_{ij}. Variable $y_{ij\ell}$ denotes the fraction of operation O_{ij} that is scheduled in snapshot ℓ. Let g be the number of snapshots, t_ℓ be the length of the ℓ-th snapshot, and let O_h denote the set of long operations processed by machine h according to the relative schedule R. Let $L_{\ell,h}$ be defined as in Section 6.3.1 and let $L'_{\ell,h}$ denote the total processing time of operations from long jobs that are executed by machine h during snapshot ℓ, i.e., $L'_{\ell,h} = \sum_{O_{ij} \in O_h | \alpha_{ij} \leq \ell \leq \beta_{ij}} y_{ij\ell} p^h_{ij}$. To avoid that some operation O_{kj} of a short job is scheduled on a machine $s_k \notin M_{kj}$ that cannot process it, we set the corresponding variable x_{jis} to 0 where $i = (i_1, \ldots, i_\mu)$ and $s = (s_1, \ldots, s_\mu)$. Furthermore, we make $x_{jis} = 0$ if the last operation of job J_j ends after the starting time of its delivery operation, i.e., if $i_\mu \geq \alpha_j$ where α_j is the snapshot where the delivery operation starts processing. We let Z_j be the set of variables x_{jis} that are set to zero. The linear program to assign operations to snapshots is the following.

$$
\begin{array}{lll}
\textbf{Minimize} & \sum_{\ell=1}^{g} t_\ell & \\
\textbf{s.t. (1')} & \sum_{\ell=\alpha_{ij}}^{\beta_{ij}} y_{ij\ell} = 1 & \text{for all } J_j \in \mathcal{L},\ i = 1, \ldots, \mu, \\
\textbf{(2)} & \sum_{\ell=\alpha_j}^{g} t_\ell = \delta_j, & \text{for all } d_j,\ j = 1, \ldots, \chi \\
\textbf{(3)} & \sum_{(i,s)\in A_j} x_{jis} = 1, & \text{for all } J_j \in \mathcal{S}, \\
\textbf{(4')} & L_{\ell,h} + L'_{\ell,h} \leq t_\ell, & \text{for all } \ell = 1, \ldots, g,\ h = 1, \ldots, m, \\
\textbf{(5)} & t_\ell \geq 0\,, & \text{for all } \ell = 1, \ldots, g, \\
\textbf{(6)} & x_{jis} \geq 0\,, & \text{for all } J_j \in \mathcal{S},\ (i,s) \in A_j, \\
\textbf{(7)} & x_{jis} = 0, & \text{for all } x_{ijs} \in Z_j \text{ and } J_j \in \mathcal{S}, \\
\textbf{(8)} & y_{ij\ell} \geq 0, & \text{for all } 1 \leq i \leq \mu,\ J_j \in \mathcal{L},\ \alpha_{ij} \leq \ell \leq \beta_{ij}.
\end{array}
$$

Lemma 6.10 *An optimum solution of this linear program has value no larger than the length of an optimum schedule S^* that respects the relative schedule R.*

Proof. The proof is similar to that of Lemma 3. ■

Note that in any solution of this linear program the schedule for the long jobs is always feasible, since there is at most one operation of a given job in any snapshot. We find an approximate solution for the linear program using the algorithm of Section 6.3.1, and then we apply our rounding procedure to this solution. After rounding there are at most mg small jobs that are preempted (see Section 6.3.1). These jobs are placed at the beginning of the schedule, as before.

Let $\tau = \frac{\varepsilon}{8\mu^3 mg}$. As for the non-preemptive case, consider the set $\mathcal{V}$ of small jobs containing at least one operation with processing time larger than τ according to the machine assigned to it by the linear program. The cardinality of this set is bounded by $\frac{m(m+1)}{\tau}$. Removing all jobs in $\mathcal{V}$ so the processing times of the remaining operations of small jobs are not larger than τ. The jobs in $\mathcal{V}$ are placed sequentially at the beginning of the schedule.

Next find a feasible schedule for the jobs in every snapshot using Sevastianov's algorithm, but since the length of the schedule produced by this algorithm depends on the value of the largest processing time, it might be necessary to split operations from long jobs into small pieces. So we split each long operation O_{ij} into pieces of size at most τ and then apply Sevastianov's algorithm. This yields a feasible schedule for the snapshot because there is at most one operation of each long job in it.

In this schedule the number of preemptions is a constant equal to the number of times that long operations are preempted, and this number that

is at most

$$\begin{aligned}\sum_{J_j\in\mathcal{L}}\sum_{i=1}^{\mu}\sum_{\ell=\alpha_{ij}}^{\beta_{ij}}\left\lceil\frac{y_{ij\ell}p_{ij}^h}{\tau}\right\rceil &\leq \sum_{J_j\in\mathcal{L}}\sum_{i=1}^{\mu}\sum_{\ell=\alpha_{ij}}^{\beta_{ij}}(\frac{y_{ij\ell}p_{ij}^h}{\tau}+1)\\ &\leq \frac{m(m+1)}{\tau}+kg,\end{aligned}$$

since in each snapshot ℓ there are at most k different operations from long jobs, and the sum of the processing times $y_{ij\ell}p_{ij}^h$ cannot be more than $m(m+1)$ (see Lemma 6.1). Hence, our solution has $O(1)$ preemptions. Choosing the size of $\mathcal{L}$ as we did for the non-preemptive case we ensure that the length of the schedule is at most $1+\varepsilon$ times the value of an optimum schedule. This algorithm runs in linear time.

Theorem 6.2 *For any fixed m and μ, there is a linear-time approximation scheme for the preemptive flexible job shop scheduling problem without migration that computes for any fixed $\varepsilon > 0$, a feasible schedule with maximum delivery completion time $(1+\varepsilon)\cdot L^*_{\max}$.*

Note that the algorithm described is also a $(2+\varepsilon)$-approximation algorithm for the flexible job shop problem with release and delivery times.

6.5 Preemptive Problem with Migration

In the preemptive flexible job shop problem with migration the processing of an operation may be interrupted and resumed later on any eligible machine. We consider the problem when each job J_j has a release time r_j and a delivery time q_j.

In this section we describe a linear time $(2+\varepsilon)$ -approximation algorithm for the problem,when the number of machines and operations per job are fixed.

Finding a Schedule When Release Times are Zero In the first step we use linear programming to assign operations to machines and to snapshots assuming that all release times are zero. We wish to make this assignment in such a way that the total length of the solution is minimized. We also assume that we have reduced the number of different delivery times to a constant χ as in the non-preemptive case. Let t denote the value of the objective function of the linear program (defined below). Define χ snapshots as follows: $[0, t-\delta_1], [t-\delta_1, t-\delta_2], ..., [t-\delta_{\chi-1}, t-\delta_\chi]$ where $\delta_1 > \delta_2 > ... > \delta_\chi$

are the delivery times. For each job J_j we use a set of decision variables $x_{jis} \in [0,1]$ for tuples $(i,s) \in A_j$.

The meaning of these variables is that $x_{jis} = 1$ if and only if, for each $1 \leq k \leq \mu$, operation O_{kj} is scheduled on machine s_k, and in snapshot $[0, t-\delta_1]$ if $i_k = 1$,or in snapshot $[t-\delta_{i_k-1}, t-\delta_{i_k}]$ if $1 < i_k \leq \chi$. Let the load $L_{\ell,h}$ on machine h in time interval ℓ be defined as the total processing time of all operations that are executed by machine h during time interval ℓ.

To avoid that some operation O_{kj} is scheduled on a machine $s_k \notin M_{kj}$ that cannot process it, we set the corresponding variable x_{jis} to 0. Furthermore, if the last operation of job J_j ends after the starting time of its delivery operation we make the corresponding variable $x_{jis} = 0$. We let Z_j be the set of variables x_{ijs} that are set to zero. The linear program LP is as follows.

Minimize t

(1)	$\sum_{(i,s)\in A_j} x_{jis} = 1\,,$	for all $J_j \in \mathcal{J}$,
(2)	$L_{1,h} \leq t - \delta_1,$	for all $h = 1, \ldots, m$,
(3)	$L_{\ell,h} \leq \delta_{\ell-1} - \delta_\ell,$	for all $h = 1, \ldots, m, \ell = 2, \ldots, \chi$,
(4)	$x_{jis} \geq 0\,,$	for all $J_j \in \mathcal{J}$, $(i,s) \in A_j$,
(5)	$x_{jis} = 0,$	for all $x_{ijs} \in Z_j$ and $J_j \in \mathcal{J}$.

Lemma 6.11 *An optimum solution of LP has value no larger than the maximum delivery time of an optimum schedule for* $\mathcal{J}$.

Proof. Consider an optimum schedule S^* for $\mathcal{J}$. We only need to show that for any job $J_j \in \mathcal{J}$ there is a feasible solution of the linear program that schedules all operations of J_j in the same positions as S^*. The proof is similar to that of Lemma 3. ■

We guess the value s of an optimum schedule. Let $LP(s,\lambda)$ be the linear program with objective function: **Minimize** λ, and constraints (1), (4), and (5) from the above linear program plus the following two

(2')	$\frac{L_{1,h}}{s-\delta_1} \leq \lambda,$	for all $1 \leq h \leq m$,
(3')	$\frac{L_{\ell,h}}{\delta_{\ell-1}-\delta_\ell} \leq \lambda,$	for all $1 \leq h \leq m$, $\ell = 2, 3, ..., \chi$.

The linear program $LP(s,\lambda)$ has the structure required by the algorithm of Grigoriadis and Khachiyan [28]. The blocks $B_j = \{< x_{jis} >_{(is)\in A_j} |$ conditions (1), (4), and (5) are satisfied $\}$ for every $J_j \in \mathcal{J}$ are $(m\chi)^\mu$-dimensional simplicies. Linear inequalities $(2')$ and $(3')$ form a set of so

called *coupling constraints*. Note that functions $\frac{L_{1,h}}{s-\delta_1}$ and $\frac{L_{\ell,h}}{\delta_{\ell-1}-\delta_\ell}$ are non-negative.

Using the algorithm [28] and binary search on the interval $[1, 1+m]$, we can find in linear time a value $s \leq (1+\frac{\varepsilon}{8})t^*$, where t^* is the value of an optimum solution of LP, such that $LP(s, 1+\rho)$ is feasible for $\rho = 1+\frac{\varepsilon}{8+\varepsilon}$. For this value s a solution of $LP(s, 1+\rho)$ has $L_{1,h} \leq (s-\delta_1)(1+\rho)$ and $L_{\ell,h} \leq (\delta_{\ell-1}-\delta_\ell)(1+\rho)$. Since $\rho = 1+\frac{\varepsilon}{8+\varepsilon}$ the total enlargement of all snapshots is at most $(s-\delta_1)\rho + \sum_{\ell=2}^{\chi}(\delta_{\ell-1}-\delta_\ell)\rho = (s-\delta_\chi)\rho \leq s\rho \leq \rho(1+\frac{\varepsilon}{8})t^* = (1+\frac{\varepsilon}{4})t^*$.

Lemma 6.12 *A solution for $LP(s, 1+\rho)$, with $s \leq (1+\frac{\varepsilon}{8})t^*$ and $\rho = 1+\frac{\varepsilon}{8+\varepsilon}$, of value at most $(1+\frac{\varepsilon}{4})t^*$ can be found in linear time.*

Using a procedure similar to that of Section 6.3.1 it is possible to modify any feasible solution of $LP(s, 1+\rho)$ to get a new feasible solution with at most $m\chi$ variables $x_{jis}$with fractional values. Moreover we can do this rounding step in linear time.

Lemma 6.13 *Any feasible solution of $LP(s, 1+\rho)$ can be transformed in linear time into another feasible solution in which the set of jobs $\mathcal{F}$ that receive fractional assignments has size $|\mathcal{F}| \leq m\chi$.*

Let $\mathcal{P}$ denote the set of jobs from $\mathcal{J}\backslash\mathcal{F}$ for which at least one operation, according to the machine assignment computed in step 1, has processing time greater than $\frac{\varepsilon s}{4\chi\mu^3 m(1+\frac{\varepsilon}{8})}$. Let $\mathcal{L} = \mathcal{F} \cup \mathcal{P}$ and $\mathcal{S} = \mathcal{J}\backslash\mathcal{L}$. We remove from the snapshots the jobs in $\mathcal{L}$ and then use Sevastianov's algorithm to find a feasible schedule $\sigma_{\mathcal{S}}$ for $\mathcal{S}$. The enlargement in the length of the schedule caused by Sevastianov's algorithm is at most $\chi\mu^3 m p_{\max}$, where $p_{\max}$ is the maximum processing time of operations in $\mathcal{S}$.

Since $p_{\max} \leq \frac{\varepsilon s}{4\chi\mu^3 m(1+\frac{\varepsilon}{8})}$, the length of the resulting schedule is $(1+\frac{\varepsilon}{4})t^* + \chi\mu^3 m p_{\max} \leq (1+\frac{\varepsilon}{4})t^* + \frac{\varepsilon s}{4(1+\frac{\varepsilon}{8})} \leq (1+\frac{\varepsilon}{2})t^* \leq (1+\frac{\varepsilon}{2})L^*_{\max}$.

Note that by considering release times different from zero this algorithm yields a schedule for $\mathcal{S}$ that is at most $(2+\frac{\varepsilon}{2})$ times the length of an optimal one, since the maximum release time cannot be more than $L^*_{\max}$. The algorithm of Sevastianov takes $O(n^2)$ time, but it can be sped up to get linear time by "merging" pairs of jobs as described in [44].

The cardinality of set $\mathcal{L}$ is bounded by a constant since $|\mathcal{P}| < m(1+\frac{\varepsilon}{4})t^* \frac{4\chi\mu^3 m(1+\frac{\varepsilon}{8})}{\varepsilon s} \leq \frac{4\chi\mu^3 m^2(1+\frac{\varepsilon}{8})(1+\frac{\varepsilon}{4})}{\varepsilon} = O(\frac{\mu^3 m^2}{\varepsilon^2})$, and so $|\mathcal{L}| = |\mathcal{F}|\cup|\mathcal{P}| = O(\frac{\mu^3 m^2}{\varepsilon^2})$.

Scheduling the Jobs in $\mathcal{L}$ Now we ignore the delivery times and consider only the release times. In the following we show how to compute a schedule $\sigma_{\mathcal{L}}$ that minimizes the makespan for the jobs from $\mathcal{L}$.

As for the delivery times, the release time of a job can be interpreted as an additional operation of the job that it has to process on a non-bottleneck machine before any of its other operations. Because of this interpretation, we can add to the set $\mathcal{O}_{\mathcal{L}}$ of operations from jobs $\mathcal{L}$, a set $\mathcal{R} = \{O_{01}, O_{02}, \ldots, O_{0\chi}\}$ of release operations. The processing time of operation O_{0j} is r_j.

Let us define a snapshot as a subset of operations from $\mathcal{O}_{\mathcal{L}} \cup \mathcal{R}$ such that two different operations of the same job do not belong to the same snapshot. A *relative order* of $\mathcal{L}$ is an ordered sequence of snapshots which defines for every operation $O_{ij} \in \mathcal{O}_{\mathcal{L}} \cup \mathcal{R}$ the first α_{ij} and the last β_{ij} snapshots in which operation O_{ij} can start and finish, respectively. Let g be the number of snapshots. A relative order is feasible if $1 \leq \alpha_{ij} \leq \beta_{ij} \leq g$ for every operation O_{ij}, and for two operations O_{ij} and O_{i+1j} of the same job $J_j \in \mathcal{L}$, we have $\beta_{ij} + 1 = \alpha_{i+1j}$. Furthermore, every release operation starts in the first snapshot, i.e., $\alpha_{0j} = 1$ for every $J_j \in \mathcal{L}$.

We observe that g can be bounded by $(\mu + 1)|\mathcal{L}|$, and therefore $g = O(\frac{\mu^4 m^2}{\varepsilon^2})$. Note that a relative order is defined without assigning operations of long jobs to machines.

Since $\mathcal{L}$ has a constant number of jobs (and hence there is a constant number of snapshots), we can consider all relative orders for $\mathcal{L}$. Any operation is scheduled in consecutive snapshots $i, i+1, \ldots, i+t$, *but* only a fraction (possibly equal to zero) of that operation might be scheduled in any of these snapshots. In every snapshot there can be at most one operation from any given job of $\mathcal{L}$.

Now we define a linear program. For each operation O_{ij} we define variables $x_{ij\ell h}$ for every $\alpha_{ij} \leq \ell \leq \beta_{ij}$ and every $h \in M_{ij}$. Variable $x_{ij\ell h}$ denotes the fraction of operation O_{ij} that is scheduled in snapshot ℓ on machine h. Let t_ℓ be the length of snapshot ℓ. The load on machine h in snapshot ℓ is equal to the total processing time of the operations of jobs in $\mathcal{L}$, i.e., $L_{\ell,h} = \sum_{J_j \in \mathcal{L}} \sum_{i=1,\ldots,\mu | \alpha_{ij} \leq \ell \leq \beta_{ij}, h \in M_{ij}} x_{ij\ell h} p_{ij}^h$. The linear program for a given relative order R is the following.

Minimize $\sum_{\ell=1}^{g} t_\ell$

s.t. **(1)** $t_\ell \geq 0, \qquad 1 \leq \ell \leq g$

(2) $\sum_{h \in M_{ij}} \sum_{\ell=\alpha_{ij}}^{\beta_{ij}} x_{ij\ell h} = 1, \qquad 1 \leq i \leq \mu,\ J_j \in \mathcal{L},$

$$\textbf{(3)} \quad L_{\ell,h} \leq t_\ell, \qquad 1 \leq \ell \leq g,\ 1 \leq h \leq m,$$

$$\textbf{(4)} \quad \sum_{h \in M_{ij}} x_{ij\ell h} p_{ij}^h \leq t_\ell, \qquad 1 \leq \ell \leq g,\ J_j \in \mathcal{L},\ \alpha_{ij} \leq \ell \leq \beta_{ij},$$

$$\textbf{(5)} \quad x_{ij\ell h} \geq 0, \qquad 1 \leq i \leq \mu,\ J_j \in \mathcal{L},\ \alpha_{ij} \leq \ell \leq \beta_{ij},$$

$$\textbf{(6)} \quad \sum_{\ell=1}^{\beta_{0j}} t_\ell = r_j, \qquad J_j \in \mathcal{L}.$$

Consider any feasible schedule of $\mathcal{L}$ according to a relative order R, and let $x_{ij\ell h} p_{ij}^h$ be the total amount of time that machine h works on operation O_{ij} in snapshot ℓ, and let t_ℓ denote the length of snapshot ℓ. It is clear that this assignment of values to t_ℓ and $x_{ij\ell h}$ constitutes a feasible solution of the above linear program. We show now that for any feasible solution of the linear program, there is a feasible solution with the same values of t_ℓ and $x_{ij\ell h}$. It is sufficient to note that any feasible solution of the linear program defines in each snapshot an instance of the preemptive open shop scheduling problem. Hence, from results in [24, 51] the claim follows. Furthermore, by using the same procedure described in [51] it is possible to construct a schedule with no more than $O(\frac{\mu^4 m^4}{\varepsilon^2})$ preemptions.

For each relative order R we solve the linear program above, and then select the solution with the smallest length. All the algorithms used in this step require polynomial time in the size of the input, therefore the overall time is $O(1)$ since the input size is $O(1)$. Note that if we consider non-zero delivery times then the maximum delivery completion time of the resulting schedule $\sigma_{\mathcal{L}}$ is at most $2L^*_{\max}$.

Combining the Schedules The output schedule is obtained by appending $\sigma_{\mathcal{S}}$ after $\sigma_{\mathcal{L}}$. The length of the solution is at most $(2 + \frac{\varepsilon}{2})L^*_{\max}$ since the maximum completion time of schedule $\sigma_{\mathcal{L}}$ is not greater than $L^*_{\max}$ and the length of the schedule $\sigma_{\mathcal{S}}$ is at most $(1 + \frac{\varepsilon}{2})L^*_{\max}$.

Theorem 6.3 *For any fixed m and μ, there is a polynomial-time approximation algorithm for the preemptive flexible job shop scheduling problem with migration that computes for any fixed $\varepsilon > 0$, a feasible schedule with maximum delivery completion time $(2 + \varepsilon) \cdot L^*_{\max}$ in $O(n)$ time.*

6.6 Multi-Purpose Machines Job Shop

The job shop scheduling problem with multi-purpose machines [9] is a special case of the flexible job shop problem, in which the processing time of

each operation p_{ij} does not depend anymore on the machine on which it is processed. For the non-preemptive version of this problem, our techniques can also handle the case in which each job J_j has a *release time* r_j and a delivery time q_j. The main difference with the previous algorithm is that we put each job J_j from the set $\mathcal{F} \cup \mathcal{V}$ (this set is defined as before) after their corresponding release operations and before their delivery operations. More precisely, the algorithm works as follows.

We use the technique by Hall and Shmoys [29] to obtain a problem instance with only a constant number of delivery and release times. We divide the set of jobs $\mathcal{J}$ into long jobs $\mathcal{L}$ and short jobs $\mathcal{S}$ as before. Then, we compute all the relative schedules of the long jobs; a relative schedule is defined as before, but we add also a constant number of release operations. For each relative schedule we define a linear program as in the non-preemptive case. We solve and round the solution of the linear program to get a solution with a constant number of fractional variables.

The solution of the linear program has a constant number of jobs with fractional assignments (set $\mathcal{F}$), and a constant number of jobs (set $\mathcal{V}$) with at least one operation with large processing time. We now use a very simple rounding procedure to obtain an integral (and possibly infeasible) solution for the linear program. If job J_j has more than one nonzero variable associated with it we set one of them to be one and others to be zero in an arbitrary manner.

The final schedule is built as follows. Assign jobs from $\mathcal{F} \cup \mathcal{V}$ to machines and snapshots according to the new integral solution. Build a left justified (that means without leaving idle times) feasible schedule for the jobs from $\mathcal{F} \cup \mathcal{V}$. Assign the remaining small jobs according to the linear programming solution, and for each snapshot $M(\ell)$, $\ell \in \{1, \ldots, g\}$, schedule operations assigned to $M(\ell)$ after the maximum finishing time in $M(\ell)$ of operations of jobs from $\mathcal{F} \cup \mathcal{V}$ (if any), by using Sevastianov's algorithm. Again, from Lemma 6.8 it is possible to choose the number of long jobs such that the enlargement due to the jobs from $\mathcal{F} \cup \mathcal{V}$ is "small" enough, and by using similar arguments as before, it is possible to show that the described algorithm is a polynomial time approximation scheme which runs in linear time.

Theorem 6.4 *For any fixed m and μ, there is a polynomial-time approximation scheme for the MPM job shop scheduling problem with release and delivery times that computes for any fixed real number ε, with $\varepsilon > 0$, a feasible schedule with maximum delivery completion time of at most $(1+\varepsilon) \cdot L^*_{\max}$ in $O(n)$ time.*

By using similar arguments as before, it is possible to provide a linear time approximation scheme also when preemption without migration is allowed.

Theorem 6.5 *For any fixed m and μ, there is a polynomial-time approximation scheme for the preemptive MPM job shop scheduling problem without migration with release and delivery times that computes for any fixed real number ε, with $\varepsilon > 0$, a feasible schedule with maximum delivery completion time of at most $(1+\varepsilon) \cdot L^*_{\max}$ in $O(n)$ time.*

Chapter 7

Cardinality Constrained Knapsack Problems

7.1 Introduction

In the classical *Knapsack Problem* (KP) we have a set $N := \{1, \ldots, n\}$ of items and a knapsack of limited capacity. To each item we associate a positive profit p_j and a positive weight w_j. The problem calls for selecting the set of items with maximum overall profit among those whose total weight does not exceed the knapsack capacity $c > 0$. KP has the following Integer Linear Programming (ILP) formulation:

$$\text{maximize} \quad \sum_{j \in N} p_j x_j \tag{7.1}$$

$$\text{subject to} \quad \sum_{j \in N} w_j x_j \leq c \tag{7.2}$$

$$x_j \in \{0, 1\}, \quad j \in N, \tag{7.3}$$

where each binary variable x_j, $j \in N$, is equal to 1 if and only if item j is selected. In general, we cannot take all items because the total weight of the chosen items cannot exceed the knapsack capacity c. In the sequel, without loss of generality, we assume that $\sum_{j \in N} w_j > c$ and $w_j \leq c$ for every $j \in N$.

The *k-item Knapsack Problem* (kKP), is a KP in which an upper bound of k is imposed on the number of items that can be selected in a solution. The problem can be formulated as (7.1)-(7.3) with the additional constraint

$$\sum_{j \in N} x_j \leq k, \tag{7.4}$$

with $1 \leq k \leq n$.

KP has widely been studied in the literature, see the book of Martello and Toth [57] for a comprehensive illustration of the problem. kKP is the subproblem to be solved when instances of the Cutting Stock Problem with cardinality constraints are tackled by column generation techniques. kKP also appears in processor scheduling problems on computers with k processors and shared memory. Furthermore, kKP could replace KP in the separation of cover inequalities, as outlined in [10].

Throughout this chapter let OPT denote the optimal solution value to the given instance and $w(F) = \sum_{j \in F} w_j$ and $p(F) = \sum_{j \in F} p_j$, where $F \subseteq N$. An algorithm A with solution value z^A is called an *ε-approximation algorithm*, $\varepsilon \in (0, 1)$, if $z^A \geq (1 - \varepsilon)OPT$ holds for all problem instances. We will also call ε the *performance ratio* of A.

Known Results It is well known that KP is NP-hard but pseudopolynomially solvable through dynamic programming, and the same properties hold for kKP [10]. Basically, the developed approximation approaches for KP and kKP can be divided into three groups:

1. *Approximation algorithms.* For KP the classical $\frac{1}{2}$-approximation algorithm (see e.g. [50]) needs only $O(n)$ running time. A performance ratio of $\frac{1}{2}$ can be obtained also for kKP by rounding the solution of the linear programming relaxation of the problem (see [10]); this algorithm can be implemented to run in linear time when the LP relaxation of kKP is solved by using the method by Megiddo and Tamir [64].

2. *Polynomial time approximation schemes* (PTAS) reach any given performance ratio and have a running time polynomial in the length of the encoded input. The best schemes currently known requiring linear space are given in Caprara et al. [10]: they yield a performance ratio of ε within $O(n^{\lceil 1/\varepsilon \rceil - 2})$ and $O(n^{\lceil 1/\varepsilon \rceil - 1})$ running time, for KP and kKP respectively.

3. *Fully polynomial time approximation schemes* (FPTAS) also reach any given performance ratio and have a running time polynomial in the length of the encoded input and in the reciprocal of the performance ratio. This improvement compared to 1. and 2. is usually paid off by

larger space requirements, which increases rapidly with the accuracy ε. The first FPTAS for KP was proposed by Ibarra and Kim [37], later on improved by Lawler [50] and Kellerer and Pferschy [48]. In Caprara et al. [10] it is shown that kKP admits an FPTAS that runs in $O(nk^2/\varepsilon)$ time.

New Results Rounding the input is a widely used technique to obtain polynomial time approximation schemes [32]. Among the developed rounding techniques, arithmetic or geometric rounding are the most successfully and broadly used ways of rounding to obtain a simpler instance that may be solved in polynomial time (see Sections 7.2.1 and 7.2.2 for an application of these techniques to kKP). We contribute by presenting a new technical idea. We show that appropriate combinations of arithmetic and geometric rounding techniques yield novel and powerful rounding methods. To the best of our knowledge, these techniques have never been combined together. By using the described rounding techniques, we present a PTAS for kKP requiring linear space and running time $O(n+k\cdot(1/\varepsilon)^{O(1/\varepsilon)})$. Our algorithm is clearly superior to the one in [10], and it is worth noting that the running time contains no exponent on n dependent on ε. Since KP is a special case of kKP, we also improve the previous result for KP. Finally we present a faster FPTAS for kKP that runs in $O(n+k/\varepsilon^4+1/\varepsilon^5)$ time and has a bound of $O(n+1/\varepsilon^4)$ on space requirements.

7.2 Rounding Techniques for kKP

The aim of this section is to transform any input into one with a smaller size and a simpler structure without dramatically decreasing the objective value. We discuss several transformations of the input problem. Some transformations may potentially decrease the objective function value by a factor of $1-O(\varepsilon)$, so we can perform a constant number of them while still staying within $1-O(\varepsilon)$ of the original optimum. Others are transformations which do not increase the objective function value. When we describe the first type of transformation, we shall say it produces $1-O(\varepsilon)$ *loss*, while the second produces *no loss*.

Let P^H denote the solution value obtained in $O(n)$ time by employing the 1/2-approximation algorithm $H^{\frac{1}{2}}$ for kKP described in [10]. In [10], it is shown that

$$2P^H \geq P^H + p_{\max} \geq OPT \geq P^H, \tag{7.5}$$

where $p_{\max} = \max_j p_j$.

Throughout this section we restrict our attention to feasible solutions with at most γ items, where γ is a positive integer not greater than k. The first observation is that at most an ε-fraction of the optimal profit OPT is lost by discarding all items j where $p_j \leq \varepsilon P^H/\gamma$, since at most γ items can be selected and $P^H \leq OPT$. From now on, consider the reduced set of items with profit values greater than $\varepsilon P^H/\gamma$.

In order to reduce further the set of items, a useful insight is that when profits are identical we pick items in non-decreasing order of weight. Since the optimal profit is at most $2P^H$, among all items with profit value $\bar{p} \in \{p_1, ..., p_n\}$, we can keep the first $\bar{n} = \min\left\{\gamma, \left\lfloor \frac{2P^H}{\bar{p}} \right\rfloor\right\}$ items with the smallest weights, and discard the others with no loss. Of course, we cannot hope to obtain a smaller instance if all profits are different. In the following, we show how the number of different profits can be reduced by rounding the original profits. We revise two rounding techniques and show that an appropriate combination of both yields to a better result. We call a profit value $\bar{p}$ *large* if $\bar{p} > \frac{2P^H}{\gamma}$, and *small* otherwise.

7.2.1 Arithmetic Rounding

A sequence $a_1, a_2, ...$ is called an *arithmetic sequence* if, and only if, there is a constant d such that $a_i = a_1 + d \cdot (i-1)$, for all integers $i \geq 1$. Let us consider the arithmetic sequence $S_a(\gamma)$ obtained by setting $a_1 = d = \varepsilon P^H/\gamma$. We transform the given instance into a more structured one by rounding each profit down to the nearest value in $S_a(\gamma)$. Since in the rounded instance the profit of each item is decreased by at most $\varepsilon P^H/\gamma$, and at most γ items can be selected, the solution value of the transformed instance potentially decreases by εP^H. Of course, by restoring the original profits we cannot decrease the objective function value, and therefore, with $1-\varepsilon$ loss, we can assume that every possible profit is equal to $a_i = \frac{\varepsilon P^H}{\gamma} \cdot i$ for some $i \geq 1$. Furthermore, since $p_{\max} = \max_{j \in N} p_j \leq P^H$, the number of different profits is now bounded by $\left\lfloor \frac{\gamma p_{\max}}{\varepsilon P^H} \right\rfloor \leq \left\lfloor \frac{\gamma}{\varepsilon} \right\rfloor$.

The largest number n_i of items with profit a_i, for $i = 1, ..., \left\lfloor \frac{\gamma}{\varepsilon} \right\rfloor$, that can be involved in any feasible solution is bounded by

$$n_i \leq \min\{\gamma, \left\lfloor \frac{OPT}{\frac{\varepsilon P^H}{\gamma} i} \right\rfloor\} \leq \min\{\gamma, \left\lfloor \frac{2\gamma}{\varepsilon i} \right\rfloor\},$$

and we can keep the first n_i items with the smallest weights, and discard the others with no loss. Now, the number of items with profit a_i is at most γ, if a_i is a small profit (i.e. when $i = 1, ..., \left\lfloor \frac{2}{\varepsilon} \right\rfloor$), and at most $\left\lfloor \frac{2\gamma}{\varepsilon i} \right\rfloor$ otherwise

$(i = \lfloor \frac{2}{\varepsilon} \rfloor + 1, ..., \lfloor \frac{\gamma}{\varepsilon} \rfloor)$. Thus, by applying the described arithmetic rounding we have at most $\lfloor 2/\varepsilon \rfloor \gamma$ items with small profits and $\sum_{i=\lfloor \frac{2}{\varepsilon} \rfloor + 1}^{\lfloor \frac{\gamma}{\varepsilon} \rfloor} \left\lfloor \frac{2\gamma}{\varepsilon i} \right\rfloor$ with large profits. Recall that when a summation can be expressed as $\sum_{k=x}^{y} f(k)$, where $f(k)$ is a monotonically decreasing function, we can approximate it by an integral (see, e.g. [14] p. 50):

$$\int_{x}^{y+1} f(k)dk \leq \sum_{k=x}^{y} f(k) \leq \int_{x-1}^{y} f(k)dk.$$

Furthermore, we are assuming that $0 < \varepsilon < 1$, and recall that $\ln(1+x) \geq x/(1+x)$, for $x > -1$. Therefore, the total number of items in the transformed instance is bounded by

$$\begin{aligned}
\left\lfloor \frac{2}{\varepsilon} \right\rfloor \gamma + \sum_{i=\lfloor \frac{2}{\varepsilon} \rfloor + 1}^{\lfloor \frac{\gamma}{\varepsilon} \rfloor} \left\lfloor \frac{2\gamma}{\varepsilon i} \right\rfloor &\leq \frac{2}{\varepsilon}\gamma + \frac{2}{\varepsilon}\gamma \sum_{i=\lfloor \frac{2}{\varepsilon} \rfloor + 1}^{\lfloor \frac{\gamma}{\varepsilon} \rfloor} \frac{1}{i} \\
&\leq \frac{2\gamma}{\varepsilon}(1 + \int_{\frac{2}{\varepsilon}-1}^{\frac{\gamma}{\varepsilon}} \frac{di}{i}) = \frac{2\gamma}{\varepsilon}(1 + \ln\gamma - \ln(2-\varepsilon)) \\
&\leq \frac{2\gamma}{\varepsilon}(1 + \ln\gamma) = O(\frac{\gamma}{\varepsilon}\ln\gamma).
\end{aligned}$$

We see that by applying the described arithmetic rounding we have at most $2\gamma/\varepsilon$ items with small profits and $\frac{2\gamma}{\varepsilon}\ln\gamma$ with large profits.

A natural question is to see if the provided bound is tight. Consider an instance having γ items for each distinct profit $\frac{\varepsilon P^H}{\gamma} \cdot i$, where $i = 1, ..., \lfloor \frac{\gamma}{\varepsilon} \rfloor$. Observe that by applying the described arithmetic rounding, we have exactly $\lfloor 2/\varepsilon \rfloor \gamma$ items with small profits and $\sum_{i=\lfloor \frac{2}{\varepsilon} \rfloor + 1}^{\lfloor \frac{\gamma}{\varepsilon} \rfloor} \left\lfloor \frac{2\gamma}{\varepsilon i} \right\rfloor$ with large profits. What remains to be shown is to bound the number of items to be $\Omega(\frac{\gamma}{\varepsilon}\ln\gamma)$:

$$\begin{aligned}
\left\lfloor \frac{2}{\varepsilon} \right\rfloor \gamma + \sum_{i=\lfloor \frac{2}{\varepsilon} \rfloor + 1}^{\lfloor \frac{\gamma}{\varepsilon} \rfloor} \left\lfloor \frac{2\gamma}{\varepsilon i} \right\rfloor &\geq \frac{2}{\varepsilon}\gamma - 1 + \sum_{i=\frac{2}{\varepsilon}+1}^{\frac{\gamma}{\varepsilon}-1} (\frac{2\gamma}{\varepsilon i} - 1) \\
&\geq \frac{2}{\varepsilon}\gamma - 1 + \frac{2}{\varepsilon}\gamma \sum_{i=\frac{2}{\varepsilon}+1}^{\frac{\gamma}{\varepsilon}-1} \frac{1}{i} - (\frac{\gamma}{\varepsilon} - 2) \\
&\geq \frac{\gamma}{\varepsilon}(1 + 2\int_{\frac{2}{\varepsilon}+1}^{\frac{\gamma}{\varepsilon}} \frac{di}{i}) = \frac{\gamma}{\varepsilon}(1 + 2\ln\gamma - 2\ln(2+\varepsilon)) \\
&= \Omega(\frac{\gamma}{\varepsilon}\ln\gamma).
\end{aligned}$$

7.2.2 Geometric Rounding

A sequence $a_1, a_2, ...$ is called a *geometric sequence* if, and only if, there is a constant r such that $a_i = a_1 \cdot r^{i-1}$, for all integers $i \geq 1$. Let us consider the geometric sequence $S_g(\gamma)$ obtained by setting $a_1 = \varepsilon P^H/\gamma$ and $r = \frac{1}{1-\varepsilon}$. We round each profit down to the nearest value among those of $S_g(\gamma)$. Since $a_i = (1-\varepsilon)a_{i+1}$, for $i \geq 1$, each item profit is at most decreased by a factor of $1-\varepsilon$, and consequently, the solution value of the transformed instance potentially decreases by the same factor of $1-\varepsilon$. Therefore, with $1-\varepsilon$ loss, we can assume that every possible profit is equal to $a_i = \frac{\varepsilon P^H}{\gamma} \cdot (\frac{1}{1-\varepsilon})^{i-1}$ for some $i \geq 1$. Furthermore, since $p_{\max} \leq P^H$, the number of different profits is bounded by the biggest integer β such that

$$\frac{\varepsilon P^H}{\gamma} \cdot (\frac{1}{1-\varepsilon})^{\beta-1} \leq P^H.$$

Since $\ln(\frac{1}{1-\varepsilon}) \geq \varepsilon$, we have $\beta - 1 \leq \frac{\ln(\gamma/\varepsilon)}{\ln(\frac{1}{1-\varepsilon})} \leq \frac{1}{\varepsilon}\ln\frac{\gamma}{\varepsilon}$. In any feasible solution, the largest number n_i of items with profit a_i, for $i = 1, ..., \beta$, is bounded by

$$n_i \leq \min\{\gamma, \left\lfloor \frac{OPT}{\frac{\varepsilon P^H}{\gamma} \cdot (\frac{1}{1-\varepsilon})^{i-1}} \right\rfloor\} \leq \min\{\gamma, \left\lfloor \frac{2\gamma}{\varepsilon}(1-\varepsilon)^{i-1} \right\rfloor\},$$

and we can keep the first n_i items with the smallest weights, and discard the others with no loss. Let $\alpha = \left\lfloor \frac{\ln(\varepsilon/2)}{\ln(1-\varepsilon)} \right\rfloor + 1$. Again, the number of items with profit a_i is at most γ, if a_i is a small profit (i.e. when $i = 1, ..., \alpha$),

and at most $\left\lfloor \frac{2\gamma}{\varepsilon}(1-\varepsilon)^{i-1} \right\rfloor$ otherwise ($i = \alpha+1, ..., \beta$). Therefore, the total number of items in the transformed instance is bounded by

$$\alpha\gamma + \sum_{i=\alpha+1}^{\beta} \left\lfloor \frac{2\gamma}{\varepsilon}(1-\varepsilon)^{i-1} \right\rfloor \leq (\frac{1}{\varepsilon}\ln(2/\varepsilon)+1)\gamma + \frac{\gamma+2\varepsilon-2}{\varepsilon} = O(\frac{\gamma}{\varepsilon}\ln\frac{1}{\varepsilon}).$$

Moreover, we can easily show that this bound is tight. Consider an instance having γ items for each distinct profit $\frac{\varepsilon P^H}{\gamma} \cdot (\frac{1}{1-\varepsilon})^{i-1}$, where $i = 1, ..., \beta$. By applying the described geometric rounding technique, we have exactly $\alpha\gamma$ items with small profits and $\sum_{i=\alpha+1}^{\beta} \left\lfloor \frac{2\gamma}{\varepsilon}(1-\varepsilon)^{i-1} \right\rfloor$ with large profits. What remains to be shown is to bound the number of items to be $\Omega(\frac{\gamma}{\varepsilon}\ln\frac{1}{\varepsilon})$:

$$\alpha\gamma + \sum_{i=\alpha+1}^{\beta} \left\lfloor \frac{2\gamma}{\varepsilon}(1-\varepsilon)^{i-1} \right\rfloor \geq \frac{\ln(\varepsilon/2)}{\ln(1-\varepsilon)}\gamma = \Omega(\frac{\gamma}{\varepsilon}\ln\frac{1}{\varepsilon}).$$

We see that by applying the geometric rounding we have at most γ/ε items with large profit, while $O(\frac{\gamma}{\varepsilon}\ln\frac{1}{\varepsilon})$ items with small profits. Contrary to arithmetic rounding, the set of items that has been reduced most is the set with large profits. This suggests us to combine the described rounding techniques as described in the following subsection.

7.2.3 Parallel Arithmetic & Geometric Rounding

We use arithmetic rounding for the set of items with small profits and geometric rounding for large items. Let us say that these two techniques are applied in "parallel". More formally, let us consider the *Parallel Arithmetic & Geometric* sequence $S_{ag}(\gamma) = (a_1, a_2, ...)$ defined by setting

$$a_i = \frac{\varepsilon P^H}{\gamma} \cdot \begin{cases} i & \text{for } i = 1, ..., \lfloor 2/\varepsilon \rfloor, \\ (\frac{1}{1-\varepsilon})^{\alpha+i-\lfloor 2/\varepsilon \rfloor - 1} & \text{otherwise.} \end{cases}$$

We round each profit down to the nearest value among those of $S_{ag}(\gamma)$. Now, consider every set N_i of items with the same rounded profit value a_i, and take the first

$$n_i = \begin{cases} \gamma & \text{for } i = 1, ..., \lfloor 2/\varepsilon \rfloor, \\ \left\lfloor \frac{2\gamma}{\varepsilon}(1-\varepsilon)^{\alpha+i-\lfloor 2/\varepsilon \rfloor - 1} \right\rfloor & \text{otherwise,} \end{cases}$$

items with the smallest weight. Selecting the first n_i items with the smallest weight can be done in $O(|N_i|)$ time. That is, $O(|N_i|)$ time is sufficient to

select the n_i-th item with the smallest weight (see [7]) and only $O(|N_i|)$ comparisons are needed to extract the $n_i - 1$ items with smaller weight. Therefore the amortized time is linear.

By using the arithmetic rounding technique for small items, we have at most $2\gamma/\varepsilon$ small items with $1-\varepsilon$ loss (see Section 7.2.1). While, by using the geometric rounding technique described in Section 7.2.2 for large items, we have at most γ/ε large items with $1-\varepsilon$ loss. The resulting transformed instance has at most $3\gamma/\varepsilon$ items with $1-2\varepsilon$ loss. Furthermore, let $\psi = \beta - \alpha + \lfloor 2/\varepsilon \rfloor + 1$. Observe that the ψ-th element of $S_{ag}(\gamma)$ is not smaller than P^H, i.e., $a_\psi \geq P^H$. Consider any subset $S \subseteq N$ of items with at most γ items, and let x_i denote the total number of items from S with profit in interval $[a_i, a_{i+1})$, $i = 1, 2, ...\psi$. Let us call vector $(x_1, x_2, ..., x_\psi)$ an *S-configuration*. It is easy to see that by using the reduced set of items it is always possible to compute a solution having the same S-configuration, as any set $S \subseteq N$ with γ items. Summarizing:

Lemma 7.1 *For any positive integer $\gamma \leq k$, it is possible to compute in linear time a reduced set $N_\gamma \subseteq N$ of items with at most $3\gamma/\varepsilon$ items, such that, for any subset $S \subseteq N$ with at most γ items, there exists a subset $S_\gamma \subseteq N_\gamma$ such that S_γ is the subset of N having the same configuration as S and with the least weights.*

Corollary 7.1 *For any subset $S \subseteq N$ with at most γ items, there exists a subset $S_\gamma \subseteq N_\gamma$ with $w(S_\gamma) \leq w(S)$, $|S_\gamma| = |S|$ and $p(S_\gamma) \geq p(S) - 2\varepsilon \cdot OPT$.*

7.3 An Improved PTAS for kKP

Our PTAS for kKP improves the scheme of Caprara et al. [10], and in fact it strongly builds on their ideas. However, there are several differences where a major one is the use of two reduced sets of items instead of the entire set N: let $\ell := \min\{\lceil 1/\varepsilon \rceil - 2, k\}$, where $\varepsilon \leq 1/2$ is an arbitrary small rational number; our algorithm uses sets N_k and N_ℓ computed by using the Arithmetic & Geometric rounding technique (see Lemma 7.1) when $\gamma := k$ and $\gamma := \ell$, respectively.

For any given instance of kKP, the approximation scheme performs the following five steps (S-1)-(S-5).

(S-1) Initialize the solution A to be the empty set and set the corresponding value P^A to 0.

(S-2) Compute the reduced sets N_k and N_ℓ.

(S-3) Consider each $L \subset N_\ell$ such that $|L| \leq \ell - 1$. If $w(L) \leq c$ and $p(L) > P^A$ let $A := L$, $P^A := p(A)$.

(S-4) Consider each $L \subseteq N_\ell$ such that $|L| = \ell$. If $w(L) \leq c$, consider sequence $S_{ag}(\ell) = (a_1, a_2, ...)$ and let h be an integer such that $a_h \leq \min_{j \in L} p_j < a_{h+1}$. Apply algorithm $H^{\frac{1}{2}}$ to the subinstance S defined by item set $\{i \in N_k \backslash L : p_i < a_{h+1}\}$, by capacity $c - w(L)$ and by cardinality upper bound $k - \ell$. Let T and $P^H(S)$ denote the solution and the solution value returned by $H^{\frac{1}{2}}$ when applied to S. If $p(L) + P^H(S) > P^A$ let $A := L \cup T$ and $P^A := p(L) + P^H(S)$.

(S-5) Return solution A of value P^A.

Observe that in steps (S-3) and (S-4), subsets L are computed by considering just the items from N_ℓ. On the other hand, in step (S-4), we remark that the subinstances S are defined by using items from N_k.

7.3.1 Analysis of the Algorithm

Step (S-2) can be performed in $O(n)$ time by Lemma 7.1. In step (S-3) the algorithm considers $O(|N_\ell|^{\ell-1})$ subsets, for each one performing operations that clearly require $O(\ell)$ time. In step (S-4) the algorithm considers $O(|N_\ell|^\ell)$ subsets L. For each L the definition of subinstance S requires $O(|N_k| \cdot \ell)$ time. Algorithm $H^{\frac{1}{2}}$ applied to subinstance S runs in $O(|S|) = O(|N_k|)$ time [10]. By Lemma 7.1, $|N_k| = O(k/\varepsilon)$ and $|N_\ell| = O(\ell/\varepsilon)$. Therefore, step (S-4) is performed in $O(|N_\ell|^\ell \cdot |N_k| \cdot \ell) = O(k \cdot (\frac{\ell}{\varepsilon})^{\ell+1}) = k \cdot (1/\varepsilon)^{O(1/\varepsilon)}$. It follows that the overall running time of the algorithm is $O(n + k \cdot (1/\varepsilon)^{O(1/\varepsilon)})$, and it is not difficult to check that steps (S-1)-(S-5) require linear space.

What remains to be shown is that steps (S-1)-(S-5) return a $(1 - O(\varepsilon))$-approximate solution.

If an optimal solution contains at most ℓ items, the solution returned is $(1 - 2\varepsilon)$-approximate solution. Indeed, by Corollary 7.1 the reduction of N to N_ℓ results in at most $(1 - 2\varepsilon)$ loss of profit, and steps (S-3)-(S-4) consider all subsets of N_ℓ having at most ℓ items.

Otherwise, let $\{j_1, ..., j_\ell, ...\}$ be the set of items in an optimal solution ordered so that $p_{j_1} \geq ... \geq p_{j_\ell} \geq ...$, and let $L^* = \{j_1, ..., j_\ell\}$ be the subset obtained by picking the first ℓ items with the largest profit. Consider subinstance S^* defined by item set $\{i \in N \backslash L^* : p_i \leq \min_{j \in L^*} p_j\}$, by capacity $c - w(L^*)$ and by cardinality upper bound $k - \ell$. Clearly,

$$p(L^*) + OPT_{S^*} = OPT, \tag{7.6}$$

where OPT_{S^*} denotes the optimal value of instance S^*. Now, consider the reduced set N_k and subinstance S_k^* defined by item set $\{i \in N_k \backslash L^* : p_i \leq \min_{j \in L^*} p_j\}$, by capacity $c - w(L^*)$ and by cardinality upper bound $k - \ell$. By Corollary 7.1, we have

$$OPT_{S_k^*} \geq OPT_{S^*} - 2\varepsilon OPT, \tag{7.7}$$

where $OPT_{S_k^*}$ denotes the optimal value of instance S^*.

Let us use L to denote the set of items having the same configuration as L^* and the least weight. By Lemma 7.1, in one of the iterations of step (S-4), set L is considered. In the remainder, let us focus on this set L, and consider the corresponding subinstance S defined in step (S-4). By Corollary 7.1, we have

$$p(L) \geq p(L^*) - 2\varepsilon OPT. \tag{7.8}$$

We need to show that the optimal solution value OPT_S of instance S cannot be smaller than $OPT_{S_k^*}$.

Lemma 7.2 $OPT_S \geq OPT_{S_k^*}$.

Proof. Recall that the subinstance S is defined by item set $I_S = \{i \in N_k \backslash L : p_i < a_{h+1}\}$, where a_{h+1} is a term of sequence $S_{ag}(\ell) = (a_1, a_2, ...)$ such that $a_h \leq \min_{j \in L} p_j < a_{h+1}$ (see step (S-4)). On the other hand, the subinstance S_k^* is defined by item set $I_{S_k^*} = \{i \in N_k \backslash L^* : p_i \leq \min_{j \in L^*} p_j\}$. Since we are assuming that L has the same configuration as L^*, there are no items from L^* with profit in intervals $[a_i, a_{i+1})$, for $i < h$. Therefore, we have $\min_{j \in L^*} p_j \geq a_h$ and $\{i \in N_k \backslash L : p_i \leq a_h\} = \{i \in N_k \backslash L^* : p_i \leq a_h\}$. Furthermore, since there is an item from L with profit in interval $[a_h, a_{h+1})$, and since L^* has the same configuration as L, there exists an item from L^* with profit $p_j < a_{h+1}$ and, therefore, $\min_{j \in L^*} p_j < a_{h+1}$. It follows that $I_{S_k^*} \subseteq \{i \in N_k \backslash L^* : p_i < a_{h+1}\}$.

By the previous arguments, the items of S_k^*, except those belonging to $A_h = \{i \in N_k \cap L : a_h \leq p_i < a_{h+1}\}$, are also items of S, i.e.,

$$I_{S_k^*} \subseteq I_S \cup A_h.$$

If there exists an optimal solution for S_k^* such that no one of the items from A_h is selected, then $OPT_S \geq OPT_{S_k^*}$, since the knapsack capacity of S_k^* is not greater than the one of S, i.e. $c - w(L) \geq c - w(L^*)$ (recall that L is the subset having the same configuration as S^* with the least weight).

Otherwise, let G_1 be the subset of items from A_h in an optimal solution for S_k^*, and let $g := |G_1|$. Let G_2 be any subset of $\{i \in L^* \backslash L : a_h \leq$

$p_i < a_{h+1}\}$ containing exactly g items. It is easy to see that G_2 exists. Furthermore, since $G_2 \subseteq L^*$ and $G_1 \subseteq I_{S_k^*}$, we have

$$\min_{j \in G_2} p_j \geq \max_{j \in G_1} p_j. \tag{7.9}$$

Observe that $w(L^*) - w(L) \geq w(G_2) - w(G_1)$. Therefore, the knapsack capacity $c - w(L)$ of S cannot be smaller than $c - w(L^*) + w(G_2) - w(G_1)$. The solution G_{12} obtained from the optimal solution for S_k^* by replacing the items from G_1 with those from G_2, requires a knapsack of capacity bounded by $c - w(L^*) + w(G_2) - w(G_1)$. Therefore, G_{12} is a feasible solution for S since the capacity of S is greater than the capacity of S_k^* by at least $w(G_2) - w(G_1)$. Finally, from inequality (7.9), the solution value of G_{12} is not smaller than $OPT_{S_k^*}$ and the claim follows. ■

Let $P^H(S)$ denote the solution value returned by $H^{\frac{1}{2}}$ when applied to S. Then we have the following

Lemma 7.3 $p(L) + P^H(S) \geq (1 - 4\varepsilon)OPT$.

Proof. Observe that by Lemma 7.2 and inequality (7.7), we have

$$OPT_S \geq OPT_{S^*} - 2\varepsilon OPT. \tag{7.10}$$

We distinguish between two cases.

1. If $p(L^*) \geq (1 - \varepsilon)OPT$ then by inequalities (7.6), (7.8) and (7.10), we have

$$\begin{aligned} p(L) + P^H(S) &\geq p(L^*) - 2\varepsilon OPT + \frac{1}{2}OPT_S \\ &\geq (1 - \varepsilon)OPT - 2\varepsilon OPT = (1 - 3\varepsilon)OPT. \end{aligned}$$

2. If $p(L^*) < (1 - \varepsilon)OPT$ then the smallest item profit in L^*, and hence every item profit in S^*, is smaller than $\frac{(1-\varepsilon)}{\ell}OPT$. The largest profit in S is at most $\frac{(1-\varepsilon)}{\ell}OPT + \frac{\varepsilon P^H}{\ell}$. Indeed, since $a_h \leq \frac{(1-\varepsilon)}{\ell}OPT \leq 2(1-\varepsilon)\frac{P^H}{\ell} \leq (\lfloor \frac{2}{\varepsilon} \rfloor - 1)\frac{\varepsilon P^H}{\ell}$, it turns out that $h \leq \lfloor \frac{2}{\varepsilon} \rfloor - 1$, and by definition of $S_{ag}(\ell)$, we have that $a_{h+1} = a_h + \frac{\varepsilon P^H}{\ell}$. Therefore, for each item j belonging to S, profit p_j is bounded by

$$p_j \leq \frac{OPT}{\ell}.$$

Since $OPT_S - P^H(S) \leq \max_{j \in S} p_j$ (see inequality (7.5)), we have

$$\begin{aligned} p(L) + P^H(S) + \frac{OPT}{\ell} &\geq p(L) + OPT_S \\ &\geq p(L^*) + OPT_{S^*} - 4\varepsilon \cdot OPT = (1 - 4\varepsilon)OPT. \end{aligned}$$

■

By the previous lemma, steps (S-1)-(S-5) return a solution that cannot be worse than $(1 - 4\varepsilon)OPT$. Thus, we have proved the following

Theorem 7.1 *There is an PTAS for the k-item knapsack problem requiring linear space and* $O(n + k \cdot (1/\varepsilon)^{O(1/\varepsilon)})$ *time.*

To compare our algorithm with the one provided in [10] notice that the running time complexity of the latter is $O(n^{\lceil 1/\varepsilon \rceil - 1})$, whereas our scheme is linear. As in [10], our algorithm can be easily modified to deal with the *Exact k-item Knapsack Problem*, that is a kKP in which the number of items in a feasible solution must be exactly equal to k. The time and space complexities, and the analysis of the resulting algorithm are essentially the same as the one described above.

7.4 An Improved FPTAS for kKP

The main goal of this section is to present a different combination of arithmetic and geometric rounding techniques. Moreover we propose an improved fully polynomial time approximation scheme that runs in $O(n + k/\varepsilon^4 + 1/\varepsilon^5)$ time. First we discuss separately the different steps in details, then we state the main algorithm and summarize the results in Section 7.4.2.

We start partitioning the set of items in two subsets $\mathcal{L} = \{j : p_j > \varepsilon P^H\}$ and $\mathcal{S} = \{j : p_j \leq \varepsilon P^H\}$. Let us say that $\mathcal{L}$ is the set of *large* items, while $\mathcal{S}$ the set of *small* items. Observe that the number of large items in any feasible solutions is not greater than $\lambda = \min\{k, \lfloor 2/\varepsilon \rfloor\}$, since $OPT \leq 2P^H$.

7.4.1 Dynamic Programming for Large Items

In principle, an optimal solution could be obtained in the following way. Enumerate all different solutions for items in $\mathcal{L}$, i.e., consider all different sets $U \subseteq \mathcal{L}$ such that $w(U) \leq c$ and $|U| \leq k$. For each of these U, compute a set $T \subseteq \mathcal{S}$ such that $w(T) + w(U) \leq c$, $|U| + |T| \leq k$ and $p(T)$ is maximized. Select from these solutions one with the largest overall profit. One of the

problems with this approach is that constructing all possible solutions for items in $\mathcal{L}$ would require considering $n^{O(1/\varepsilon)}$ cases. To avoid the exponential dependence on $1/\varepsilon$ (our aim is to obtain a fully polynomial approximation scheme), we will not treat separately all of these solutions. We begin with the description of a basic procedure that generates a list of all "interesting" feasible combinations of profit and number of selected large items. Each such combination is represented by a pair (a, l), for which there is a subset of items $U \subseteq \mathcal{L}$ with $p(U) = a$, $|U| = l$ and $w(U) \leq c$. Moreover $w(U)$ is the smallest attainable weight for a subset of large items with profit at least equal to a and cardinality at most l. This list of all "interesting" feasible combinations is computed by using a pseudopolynomial dynamic programming scheme. Clearly, an optimal solution can be computed by using only the subsets U of large jobs associated to each pair (a, l). The time complexity will be then reduced, with $1 - O(\varepsilon)$ loss, by applying arithmetic and geometric rounding techniques, as described in Section 7.4.1.

Let α be the number of large items, and let β denote the number of all distinct feasible solution values obtained by considering only large items, i.e. β is the size of set

$$V = \{p(U) | U \subseteq \mathcal{L} \text{ and } w(U) \leq c \text{ and } |U| \leq k\}.$$

A straightforward dynamic programming recursion which has time complexity $O(\alpha\beta\lambda)$ and space complexity $O(\lambda^2\beta)$ (see [10]), can be stated as follows. Let us renumber the set of items such that the first $1, ..., |L|$ items are large. Denote by function $g_i(a, l)$ for $i = 1, ..., |L|$, $a \in V$, $l = 1, ..., \lambda$, the optimal solution of the following problem:

$$g_i(a, l) = \min \left\{ \sum_{j=1}^{i} w_j x_j : \sum_{j=1}^{i} p_j x_j = a; \sum_{j=1}^{i} x_j = l; x_j \in \{0, 1\}, j = 1, ..., i \right\}.$$

One initially sets $g_0(a, l) = +\infty$ for all $l = 0, ..., \lambda$, $a \in V$, and then $g_0(0, 0) = 0$. Then, for $i = 1, ..., |L|$ the entries for g_i can be computed from those of g_{i-1} by using the formula

$$g_i(a, l) = \min \left\{ \begin{array}{l} g_{i-1}(a, l), \\ g_{i-1}(a - p_i, l - 1) + w_i \quad \text{if } l > 0 \text{ and } a \geq p_i \end{array} \right\}.$$

Since $\beta = O(P^H)$ the described dynamic programming algorithm is only pseudopolynomial. In order to reduce the time complexity, we first preprocess large items by using a combination of arithmetic and geometric rounding techniques, then we apply the above dynamic programming scheme. We

start analyzing the two rounding techniques separately, then we show how to combine them.

Geometric Rounding

The time complexity of the described dynamic programming can be reduced by decreasing the number α of large items and the number β of distinct solution values.

We observed in Section 7.2 that if we want to reduce as much as possible the number of large items it is convenient to use geometric rounding. Consider the geometric sequence $S_g(\gamma)$ described in Section 7.2.2. By applying the geometric rounding technique with $\gamma = \lambda$, the number α of large items can be reduced from $O(n)$ to $O(1/\varepsilon^2)$ with $1-\varepsilon$ loss.

The next step is to compute the number of possible solution values after geometric rounding, i.e. the cardinality β of set V after that all profit values of large items have been geometrically rounded. The main result of this section is stated as follows.

Theorem 7.2 *The number of solution values after geometric rounding can be exponential in* $\frac{1}{\varepsilon}$.

By the above theorem it follows that the running time of dynamic programming after geometric rounding is a constant that may depend exponentially on $\frac{1}{\varepsilon}$. Therefore, to avoid this exponential dependence on $\frac{1}{\varepsilon}$, we will look at other rounding techniques.

In the remaining part of this subsection we prove Theorem 7.2. The goal is to derive a lower bound on the number of possible solution values after geometric rounding, i.e. a lower bound on $|V|$. Recall that we defined the geometric sequence $a_i = \frac{\varepsilon P^H}{\lambda}(\frac{1}{1-\varepsilon})^{i-1}$ in Section 7.2.2, and here we assume that $\gamma = \lambda$. We focus on worst-case analysis. With this aim let us consider an instance I that after geometric rounding has at least $\lfloor P^H/a_i \rfloor$ items for each distinct profit value a_i. Moreover, we assume that $w_j = p_j$ for every $j \in \mathcal{L}$, $c = P^H$ and $k \geq 1/\varepsilon$. By definition of instance I we see that every subset U with $p(U) < P^H$ is a feasible solution. Indeed, we have $w(U) < P^H = c$ and $|U| < p(U)/(\min_{j\in\mathcal{L}} p_j) < 1/\varepsilon \leq k$. By the previous arguments we see that $|V|$ is bounded by below by the number of solution values $y = \sum_{i=0}^{\infty} c_i a_{i+1} < P^H$ with $\mathbf{c} = (c_0, c_1, ...) \in \mathbb{N}_0^\infty$, where $\mathbb{N}_0^\infty$ is the set of sequences with non-negative integer components. Inserting a_i and

$\varepsilon = \frac{\varepsilon'}{1+\varepsilon'}$ this is equivalent to

$$\sum_{i=0}^{\infty} c_i \varepsilon'(1+\varepsilon')^i < \lambda(1+\varepsilon'). \tag{7.11}$$

Clearly

$$\sum_{i=0}^{\infty} c_i \varepsilon'(1+\varepsilon')^i < 1. \tag{7.12}$$

implies (7.11), since $\lambda(1+\varepsilon') > 1$.

For simplicity of notation we replace ε' with ε, and we focus on the cardinality of sets of the form

$$R_\varepsilon^d := \left\{ y < 1 : y = \sum_{i=0}^{d-1} c_i \varepsilon(1+\varepsilon)^i, \quad \mathbf{c} \in I\!N_0^d \right\},$$

where $I\!N_0^d$ is the set of d-dimensional vectors $\mathbf{c} = (c_0, c_1, ..., c_{d-1})$ with non-negative integer components. It is more easy to find lower bounds on the set of vectors

$$S_\varepsilon^d := \left\{ \mathbf{c} \in I\!N_0^d : \sum_{i=0}^{d-1} c_i \varepsilon(1+\varepsilon)^i < 1 \right\} \tag{7.13}$$

itself. To consider $|S_\varepsilon^d|$ instead of $|R_\varepsilon^d|$ is justified by the following Lemma.

Lemma 7.4 *For rational and for transcendental $\varepsilon > 0$, the sets R_ε^d and S_ε^d have the same cardinality, i.e. the mapping $f : S_\varepsilon^d \to R_\varepsilon^d$ with $f(\mathbf{c}) = \sum_{i=0}^{d-1} c_i \varepsilon(1+\varepsilon)^i$ is one-to-one and onto.*

Proof for transcendental ε.

$$y = y' \quad \Leftrightarrow \quad \sum_{i=0}^{d-1} b_i x^i = 0 \quad \text{with} \quad b_i := c_i - c_i' \in Z\!\!Z \quad \text{and} \quad x := 1+\varepsilon$$

A real number is said to be transcendental if it is not the root of a polynomial with integer coefficients. For transcendental ε also x is transcendental, which implies $b_i \equiv 0$. Hence, $y = y'$ implies $\mathbf{c} = \mathbf{c}'$. Obviously $\mathbf{c} = \mathbf{c}'$ implies $y = y'$. This proves that S_ε^d has the same cardinality as R_ε^d for transcendental ε.

Proof for rational ε. Assume by contradiction that there are two different vectors $\mathbf{c} \neq \mathbf{c}' \in I\!N_0^d$ with same solution value $y = y'$. Let n be the

largest index i such that $c_i \neq c'_i$. Furthermore, let $\varepsilon = \frac{p}{q} - 1 > 0$ be rational with $p > q \in I\!N$ having no common factors. With $\mathbf{b} := \mathbf{c} - \mathbf{c}'$ we have

$$\begin{aligned} 0 &= [y - y'] = \sum_{i=0}^{d-1} b_i \varepsilon (1+\varepsilon)^i \\ &= \varepsilon \sum_{i=0}^{n} b_i \frac{p^i}{q^i} = \frac{\varepsilon}{q^n} \Big[b_n p^n + q \overbrace{\sum_{i=0}^{n-1} [b_i p^i q^{n-i-1}]}^{integer} \Big] \end{aligned}$$

Since the last term $q \sum[...]$ is a multiple of q, $y - y'$ can only be zero if also $b_n p^n$ is a multiple of q. With p also p^n has no common factor with q, hence b_n must itself be a multiple of q. The sum in (7.13) can only be less than 1 if each term is less than 1, i.e. $c_i \varepsilon (1+\varepsilon)^i < 1$. This implies

$$c_i < \varepsilon^{-1}(1+\varepsilon)^{-i} \leq \varepsilon^{-1} \quad \text{for all} \quad i. \tag{7.14}$$

Together we get

$$0 \leq c_n^{(\prime)} < \frac{1}{\varepsilon} = \frac{q}{p-q} < q \;\Rightarrow\; |b_n| = |c_n - c'_n| < q \;\Rightarrow\; b_n = 0 \;\Rightarrow\; c_n = c'_n$$

which contradicts our assumption $c_n \neq c'_n$. Hence, $y = y'$ implies $\mathbf{c} = \mathbf{c}'$. Again, that $\mathbf{c} = \mathbf{c}'$ implies $y = y'$ is obvious. This shows that S_ε has the same cardinality as R_ε for rational ε. ■

We don't know whether Lemma 7.4 also holds for algebraic ε. The following Lemma lower bounds S_ε^∞.

Lemma 7.5 $|S_\varepsilon^\infty| \geq Ce^{B/\varepsilon}$ *with* $B = 0.3172...$ *and* $C = 0.3200...$

Proof. ¿From (7.14) we see that all c_i are zero for too large i ($c_i = 0 \,\forall i \geq d_{max} := \lceil \frac{\ln(1/\varepsilon)}{\ln(1+\varepsilon)} \rceil$). This shows that $|S_\varepsilon^1| \leq |S_\varepsilon^2| \leq ... \leq |S_\varepsilon^{d_{max}}| = |S_\varepsilon^{d_{max}+1}| = ... = |S_\varepsilon^\infty|$. The main idea in the following is to relate S_ε^d to the volume of a d-dimensional simplex with volume larger than $Ce^{B/\varepsilon}$ for suitable d.

We define a d-dimensional subset $U_\varepsilon^d \subset I\!R^d$, which is the disjoint union of unit cubes $[c_0, c_0 + 1) \times ... \times [c_{d-1}, c_{d-1} + 1)$ for every $\mathbf{c} \in S_\varepsilon^d$. This set can be represented in the following form

$$U_\varepsilon^d := \left\{ \mathbf{r} \in [0,\infty)^d : \sum_{i=0}^{d-1} \lfloor r_i \rfloor \varepsilon (1+\varepsilon)^i < 1 \right\}$$

The volume $\mathrm{Vol}(U_\varepsilon^d)$ coincides with the cardinality of set S_ε^d since each point in S_ε^d corresponds to exactly one unit cube in U_ε^d, each having volume 1. Furthermore, let us define the d-dimensional (irregular) tetrahedron

$$T_\varepsilon^d := \left\{ \mathbf{r} \in [0,\infty)^d : \sum_{i=0}^{d-1} r_i \varepsilon (1+\varepsilon)^i < 1 \right\}$$

Obviously $T_\varepsilon^d \subseteq U_\varepsilon^d$, since $\lfloor r_i \rfloor \leq r_i$. So we have $|S_\varepsilon^\infty| \geq |S_\varepsilon^d| = \mathrm{Vol}(U_\varepsilon^d) \geq \mathrm{Vol}(T_\varepsilon^d)$. The tetrahedron T_ε^d is orthogonal at the vertex $\mathbf{r} = 0$. The edges $\mathbf{r} = (0,...,0,r_i,0,...,0)$ have lengths $[\varepsilon(1+\varepsilon)^i]^{-1}$, $i = 0...d-1$. Hence, the volume of the tetrahedron is

$$\begin{aligned} \mathrm{Vol}(T_\varepsilon^d) &= \frac{1}{d!} \prod_{i=0}^{d-1} [\varepsilon(1+\varepsilon)^i]^{-1} = [d!\, \varepsilon^d (1+\varepsilon)^{d(d-1)/2}]^{-1} \\ &\geq e^{[-(d\varepsilon)\ln(d\varepsilon) - \frac{1}{2}(d\varepsilon)^2]/\varepsilon} = e^{f(d\varepsilon)/\varepsilon} \end{aligned}$$

with $f(x) := -x \ln x - \frac{1}{2}x^2$. In the inequality we replaced $d-1$ by d and used $d! \leq d^d$ and $1+\varepsilon \leq e^\varepsilon$. The best bound is found by maximizing $e^{f(\varepsilon d)/\varepsilon}$ w.r.t. d, or equivalently by maximizing $f(x)$ w.r.t. x. We have $-f'(A) = \ln A + 1 + A = 0$ for $A = 0.2784....$ Hence, d should be chosen as $\frac{A}{\varepsilon}$, but since d is integer we have to round somehow, for instance $d = \lfloor \frac{A}{\varepsilon} \rfloor$. Note that $|S_\varepsilon^d|$ increases with d, but our approximation becomes crude for d near d_{max}. This is the reason why the maximizing d is less than d_{max}. For small ε we have $f(\varepsilon \lfloor \frac{A}{\varepsilon} \rfloor) \approx f(\varepsilon \frac{A}{\varepsilon}) = f(A) = -A \ln A - \frac{1}{2}A^2 = A(1 + \frac{1}{2}A) =: B = 0.3172...$ with corrections of order $O(\varepsilon)$. This establishes an asymptotic bound $\sim e^{B/\varepsilon}$. More exactly, one can show that $f(\varepsilon \lfloor \frac{A}{\varepsilon} \rfloor) \geq f(A)(1 - \frac{\varepsilon}{A})$ for all ε. This yields the bound

$$|S_\varepsilon^\infty| \geq \max_d \mathrm{Vol}(T_\varepsilon^d) \geq e^{f(A)(1-\frac{\varepsilon}{A})/\varepsilon} = Ce^{B/\varepsilon} \quad \text{with} \quad C = e^{-B/A} = 0.3200...$$

■

The coefficient B can be improved to 0.7279... for sufficiently small ε by using the more accurate Stirling approximation for $d!$.

Using Lemma 7.4 and 7.5 it is now easy to lower bound the number of possible solution values for geometric profit distribution. ¿From Lemma 7.5 we know that (7.12) has at least $Ce^{B/\varepsilon}$ solution vectors $\mathbf{c}$ and from Lemma 7.4 that (7.11) has at least $Ce^{B/\varepsilon}$ solution values y for rational ε, and the proof of Theorem 7.2 follows.

Arithmetic Rounding

Alternatively, we may think to apply arithmetic rounding to the set of large items. Let us consider the arithmetic sequence $S_a(\gamma)$ described in Section 7.2.1. By applying the arithmetic rounding technique with $\gamma = \lambda$, we observe the number of large items can be reduced to be bounded by $O(\frac{1}{\varepsilon^2} \ln \frac{1}{\varepsilon})$ with $1-\varepsilon$ loss. Moreover each element of set V is equal to $\frac{\varepsilon P^H}{\lambda} i$ for some $i = \lambda, \lambda+1, ..., 2\lfloor \lambda/\varepsilon \rfloor$. It follows that the size of set V is bounded by $O(1/\varepsilon^2)$, and the overall time of the dynamic programming algorithm is now $O(\frac{1}{\varepsilon^5} \ln \frac{1}{\varepsilon})$. We see that in comparison to the geometric rounding and although the number of large items is larger, the arithmetic rounding technique is able to reduce much more the size of set V. However and again, we can take advantage from both techniques by combining them as described in the following.

Serial Geometric & Arithmetic Rounding

We first apply geometric rounding with $1-\varepsilon$ loss. This reduces the number of large items to be bounded by $O(1/\varepsilon^2)$. Then, with $1-\varepsilon$ loss, we apply arithmetic rounding on the reduced set of large items. Clearly the latter does not increase the number of items and each profit value is now equal to $\frac{\varepsilon P^H}{\lambda} i$ for some $i = \lambda, \lambda+1, ..., 2\lfloor \lambda/\varepsilon \rfloor$. By using this set of items with profits rounded by using geometric first and arithmetic rounding then, the size of set V has a bound of $O(1/\varepsilon^2)$, and the overall time of the dynamic programming algorithm is $O(1/\varepsilon^5)$. We call this combination a *Serial Geometric & Arithmetic rounding technique*.

7.4.2 Adding Small Items

In the following we show how to add the small items. First, with $1-2\varepsilon$ loss, we reduce the number of small items to be $O(k/\varepsilon)$ by using the Parallel Arithmetic & Geometric rounding (see Section 7.2.3). Then, for each pair (a, l) in the final list, fill in the remaining knapsack capacity $c - g_{|L|}(a, l)$ with at most $k - l$ small items, by using algorithm $H^{\frac{1}{2}}$ for kKP [10]. These small items yield total profit $P^H(c - g_{|L|}(a, l), k - l)$. By inequality (7.5) and by definition of small items, we have

$$P^H(c - g_{|L|}(a, l), k - l) + \varepsilon P^H \geq OPT(c - g_{|L|}(a, l), k - l), \qquad (7.15)$$

where $OPT(c - g_{|L|}(a, l), k - l)$ is the optimal solution value obtained by using at most $k - l$ small items and knapsack capacity $c - g_{|L|}(a, l)$. The

approximate solution, a combination of large and small items, is chosen to yield profit P, where

$$P = \max_{(a,l)} \left\{ a + P^H(c - g_{|L|}(a,l), k - l) \right\}$$

By inequality (7.15) and since our algorithms considers all the "interesting" pairs (a,l) with $1 - O(\varepsilon)$ loss, it is easy to verify that P is $1 - O(\varepsilon)$ times the optimal solution.

To summarize, the steps of the FPTAS are as follows.

(S-1) Partition the set of items into "large" and "small". Apply the Serial Geometric & Arithmetic rounding technique to the set of large items. Apply the Parallel Arithmetic & Geometric rounding technique to the set of small items.

(S-2) Solve for the "large" items using dynamic programming: generate a list of all "interesting" feasible combinations (a,l) of profit a and number l of selected large items.

(S-3) For each pair (a,l) in the final list, fill in the knapsack by applying algorithm $H^{\frac{1}{2}}$ with the reduced set small items.

(S-4) Return the best found solution.

Step (S-1) can be performed in $O(n)$ time. Step (S-2) takes $O(1/\varepsilon^5)$ time. Algorithm $H^{\frac{1}{2}}$ applied to the reduced set of small items runs in $O(k/\varepsilon)$ time [10]. In step (S-3) the algorithm considers $O(1/\varepsilon^3)$ pairs, for each one performing operations that require $O(k/\varepsilon)$ time. It follows that the overall running time of the algorithm is $O(n + k/\varepsilon^4 + 1/\varepsilon^5)$. The space complexity has a bound of $O(n + 1/\varepsilon^4)$, since the space required by the dynamic programming is $O(\lambda^2\beta)$ where $\lambda = O(1/\varepsilon)$ and $\beta = O(1/\varepsilon^2)$.

Theorem 7.3 *There is a fully polynomial time approximation scheme for the k-item knapsack problem requiring* $O(n + k/\varepsilon^4 + 1/\varepsilon^5)$ *time and* $O(n + 1/\varepsilon^4)$ *space.*

Chapter 8

Compilable Problems

In this thesis we have presented polynomial time approximation schemes for several NP-complete scheduling problems. For some of them we were able to provide an EPTAS, while for others we conjecture that only a PTAS is possible. While algorithms of the first kind can return in a reasonable amount of time a good approximation for an enormous instance, the second ones becomes useless even for moderate values of $1/(r-1)$ and n. However, the running time of an EPTAS may also depend arbitrarily on $1/(r-1)$ with better approximations yielding longer running times. In some cases we may construct approximation schemes with running times that are polynomial both in the size of the instance and in $1/(r-1)$. In such cases the possibility of approaching the optimal solution with arbitrarily small error is more concrete. Fully polynomial time approximation schemes (FPTAS) are polynomial time approximation schemes having running time growing polynomially with the reciprocal of the error bound. Some problems are known not to have fully polynomial time approximation schemes unless P=NP. This lower bound applies to strongly NP-complete problems that fulfill some weak and natural supplementary conditions (see Garey & Johnson [22] and Chapter 1).

In contrast to the latter negative result, we present the following surprising positive result for the class EPTAS. We show that if a problem $\mathcal{P}$ belongs to EPTAS then, for every fixed $r > 1$, it is possible to obtain in constant time an r-approximation algorithm that runs in polynomial time, and where the hidden constant is reasonably small and independent of the accuracy r. The main idea is the following. For every fixed $r > 1$, partition the set of instances of $\mathcal{P}$ in two subsets: the set of *small instances*, i.e., the set of instances whose length is smaller than a constant C_r that depends

on r, and the set of *large instances*, i.e., the remaining ones. Compute and memorize in a look-up table T_r the small instance solutions. When table T_r has been built (the time required to build it is constant) we say that problem $\mathcal{P}$ has been *compiled* and, by using table T_r we get a table-lookup r-approximate algorithm for $\mathcal{P}$ whose polynomial running time is independent of the accuracy r. Note that unless we are careful we can arrive at contrasting complexity results. Here we claim that there are strongly NP-complete problems which admit, for every fixed $r > 1$, an algorithm that returns an r-approximate solution in time independent on r. Actually this algorithm runs in fully polynomial time, but the time required to obtain it is bounded by a constant that is exponential in $1/(r-1)$, which makes this result possible.

Theorem 8.1 *If problem $\mathcal{P}$ belongs to class EPTAS then, for every fixed $r > 1$, it is possible to obtain in constant time an r-approximation algorithm that runs in polynomial time and where the hidden constant is reasonably small and independent of the accuracy r.*

Proof. We start with the following observation. Let $f : \mathbb{R} \to \mathbb{R}$ denote an arbitrary function and c a constant that is independent on r, then we have the following: $f(r)n^c \leq f(r)^{c+1} + n^{c+1}$. Indeed if $f(r) \leq n$ then $f(r)n^c \leq n^{c+1}$. Otherwise if $f(r) > n$ then $f(r)n^c < f(r)^{c+1}$. By the previous observations the running time of every EPTAS can be bounded by $O(n^k + f(r))$, for some constant k and function $f(r)$ and where n is the size of the given input. Now, assume, without loss of generality, that a problem admits an EPTAS returning an r-approximate solution in time $O(n^k + f(r))$. We distinguish between two cases. First, if the length n of any given input is greater than $\sqrt[k]{f(r)}$ then $n^k + f(r) < 2n^k$. Therefore, in this case the given EPTAS runs in $O(n^k)$ time, where the constant hidden in $O(n^k)$ is reasonably small and independent of the accuracy $(r-1)$. Second, if $n^k \leq f(r)$ then we do as follows. For a string x, we denote by $|x|$ the length of its binary encoding. We start observing that there is only a constant number of instances x with $|x|^k \leq f(r)$. Let us say that an instance is *small* if $|x|^k \leq f(r)$. We solve all these small instances and make a table T_r containing their optimal solutions. This table can be clearly computed in constant time. In the following we show that by using table T_r we can obtain an algorithm that returns an optimal solution $opt(x)$ for any given instance x with $|x|^k \leq f(r)$ in $2+|x|+|opt(x)|$ steps. Note that if the length of any optimal solution is bounded by a linear function of the input size then we have an r-approximate solution in linear time.

In order to formalize this algorithm, we consider as model for computation the deterministic one-tape Turing machine (DTM). It consists of a *finite state control*, a *read-write head*, and a *tape* made up of an infinite sequence of *tape squares* (see, e.g. [19]). Clearly, we have that the number c of small instances is bounded by $O(2^{\sqrt[k]{f(r)}})$. Let $i_1, ..., i_c$ denote the small instances in lexicographic order. A program for a DTM is specified by providing the following information:

1. a finite set $\Gamma = \{0, 1, blank\}$ of tape symbols;
2. a finite set Q of states:

$$Q = \{q_0, q_{1,1}^{in}, ..., q_{c,|i_c|}^{in}, q_{1,1}^{out}, ..., q_{c,|i_c|}^{out}, q_f\};$$

3. a transition function $\delta : (Q - \{q_f\}) \times \Gamma \rightarrow Q \times \Gamma \times \{left, right\}$.

The operation of such a program is straightforward. The input to the DTM is a string x which we assume, without loss of generality, encoded in binary. The string x is placed in tape squares 1 through $|x|$, one symbol per square. All other squares initially contain the blank symbol. The program starts its operation in state q_0, with the read-write head scanning tape square 1. Now, let i_j be the first small instance, according to the lexicographic order, having as first symbol the symbol in square 1. Then the finite state control changes its state from q_0 to $q_{j,1}^{in}$ and the read-write head moves one square to the right. The computation then proceeds in a step-by-step manner. Assume that the current state is $q_{j,k}^{in}$. Then the next state is $q_{j,k+1}^{in}$ if the symbol of the current square is equal to the $(k+1)$-st symbol of i_j. Otherwise and if the symbol of the current square is not blank, the state changes to $q_{h,k+1}^{in}$, where $q_{h,k+1}^{in}$ is the state corresponding to the next (according to the lexicographic order) instance i_h having the first k symbols as i_j and the $(k+1)$-st equal to the symbol in the current square. In both cases, the read-write head moves one square to the right. Finally, if the current square contains the blank symbol then x is instance i_j and we write its optimal solution $opt(i_j)$ as follows: move to the right and change state to $q_{j,1}^{out}$; if we are in state $q_{j,k}^{out}$ then write the k-th symbol of solution $opt(i_j)$ and move to the right. This is repeated until state $q_{j,t}^{out}$ is reached, where $t = |opt(i_j)| = |opt(x)|$. Afterwards, the state changes to state q_f and the computation ends. The total number of steps is $2 + |x| + |opt(x)|$. ■

We would like to end this chapter by observing the following. Assume that a given problem $\mathcal{P}$ admits an EPTAS running in time $O(|x| + f(r))$ and

such that $|opt(x)| = O(|x|)$, where x is any given instance of problem $\mathcal{P}$. Then by the previous arguments, it is possible to obtain in constant time an r-approximate algorithm for problem $\mathcal{P}$ that runs in linear time, where the hidden constant is reasonably small and independent of the accuracy r. We say that these approximation algorithms are *optimal*, since their time complexity is the best possible. In this thesis we obtained for several problems ($1|r_j, prec|L_{\max}$, $P|r_j|L_{\max}$, $Rm||C_{\max}$ with costs and $Jm|op \leq \mu|C_{\max}$) linear time EPTAS where the constant that depends on r is additive and $|opt(x)| = O(|x|)$, for any given instance x. Therefore we have the following

Corollary 8.1 *For any fixed $r > 1$, it is possible to obtain in constant time an optimal r-approximate algorithms for problems $1|r_j, prec|L_{\max}$, $P|r_j|L_{\max}$, $Rm||C_{\max}$ with costs and $Jm|op \leq \mu|C_{\max}$.*

Bibliography

[1] N. Alon, Y. Azar abd G.J. Woeginger, and T. Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1:55–66, 1998.

[2] K.M. Anstreicher. Linear programming in $o(\frac{n^3 \ln n}{l})$ *operations. SIAM Journal on Optimization*, 9:803–812, 1999.

[3] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45:753–782, 1998.

[4] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. In *Proceedings of the 33rd Annual IEEE Symposium on the Foundations of Computer Science*, pages 14–23, 1992.

[5] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation.* Springer, 1999.

[6] B.S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994.

[7] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7:448–461, 1973.

[8] P. Brucker. *Scheduling algorithms.* Springer, 1997.

[9] P. Brucker, B. Jurisch, and A. Kramer. Complexity of scheduling problems with multi-purpose machines. *Annals of Operations Research*, 70:57–73, 1997.

[10] A. Caprara, H. Kellerer, U. Pferschy, and D. Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operations Research*, 123:333–345, 2000.

[11] M. Cesati and L. Trevisan. On the efficiency of polynomial time approximation schemes. *Information Processing Letters*, 64(47):165–171, 1997.

[12] Z.L Chen, Q. Lu, and G. Tang. Single machine scheduling with discretely controllable processing times. *Operation Research Letters*, 21:69–76, 1997.

[13] T.C.E. Cheng and N. Shakhlevich. Proportionate flow shop with controllable processing times. *Journal of Scheduling*, 2:253–265, 1999.

[14] T.H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press and McGraw-Hill Book Company, 6th edition, 1992.

[15] W. Fernandez de la Vega and G.S. Lueker. Bin packing can be solved within $1 + \varepsilon$ in linear time. *Combinatorica*, 1:349–355, 1981.

[16] R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer, 1998.

[17] A. Fishkin, K. Jansen, and M. Mastrolilli. Grouping techniques for scheduling problems: simpler and faster. In *9th Annual European Symposium on Algorithms (ESA 2001)*, volume LNCS 2161, pages 206–217, 2001.

[18] L.M. Gambardella, M. Mastrolilli, A. Rizzoli, and M. Zaffalon. An optimization methodology for intermodal terminal management. *Journal of Intelligent Manufacturing*, 12:521–534, 2001.

[19] M. R. Garey and D. S. Johnson. *Computers and intractability; a guide to the theory of NP-completeness*. W.H. Freeman, 1979.

[20] M.R. Garey, R.L. Graham, and J.D. Ullman. Worst-case analysis of memory allocation algorithms. In *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*, pages 143–150, Denver, Colorado, 1972.

[21] M.R. Garey and D.S. Johnson. The complexity of near-optimal graph coloring. *Journal of the ACM*, 23(1):43–49, 1976.

[22] M.R. Garey and D.S. Johnson. Strong NP-completeness results: Motivation, examples and implications. *J. ACM*, 25:499–508, 1978.

[23] L.A. Goldberg, M. Paterson, A. Srinivasan, and E. Sweedyk. Better approximation guarantees for job-shop scheduling. *SIAM Journal on Discrete Mathematics*, 14(1):67–92, 2001.

[24] T. Gonzales and S. Sahni. Flowshop and jobshop schedules: complexity and approximation. *Operations Research*, 26:36–52, 1978.

[25] R. Graham, E. Lawler, J.K. Lenstra, and A. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. In *Annals of Discrete Mathematics*, volume 5, pages 287–326. North–Holland, 1979.

[26] R.L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal (BSTJ)*, 45:1563–1581, 1966.

[27] R.L. Graham. Bounds on multiprocessor timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.

[28] M. D. Grigoriadis and L. G. Khachiyan. Coordination complexity of parallel price-directive decomposition. *Mathematics of Operations Research*, 21:321–340, 1996.

[29] L.A. Hall and D.B. Shmoys. Approximation algorithms for constrained scheduling problems. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, pages 134–139, 1989.

[30] L.A. Hall and D.B. Shmoys. Near-optimal sequencing with precedence constraints. In *Proceedings of the 1st Integer Programming and Combinatorial Optimization Conference*, pages 249–260. University of Waterloo Press, 1990.

[31] L.A. Hall and D.B. Shmoys. Jackson's rule for single-machine scheduling: Making a good heuristic better. *MOR: Mathematics of Operations Research*, 17:22–35, 1992.

[32] D.S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. ITP, 1995.

[33] D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.

[34] D.S. Hochbaum and D.B. Shmoys. A polynomial approximation scheme for machine scheduling on uniform processors: Using the dual approximation approach. *SIAM J. on Computing*, 17:539–551, 1988.

[35] E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM*, 21(2):277–292, 1974.

[36] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the ACM*, 23(2):317–327, 1976.

[37] O.H. Ibarra and C.E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22:463–468, 1975.

[38] J.R. Jackson. Scheduling a production line to minimize maximum tardiness. Technical Report Research Report 43, Management Science Research Project, UCLA, 1955.

[39] K. Jansen and M. Mastrolilli. Parallel machine scheduling problems with controllable processing times. In *Proceedings of ICALP Workshops 2000 (ARACNE)*, pages 179–189, 2000.

[40] K. Jansen, M. Mastrolilli, and R. Solis-Oba. Approximation algorithms for flexible job shop problems. In *Proceedings of Latin American Theoretical Informatics 2000*, pages 68–77, 2000.

[41] K. Jansen, M. Mastrolilli, and R. Solis-Oba. Job shop scheduling problems with controllable processing times. In *Proceedings of the 7th Italian Conference on Theoretical Computer Science*, volume LNCS 2202, pages 107–122, 2001.

[42] K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. In *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing*, pages 408–417, 1999.

[43] K. Jansen and L. Porkolab. Linear-time approximation schemes for scheduling malleable parallel tasks. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 490–498, 1999.

[44] K. Jansen, R. Solis-Oba, and M. Sviridenko. A linear time approximation scheme for the job shop scheduling problem. In *APPROX'99*, volume LNCS 1671, pages 177–188, 1999.

[45] K. Jansen, R. Solis-Oba, and M. Sviridenko. Makespan minimization in job shops: a polynomial time approximation scheme. In *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing*, pages 394–399, 1999.

[46] D.S. Johnson. Approximate algorithms for combinatorial problems. *Journal of Computer and Systems Sciences*, 9:256–278, 1974.

[47] R.M. Karp. *Reducibility Among Combinatorial Problems*, pages 85–103. Plenum Press, NY, 1972.

[48] H. Kellerer and U. Pferschy. A new fully polynomial approximation scheme for the knapsack problem. *APPROX'98*, LNCS 1444:123–134, 1998.

[49] B.J. Lageweg, J.K. Lenstra, and A.H.G. Rinnooy Kan. Minimizing maximum lateness on one machine: Computational experience and some applications. *Statist. Neerlandica*, 30:25–41, 1976.

[50] E.L. Lawler. Fast approximation algorithms for knapsack problems. *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 206–218, 1977.

[51] E.L. Lawler and J. Labetoulle. On preemptive scheduling of unrelated parallel processors by linear programming. *Journal of the ACM*, 25:612–619, 1978.

[52] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: Algorithms and complexity. *Handbook in Operations Research and Management Science*, 4:445–522, 1993.

[53] J. K. Lenstra and A. H. G. Rinnooy Kan. The complexity of scheduling under precedence constraints. *Operations Research*, 26:22–35, 1978.

[54] J. K. Lenstra, D. B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

[55] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Operations Research*, 1:343–362, 1977.

[56] H.W. Lenstra, Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.

[57] S. Martello and P. Toth. *Knapsack Problems.* Wiley, 1990.

[58] M. Mastrolilli. Combining arithmetic and geometric rounding techniques for knapsack problems. In *Proceedings of 1st International Workshop on Efficient Algorithms (WEA'01)*, volume LNCS 2138, pages 525–534, 2001.

[59] M. Mastrolilli. Compilable optimization problems: theoretical and practical implications. Technical report, 2001.

[60] M. Mastrolilli. Grouping techniques for one machine scheduling subject to precedence constraints. In *Proceedings of the 21st Foundations of Software Technology and Theoretical Computer Science*, volume LNCS 2245, pages 268–279, 2001.

[61] M. Mastrolilli. Efficient approximation schemes for scheduling problems with release dates and delivery times. *Journal of Scheduling*, accepted for publication.

[62] M. Mastrolilli and L. M. Gambardella. Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3:3–20, 2000.

[63] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 12:1–12, 1959.

[64] N. Megiddo and A. Tamir. Linear time algorithms for some separable quadratic programming problems. *Operations Research Letters*, 13:203–211, 1993.

[65] E. Nowicki. An approximation algorithm for the m-machine permutation flow shop scheduling problem with controllable processing time. *European Journal of Operational Research*, 70:342–349, 1993.

[66] E. Nowicki and S. Zdrzalka. A two-machine flow shop scheduling problem with controllable job processing times. *European Journal of Operational Research*, 34:208–220, 1988.

[67] E. Nowicki and S. Zdrzalka. A survey of results for sequencing problems with controllable processing times. *Discrete Applied Mathematics*, 26:271–287, 1990.

[68] E. Nowicki and S. Zdrzalka. A bicriterion approach to preemptive scheduling of parallel machines with controllable job processing times.

DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science, 63, 1995.

[69] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. In *Proceedings of 20th Annual ACM Symposium on Theory of Computing*, pages 229–234, 1988.

[70] M. Pinedo. *Scheduling : theory, algorithms, and systems*. Prentice-Hall, Englewood Cliffs, 1995.

[71] S. Sahni. Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM*, 22(1):115–124, 1975.

[72] Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, 1976.

[73] S. V. Sevastianov. On some geometric methods in scheduling theory: a survey. *Discrete Applied Mathematics*, 55:59–82, 1994.

[74] S.V. Sevastianov. Bounding algorithms for the routing problem with arbitrary paths and alternative servers. *Cybernetics (in Russian)*, 22:773–780, 1986.

[75] D.B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. *SIAM Journal on Computing*, pages 617–632, 1994.

[76] D.B. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, pages 461–474, 1993.

[77] M. Trick. Scheduling multiple variable-speed machines. *Operations Research*, 42:234–248, 1994.

[78] R.J.M. Vaessens. *Generalized Job Shop Scheduling: Complexity and Local Search*. PhD thesis, Eindhoven University of Technology, 1995.

[79] R.G. Vickson. Choosing the job sequence and processing times to minimize total processing plus flow cost on a single machine. *Operations Research*, 28:1155–1167, 1980.

[80] R.G. Vickson. Two single machine sequencing problems involving controllable job processing times. *AIIE Trans.*, 12:258–262, 1980.

[81] D.P. Williamson, L.A. Hall, J.A. Hoogeveen, C.A.J. Hurkens, J.K. Lenstra, S.V. Sevastianov, and D.B. Shmoys. Short shop schedules. *Operations Research*, 45:288–294, 1997.

[82] S. Zdrzalka. Scheduling jobs on a single machine with release dates, delivery times, and controllable processing times: worst-case analysis. *Operations Research Letters*, 10:519–532, 1991.